



Creating a Metric to Measure Software Flexibility in  
Object-Oriented Programming

Submitted for the Degree of  
Doctor of Philosophy  
At the University of Northampton

2020

Thomas Butler

© Thomas Butler 2020.

This thesis is copyright material and no quotation from it may be published  
without proper acknowledgement.

## Abstract

Business requirements inevitably change over time due to market shifts, law changes, new product launches or any number of other factors. The software being used by these businesses therefore also has to be adapted to meet the new requirements. How software is built has an impact on how easily and quickly the software can be changed to meet these new requirements. This thesis firstly identifies programming practices which make software difficult to adapt. To establish that these practices are genuinely considered "bad practice" a metric was created for grading the academic rigour of articles discussing a programming practice and this metric was used to perform meta-analyses of each practice identified, this meta-analysis methodology is based loosely on the methodology used for performing meta-analyses of clinical trials. The results of these meta-analyses demonstrated that the identified practices were widely considered bad practice by developers. Another metric was created to grade source code based on the frequency these bad practices appear in the code and give an overview of how flexible the code is. The aim of this metric is to facilitate learning for junior programmers while allowing more experienced programmers to evaluate the flexibility of software. A software tool was launched to enable users to evaluate and test the metric which was created. The metric was evaluated by comparison to alternative metrics and through user feedback.

# Table of Contents

<b>1. Introduction</b>	1
<b>1.1 Introduction</b>	2
<b>1.2 What is a bad practice?</b>	5
1.2.1 Case study: The Singleton Pattern	7
1.2.2 Anecdotal example	9
<b>1.3 Aims &amp; Objectives</b>	10
<b>1.4 Literature Review</b>	11
1.4.1 Code Smells	12
1.4.2 Software Metrics	12
1.4.2.1 Length of code	12
1.4.2.2 Object-Oriented Metrics.	13
1.4.3 How long is "too long"?	13
1.4.4 Software Quality Metrics	14
1.4.7 Academic vs Industry assessments of code quality	17
<b>1.5 Current Tools/Strageies</b>	21
1.5.1 List of tools for a single language	21
1.5.2 Relevant tools	26
<b>1.6 Methodology</b>	29
<b>1.7 Chapter Review</b>	31
<b>2. Aggregation of Bad Practices</b>	32
<b>2.1 Documentation of bad practices</b>	33
<b>2.2 Aims &amp; Objectives</b>	35
<b>2.3 Methodology</b>	36
2.3.1 Documenting Bad Practices	36
2.3.2 Storing bad practices and code samples	36
2.3.3 Traits & Severity	41
<b>2.4 Results</b>	43
2.4.1 Using the new keyword in a constructor.	43
2.4.2 Annotations for configuration	43
2.4.3 Global/static variables	43
2.4.4 Singletons	43
2.4.5 Inheritance	43
2.4.6 Service locators	43
2.4.7 Setter Injection	44
2.4.8 Static methods	44
2.4.9 Negative Traits	44
2.4.7 Table of bad practices	45
<b>2.5 Chapter Review</b>	47
<b>3. Demonstrating the practices really are "bad".</b>	48
<b>3.1 Introduction</b>	49
3.1.1 Background	49
3.1.2 Rationale	50
<b>3.2 Aims and Objectives</b>	52
3.2.1 Aims	52
3.2.2 Objectives	52
<b>3.3 Methodology</b>	53
3.3.1 Metric for comparing analytical rigour in programming articles	53
3.3.2. Meta-analysis	54

3.3.3. Collecting data .....	56
3.3.4 Additional considerations .....	57
3.3.5 Test methodology .....	58
<b>3.4 Results</b> .....	59
3.4.1 Singleton .....	59
3.4.2 Dependency Injection .....	61
Key findings - Dependency Injection .....	61
<b>3.5 Conclusion</b> .....	63
3.5.1 Key findings .....	63
3.5.2 Problems Encountered .....	63
3.5.3 Evaluation .....	64
3.5.4 Future Research .....	66
<b>3.6 Results for remaining bad practices</b> .....	66
3.6.1 Annotations .....	67
3.6.2 Global Variables .....	70
3.6.3 Inheritance .....	72
3.6.4 new in constructor .....	75
3.6.5 Service Locator .....	77
3.6.6 Static Methods .....	79
3.6.7 Setter Injection .....	81
<b>3.7 Meta-analyses overall conclusions</b> .....	83
3.7.1 Possible further research .....	84
<b>3.8 Chapter Review</b> .....	85
<b>4. Creating a metric</b> .....	86
<b>4.1 Introduction</b> .....	87
<b>4.2 Aims and Objectives</b> .....	88
4.2.1 Aim .....	88
4.2.3 Objectives .....	88
<b>4.3 Methodology</b> .....	89
4.3.1 Introduction .....	89
4.3.2 Software Size .....	89
4.3.3 Severity .....	90
4.3.4 What to grade .....	91
4.3.5 Grading range and visualisation .....	92
4.3.6 Grade Calculation .....	92
4.3.6 Model refinement .....	93
4.3.7 Further refinement .....	95
4.3.8 Project score .....	96
<b>4.4 Results</b> .....	98
4.4.1 Preliminary Evaluation .....	99
4.4.2 Conclusion .....	101
<b>4.5 Chapter review</b> .....	102
<b>5. Testing the metric by creating a tool</b> .....	103
<b>5.1 Introduction</b> .....	104
<b>5.2 Aims and Objectives</b> .....	104
<b>5.3 Design</b> .....	105
5.3.1 Web based or application. ....	105
5.3.2 Language choice .....	105
5.3.3 Specification .....	107
5.3.4 Report format .....	107
5.3.5 Methodology .....	108

5.3.6 Unit 1 - Utility class 1: Navigating code .....	108
5.3.7 Unit 2 - Utility class 2: Calculate namespace .....	109
5.3.8 Unit 4 - Project .....	110
5.3.9 Unit 5 - Scan for bad practices .....	111
5.3.10 Unit 6 - Combining the rules .....	112
5.3.11 Unit 7 - Metric .....	112
5.3.12 Unit 8 - Class Issues .....	113
5.3.13 Unit 9 - GUI .....	113
5.3.14 Unit 10 - Automated corrections .....	113
<b>5.4 Implementation</b> .....	113
5.4.1 Technical challenges .....	121
5.4.2 Known limitations .....	123
<b>6. Evaluation</b> .....	125
<b>6.1 Introduction</b> .....	126
<b>6.2 Evaluation techniques used by other software metrics</b> .....	126
6.2.1 Other academic approaches .....	127
<b>6.3 Evaluation techniques for this project.</b> .....	129
<b>6.4 Real developer evaluations</b> .....	129
<b>6.5 Compared to other metrics</b> .....	130
6.5.1 Methodology .....	131
6.5.2 Results .....	132
6.5.3 Conclusions .....	134
<b>6.6 Bad practice frequencies</b> .....	135
<b>6.7 User evaluation</b> .....	139
6.7.1 Results .....	141
6.7.2 Conclusions .....	159
6.7.3 User comments .....	163
<b>6.8 Project outputs</b> .....	164
<b>6.9 Project strengths</b> .....	165
<b>6.10 Project weaknesses</b> .....	166
<b>6.11 Research relevance and use-cases</b> .....	166
<b>6.12 Future improvements and limitations</b> .....	167
6.11.1 Future improvements/Limitations - Chapter 2 - Aggregation .....	167
6.12.2 Future improvements/Limitations - Chapter 3 - Meta-analysis .....	167
6.12.3 Future improvements/Limitations - Chapter 4 - Metric .....	168
6.12.4 Future improvements/Limitations - Chapter 5 - Tool .....	168
6.12.5 Future improvements/Limitations - Chapter 6 - Evaluation .....	170
<b>6.13 Chapter review</b> .....	170
<b>7. References</b> .....	173
<b>8. Appendices</b> .....	189
<b>Appendix I: Markdown extensions</b> .....	189
8.1.1 References .....	189
references.json format .....	190
8.1.2 Example code .....	191
<b>Appendix II. Raw JSON files describing bad practices</b> .....	193
8.2.3 Service Locator .....	193
8.2.4 Singleton .....	193
8.2.7 Object not initialised after constructor finishes (`initialize` and `set` methods) .....	193
8.2.10 Annotations for configuration .....	194
8.2.11 Use of static methods .....	194
8.2.12 Using `new` in constructor .....	195

8.2.13 Inheritance .....	195
8.2.14 Global/Static variables .....	195
<b>Appendix III. Full explanations of bad practices .....</b>	<b>197</b>
8.3.1 Service Locator .....	198
8.3.2 Singleton .....	200
8.3.3 Object not initliased after constructor finishes (`initialize` and `set` methods) ` .....	203
8.3.4 Annotations for configuration .....	208
8.3.5 Use of static methods .....	213
8.3.6 Using `new` in constructor .....	216
8.3.7 Inheritance .....	219
8.3.8 Global/Static variables .....	226
<b>Appendix IV. Full explanations of negative traits .....</b>	<b>230</b>
8.4.1 Broken encapsulation .....	230
8.4.2 Single Responsibility Principle .....	233
8.4.3 Unclear dependencies .....	234
8.4.4 Temporal Coupling .....	234
8.4.5 Law of Demeter .....	235
8.4.6 Tight Coupling .....	237
8.4.7 Global State .....	246
8.4.8 Unnecessary Coupling .....	246
8.4.9 Action at a Distance .....	247
<b>Appendix V. Paper: Seven Deadly Sins of Software Flxibility .....</b>	<b>249</b>
<b>Appendix VI. Meta-Analysis Raw Data .....</b>	<b>257</b>
Singleton .....	257
Dependency Injection .....	258
Annotations for configuration .....	259
Global Variables .....	260
Inheritance .....	261
Service Locator .....	263
Static Methods .....	264
Setter Injection .....	265
<b>Appendix VII. Paper: A Methodology for Performing Meta-analyses of Developers Attitudes Towards Programming Practices .....</b>	<b>267</b>
<b>Appendix VIII. Questionnaire questions .....</b>	<b>279</b>
<b>Appendix IX. Industry published article about the tool .....</b>	<b>282</b>
<b>Appendix X. Questionnaire results raw data .....</b>	<b>292</b>

# List of Figures

<b>Figure 1.1 Overview of academic/industry literature</b>	20
<b>Figure 2.1 Bad practice storage format</b>	39
<b>Figure 2.2 Reference storage format</b>	39
<b>Figure 3.1 Singleton results</b>	59
<b>Figure 3.2 Dependency Injection results</b>	61
<b>Figure 3.3 Singleton results</b>	65
<b>Figure 3.4 Dependency Injection Results</b>	65
<b>Figure 3.4 Meta-analysis results: Annotations</b>	68
<b>Figure 3.5 Meta-analysis results: Global Variables</b>	70
<b>Figure 3.6 Meta-analysis results: Inheritance</b>	73
<b>Figure 3.7 Meta-analysis results: Dependency Injection</b>	75
<b>Figure 3.8 Meta-analysis results: Service Locator</b>	77
<b>Figure 3.9 Meta-analysis results: Static Methods</b>	79
<b>Figure 3.10 Meta-analysis results: Setter Injection</b>	81
<b>Figure 3.11 Depth of discussion about each practice</b>	83
<b>Figure 4.1 Initial grade calculation</b>	93
<b>Figure 4.2 Refined grade calculation</b>	95
<b>Figure 4.3 Initial inheritance calculation demonstration</b>	95
<b>Figure 4.4 Refined class grade calculation</b>	95
<b>Figure 4.5 Project score calculation</b>	97
<b>Figure 4.6 Graph of results and number of classes</b>	100
<b>Figure 5.1 Sample Tokenizer usage</b>	108
<b>Figure 5.2 Sample Tokenizer usage (b)</b>	109
<b>Figure 5.3 Sample local class aliasing in PHP</b>	109
<b>Figure 5.4 Tool for resolving global class name</b>	110
<b>Figure 5.5 Project class usage</b>	111
<b>Figure 5.6 Rule interface</b>	111
<b>Figure 5.7 Sample rule implementation</b>	112
<b>Figure 5.8 Sample Insphpect instance</b>	112
<b>Figure 5.8 Code for generating the grade of a project</b>	113
<b>Figure 5.9 Screenshot of Insphpect home page</b>	115
<b>Figure 5.10 Screenshot of sample report</b>	116
<b>Figure 5.11 Screenshot of sample class</b>	117
<b>Figure 5.12 Screenshot of sample class with issue expanded</b>	118
<b>Figure 5.13 Screenshot of automated fix instructions</b>	119
<b>Figure 5.14 Screenshot of generated patch</b>	121
<b>Figure 5.14 Code prior to being rewritten</b>	121
<b>Figure 5.15 Code after being rewritten by Insphpect</b>	122
<b>Figure 5.16 Complex return statement</b>	122
<b>Figure 5.17 Complex return statement after being rewritten by Insphpect</b>	122
<b>Figure 5.18 Alternative complex return statement</b>	123
<b>Figure 5.19 Alternative complex return statement after being rewritten on the fly</b>	123
<b>Figure 6.1 Results of each tool, Normalised</b>	132

<b>Figure 6.2 Normalised Root Mean Square Deviation of Insphpect and Scrutinizer-CI</b> .....	135
<b>Figure 6.3 Mean frequency of bad practices across all projects</b> .....	138
<b>Figure 6.4 Question 1. How would you describe yourself as a programmer?</b> .....	141
<b>Figure 6.5 Question 2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.</b> .....	142
<b>Figure 6.6 Question 3. Do you use code reviews as part of your workflow?</b> .....	143
<b>Figure 6.7 Question 4. Do you use code review tools such as scrutimizer, phpmd, pmd, etc?</b> .....	144
<b>Figure 6.8 During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.</b> .....	145
<b>Figure 6.9 Question 6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply.</b> .....	146
<b>Figure 6.10 Question 7. Do you try to follow Object-Oriented best-practices when developing software?</b> .....	147
<b>Figure 6.11 Question 8. Do you actively try to avoid programming practices which go against best practice principles? For example, do you actively avoid global variables and singletons</b> .....	148
<b>Figure 6.12 Which, if any, programming practices do you actively avoid using tick all that apply, ignore any you are unfamiliar with</b> .....	149
<b>Figure 6.13 Question 11. The insphpect site is intuitive and easy to use</b> .....	151
<b>Figure 6.14 Question 12. How much do you agree with the statement:</b> .....	152
<b>Figure 6.15 Question 13. Do you agree with the recommendations made by Insphpect?</b> ...	153
<b>Figure 6.16 Question 14. How much do you agree with the following statement:</b> .....	153
<b>Figure 6.17 Question 15. How much do you agree with the following statement:</b> .....	154
<b>Figure 6.18 Question 16. How much do you agree with the following statement:</b> .....	156
<b>Figure 6.19 Question 13. Do you agree with the recommendations made by Insphpect? Senior Developers Only</b> .....	160
<b>Figure 6.20 How much do you agree with the following statement:</b> .....	161
<b>Figure 6.21 Question 13. Do you agree with the recommendations made by Insphpect? Senior Developers who use code reviews</b> .....	162
<b>Figure 6.22 Question 12. How much do you agree with the statement:</b> .....	162
<b>Figure 6.23 Question 14. How much do you agree with the following statement:</b> .....	163
<b>Figure 8.1 Service locator example 1</b> .....	198
<b>Figure 8.2 Initialize methods</b> .....	203
<b>Figure 8.3 Setter injection</b> .....	204
<b>Figure 8.4 Setter injection problem demonstration (a)</b> .....	205
<b>Figure 8.5 Setter injection problem demonstration (b)</b> .....	205
<b>Figure 8.6 Setter injection problem demonstration (c)</b> .....	205
<b>Figure 8.7 Setter injection solution</b> .....	207
<b>Figure 8.8 Annotations example</b> .....	208
<b>Figure 8.9 Annotations solution</b> .....	209
<b>Figure 8.10 Annotations example 2</b> .....	210
<b>Figure 8.11 Annotations example</b> .....	210
<b>Figure 8.12 Static methods example</b> .....	213



<b>Figure 8.13 Static methods solution</b>	213
<b>Figure 8.14 Static methods solution</b>	214
<b>Figure 8.15 Removal of static methods offers more flexibility</b>	214
<b>Figure 8.16 New in constructor example</b>	216
<b>Figure 8.17 New in constructor example 2</b>	216
<b>Figure 8.18 New in constructor solution</b>	217
<b>Figure 8.19 Dependency Injection</b>	217
<b>Figure 8.20 Inheritance example</b>	220
<b>Figure 8.21 Inheritance solution</b>	220
<b>Figure 8.22 Benefits of removing inheritance</b>	221
<b>Figure 8.23 Fragile base class example</b>	222
<b>Figure 8.24 Fragile base class problem</b>	223
<b>Figure 8.25 The diamond problem</b>	224
<b>Figure 8.26 Global variables example</b>	228
<b>Figure 8.27 Broken Encapsulation example</b>	230
<b>Figure 8.28 Demonstration of Broken Encapsulation issue (a)</b>	231
<b>Figure 8.29 Demonstration of Broken Encapsulation issue (b)</b>	231
<b>Figure 8.30 Operating on data outside of the scope it is defined in</b>	231
<b>Figure 8.31 Alternative approach using an interface</b>	232
<b>Figure 8.32 Using the interface from figure 2.33</b>	232
<b>Figure 8.33 Broken single responsibility principle</b>	233
<b>Figure 8.34 Demonstrating the single responsibility principle</b>	233
<b>Figure 8.35 Advantage of following the single responsibility principle</b>	234
<b>Figure 8.36 Temporal Coupling example</b>	235
<b>Figure 8.37 Law of Demeter example</b>	235
<b>Figure 8.38 Avoiding breaking the Law of Demter</b>	236
<b>Figure 8.39 Digging deeper into the object graph</b>	236
<b>Figure 8.40 Object graph required to test the code shown in figure 2.41</b>	237
<b>Figure 8.41 Example of Tight Coupling</b>	238
<b>Figure 8.42 No dependencies are visible externally</b>	238
<b>Figure 8.43 Example of loose coupling (a)</b>	238
<b>Figure 8.44 Example of loose coupling (b)</b>	239
<b>Figure 8.45 Real world example of tight coupling</b>	240
<b>Figure 8.46 Enhanced Signup example class</b>	240
<b>Figure 8.47 Enhanced Signup example class usage</b>	240
<b>Figure 8.48 Signup class using loose coupling</b>	241
<b>Figure 8.49 Signup class using loose coupling usage</b>	241
<b>Figure 8.50 Tight coupling with inheritance</b>	242
<b>Figure 8.51 The Signup class modeled using inheritance</b>	242
<b>Figure 8.52 Tight coupling using static methods</b>	243
<b>Figure 8.53 Using static methods to model the Signup class</b>	243
<b>Figure 8.54 Loosely coupled Dave class</b>	244
<b>Figure 8.55 Loosely coupled Signup class</b>	245
<b>Figure 8.56 Unnecessary Coupling example</b>	247
<b>Figure 8.57 Removing Unnecessary Coupling</b>	247



# List of Tables

<b>Table 1.1 QMOOD Metrics</b> .....	15
<b>Table 1.2 QMOOD Quality Metrics</b> .....	15
<b>Table 1.3 Review of static analysis tools</b> .....	22
<b>Table 2.1 Bad practice file and folder structure</b> .....	40
<b>Table 2.2 Trait file/folder structure</b> .....	41
<b>Table 2.3 Table of bad practices</b> .....	45
<b>Table 4.1 Bad practice severity ratings</b> .....	91
<b>Table 4.2 Results table for 20 projects</b> .....	98
<b>Table 5.1 Pros/Cons of a web-based tool</b> .....	105
<b>Table 6.1 Frequency of bad practices</b> .....	136
<b>Table 6.2 Frequency of bad practices (average per class)</b> .....	136
<b>Table 6.3 Which bad practice caused grade reductions</b> .....	138





# 1. Introduction

## 1.1 Introduction

This basis for this research can be summed up with two mundane observations:

1. Business requirements change over time.
2. Programmers are not clairvoyant.

Whether due to changes in the law like GDPR or VAT rates, new product launches, response to competition, opening in new markets, etc, regardless of how large or small a business is, the requirements will change over time.

From these observations it can be inferred that programmers know that business software will need to be changed over time to fit new requirements but they have no way of knowing the nature of those changes or when changes will be needed. Therefore programmers need to build software that is capable of being modified in ways they may not have anticipated at the start of the project.

A survey of 1,000 developers by Stripe (2018) found that half of a programmer's time is wasted dealing with existing "bad code". This was extrapolated to estimate that "bad code" costs \$300 billion USD every year worldwide. In addition, they concluded that if developers worked more efficiently it could be worth 300 trillion dollars over the next decade.

Stripe did not define "bad code" when surveying the developers and it was left to the developer being questioned what "bad code" meant to them. This research aims to document what "bad code" is and explain how to identify and avoid it, before developing a metric to grade software on its flexibility.

Like any engineering discipline, when developing software there are potentially hundreds or thousands of ways writing code to achieve a particular outcome, even for a relatively small and simple project. Each of these potential solutions is architecturally different and will come with its own positives and negative aspects in relation to other solutions. Can these positives and negatives be measured? And if they can be measured, can these positive/negative traits be automatically detected and the two different projects compared?

There are many programming techniques which are considered "bad practice" and commonly touted as the wrong way of programming. Two widely discussed and highly prominent examples are *global variables* (Sayfan, n.d.; Koopman, 2010; Svennerberg, 2012; Zakas, 2006; Ferreira, 2013; IBM, 2012; Crockford, 2006; Hevery, 2008) and the *singleton pattern* (Densmore, 2004; Radford, 2003; Yegge, 2004; Ronacher, 2009; Brown, 2013; Kofler, 2012; Hevery, 2008; Hevery, 2008; Sayfan, n.d.; Weaver, 2010) which are regarded as bad practices and have many detractors who advocate

avoiding their use and using alternatives in their place.

Along with global state and singletons, there are numerous other bad practices which have varying levels of discussion and documentation surrounding them. Books such as *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* (Brown et al, 1998), *Refactoring: Improving the Design of Existing Code (Object Technology Series)* (Fowler et al, 1999) and *Code Complete: A Practical Handbook of Software Construction* (McConnell, 2004) are dedicated to teaching software engineers how to identify and avoid these bad practices and advice on what they should be doing instead and why. Other books such as *Design Patterns: Elements of Reusable Object-Oriented Software*. (Gamma et al, 1994) focus on best practices rather than highlighting bad practices.

From a programmer's point of view software is poorly written if it is difficult to work with. Although "difficult to work with" may be subjective, in Object-Oriented Programming, the "bad practices", "anti-patterns" and "code-smells" identified by authors are labeled "bad practice" because they cause the code to be difficult to modify. A program that is difficult to change is therefore labelled inflexible.

Flexibility is not a specific component of the software, there are no lines of code that can be written to "add flexibility". However, some programming practices have been described as "flexible" or "inflexible" (Eden et al, 2006; Hevery, 2008), flexibility is like the centre of gravity of a car, there is no single component or line of code which conveys this property, but the flexibility, like centre of gravity, is a function of all the parts of the system and how they are connected. There are several ways flexibility can be inferred, for example:

- Needing as few changes as possible to add new functionality.
- If the software can easily have parts replaced entirely without modifying other parts of the code. For example, replacing a gearbox in a car without needing to modify the engine.
- If the existing parts can be taken and re-structured to make the program act differently. This is the difference between a toy ship made of Lego and a ship in a bottle. The Lego ship can be taken apart and turned into a house or a rocket. Doing the same with the ship in the bottle is considerably more difficult.
- Keeping everything self-contained. If a change in one place in the software accidentally causes a change elsewhere, this can lead to subtle bugs and problems. This is informally referred to as "Action-at-a-distance" (Wenger, 1995).



These metrics are imprecise, vague, nuanced and require a high level of understanding of both the problems that cause inflexibility and the code being analysed. The primary goal of this research is to create a new metric for measuring flexibility by analysing the number of bad practices that exist in a piece of software. The more of these bad practices a piece of software has, the less flexible it is. However, not all "bad practices" relate directly to flexibility. For the purposes of this research, the term "bad practice" will be used to describe a programming practice that hinders flexibility in some way. Other "bad practices" which relate to metrics other than flexibility such as performance or user interface design are beyond the scope of this research.

Given any two pieces of software, it could be claimed that one is "better designed" from the programmer's point of view by looking at how many bad practices it contains. Fowler *et al* (1999) and Hevery (2008) have demonstrated that design patterns in code which cause side-effects and negative consequences can be identified using a step-by-step process. With an adequate model, it would be possible to construct a scoring system that could be applied to any piece of software and allow the user to get an idea of how easy the software is to work with, extend or make changes to.

## 1.2 What is a bad practice?

"Bad practices" are defined as such because they make software difficult to *maintain*. Maintenance issues are caused by a number of practical problems including:

- Lack of documentation
- Incorrect or insufficient unit tests
- Poor or no code comments
- Poorly structured code (flexibility)

Each of these points could be used to measure source code maintainability from the perspective of a programmer. The first three are external to the code itself and can be identified fairly quickly by anyone working on the project.

The last point, flexibility, is more nuanced as it is a property deeply embedded within the code. The way the code is structured and the way that different parts of the code interact have a bearing on how easy it is to change. This research is focused entirely on flexibility and other issues such as documentation and unit test coverage are beyond its scope.

*Flexibility has been defined as a desirable trait since the earliest days of software engineering*

---

Eden *et al* (2006)

There are different methods of defining a "bad practice". Slow or buggy code could be declared "bad". However, when developers refer to "bad code" they are usually talking about code that is difficult to maintain (Fowler *et al*, 1999; Martin, 2008). This research focuses entirely on flexibility and any reference to "bad practice" will refer to a practice which makes the code more difficult to modify or reuse than an alternative approach.

Flexibility is desirable because it helps with reuse and maintenance. If a component can be used in multiple situations on different projects, it's better than a component that is specific to a project and cannot be easily modified to work in a different way. The main reason, the flexible component is "better" is because it saves developers time and can help lower the costs of software development.

Making software flexible is a very difficult task and can have several repercussions on the project:

- Increased design/development time, the flexibility has to be designed into the program and the program built to follow the design (Although this upfront flexibility can often save time

later on (Fowler, 2015))

- Increased complexity, if a piece of code can handle different use-cases it is often more complex than a piece of software which can handle only one very rigid task.

Because of this, there is often a trade-off between time/complexity and flexibility. Discussions around *best practices* often turn into discussions about when this trade-off is worthwhile (Sanders, 2013; Atwood, 2007; Funaro, 2009). This research does not attempt to weigh in on the discussion about pros/cons of flexibility. The goal of this research is to provide a metric for measuring flexibility in software, not to calculate whether the flexibility is required or not for any given project.

Practices such as *global variables* have been widely identified as impacting the flexibility of code going back at least as far as Wulf *et al* (1973). Others have highlighted problems with their use since (Sayfan, n.d.; Koopman, 2010; Svennerberg, 2012; Zakas, 2006; Ferreira, 2013; IBM, 2012; Crockford, 2006; Hevery, 2008), however there are many other bad practices which have been identified as the complexity of programs and development methodology grows. For example Hevery (2008) has identified several bad practices which have emerged due to a shift towards Test-Driven-Development (TDD) among developers. These practices were still problematic before the widespread use of TDD, however because TDD requires an extra level of flexibility, practices which limit flexibility become apparent to the developer considerably faster than when using alternative development methodologies.

As programming methods such as TDD and Agile programming have emerged and gained popularity, the importance of flexibility of code has been magnified. For example, the use of mock objects in TDD requires that code is separated and *loosely coupled*, making it easier to test.

*Applications with loosely coupled components, on the other hand, are modular, flexible, and easily tested with unit tests.*

---

Weiskotten (2006)

Programming methodology trends like Test Driven Development have lead to requirements for code to be *loosely coupled* so that it can be easily tested, which in turn has lead to practices that make code *tightly coupled* being considered "bad practice" due to making tests more difficult to write and produce less useful results. However, this flexibility affects more than just unit testing, the same lack of flexibility exists in the program, it's often the case that unit tests are the first situation that highlights it. Hevery (2008) wrote *Guide: Writing Testable code*. It is no coincidence that all of the items in this guide relate to flexibility.

Along with *tight coupling*, there are other traits which have been identified as limiting to flexibility. These include: *action at a distance*, *breaking the Law of Demeter* (also known as *Digging into collaborators* (Hevery, 2008) and *Tell, Don't ask* (Fowler, 2013)), *brittle code*, *breaking encapsulation* and *breaking the single responsibility principle*. These traits are not detectable in and of themselves but the result of implementing a bad practice. Static methods introduce tight coupling but tight coupling can also be introduced by inheritance and using the `new` keyword.

Bad practices are considered as "bad" because they introduce one or more of these negative traits in the code and these traits limit flexibility. This can be confusing to developers, as it's not immediately clear why the practice has been considered "bad practice" without understanding why the underlying traits it introduces limit flexibility.

Some bad practices are easily discovered by junior developers while others only start to appear in larger applications as complexity increases. For example, global variables are widely considered "bad practice" and it's difficult to get far in a programming career without being informed of the dangers of global variables or falling into some of the problems they cause (Wulf *et al*, 1973) and working it out yourself.

This is down to several factors: Global variables are very easy to use/understand and the problems they create become apparent very quickly, even to novice developers. This leads to a lot of discussion around the topic and considerable awareness among developers.

A "bad practice" can be summed up as a practice which limits the flexibility of the code.

### 1.2.1 Case study: The Singleton Pattern

Bad practices become bad practices through experimentation. Developers use the practice in their code and then run into problems it causes.

*for it is true that global variables are often demonised and more recently the Singleton has befallen the same fate.*

---

Knack-Nielsen (2008)

After global variables, the most talked about and often identified bad practice is the Singleton Pattern (Knack-Nielsen, 2008). This is likely because the pattern itself gained popularity after being demonstrated in the popular book *Patterns of Enterprise Application Architecture* (Fowler, 2002). Since then the pattern has been described as bad practice and an "anti-pattern" by numerous authors (Densmore, 2004; Radford, 2003; Yegge, 2004; Ronacher, 2009; Brown, 2013; Kofler, 2012; Hevery, 2008; Hevery, 2008; Sayfan, n.d.; Weaver, 2010) and can be seen being discouraged within online

communities with them generally being discouraged when brought up: (Reddit, 2013; Adobe, 2013).

The singleton has become regarded as a bad practice by most developers. This is for several reasons that don't apply to many other bad practices:

- The singleton was one of the patterns mentioned in two very popular and highly referenced books: *Design Patterns: Elements of Reusable Object-Oriented Software*. (Gamma et al, 1994) and *Patterns of Enterprise Application Architecture* (Fowler, 2002). This caused widespread knowledge of the pattern and it took some time until the issues it causes were discovered and documented.
- The pattern was given a formal name and is easy to identify
- The pattern has been considered bad practice for over a decade (Densmore, 2004)
- The problems it causes are severe compared to other more subtle bad practices, making it easier to demonstrate as there are multiple issues caused. This makes discovering the problems a simpler task for people using the pattern. They only have to discover one of the problems before realising the issues it introduces.

Because of the abundance of the pattern's usage, how long ago the pattern was recognised as bad practice, formal name and the severity of the issues caused there is significantly more discussion surrounding the Singleton Pattern compared to other bad practices.

On the other hand, the bad practice of "Digging into collaborators" (Hevery, 2008) is considerably more subtle and difficult to understand. This limits the number of people talking about it, as only experienced developers will understand the implications, this is further complicated because the same bad practice can have multiple names. For example "Digging into collaborators" has also been called "Law Of Demeter violation" (Brock, 2000) and "Tell, Don't ask" (Fowler, 2013).

Another problem surrounding discussion of these practices is that several people such as Seeman (2010) and Hevery (2008) have identified Service Locators as a bad practice, however a service locator is a specific implementation of "digging into collaborators" that has been named and become widespread.

How much these bad practices are talked about will depend on how often they are used and how easy they are to implement. These bad practices are seldom analysed in academia because they

tend to affect large scale software projects and contain nuances that are only relevant to people working on code daily. Because of this, most of the analysis of these practices has been done by leading industry experts such as Misko Hevery, a programming coach at Google, and consultants such as Martin Fowler who have written prominent books in the area.

### 1.2.2 Anecdotal example

A web development agency, Media-Web Solutions, had developed a website for a recruitment agency and had been maintaining it for several years when the client asked for a change to the website:

*We have just taken over another smaller recruitment agency and so we've inherited their website. We want to keep their brand and website as it's well known in their niche.*

*Some of the jobs we post would be useful on both websites. Can you make it so that when a job is posted on the new companies website there is a checkbox that also posts the job to our site? And visa versa?*

---

The same change was needed on both sites: When posting a job, add a checkbox to cross-post the job on the other website.

On the site that Media-Web had created for the client initially this was a simple task: the developer created a new database connection instance to connect to the other website and insert the job as a record into both databases.

However, the site that had been inherited used a singleton for the database connection because the original author of the website had assumed that only one database connection would ever be needed.

Two sets of code with the same modification needed, on one website the change was simple, on the other it was made needlessly difficult because of a design decision made that impeded the flexibility of the code.

## 1.3 Aims & Objectives

1. Publish an extensible list of bad practices in a format that can be easily extended by other academics or developers and embedded in third party documentation/applications.
2. Demonstrate that "bad practices" are "bad" on more than just a subjective level. E.g. practical examples and/or show that different developers independently come to the same conclusions.
3. Create a metric that can be used to analyse and grade the quality of source code based on the frequency of known bad practices.
4. Develop a proof of concept software tool that implements the metric and can analyse source code for bad practices.
5. Evaluate the metric by asking developers their opinion on the results and compare results to results of other similar tools.
6. Explore the possibility of extending the tool to allow automated source code correction of identified bad practices.

## 1.4 Literature Review

In the seminal book *Refactoring: Improving the Design of Existing Code (Object Technology Series)* (Fowler et al, 1999), Fowler et al (1999) coined the term "code smell" to refer to a piece of code that "stinks" and exhibits characteristics of "bad design". A specific definition of "code smell" is not given, only several examples. For example, the first "smell" described is duplicated code. Even a junior developer will see duplicated code and understand that it causes maintainability problems. If the logic has to change, it must be changed in (and the developer must remember to change) multiple locations.

Fowler et al (1999) go on to define dozens of other code smells including *Long Method*, *Large Class*, *Long Parameter List*, *Shotgun Surgery*, *Switch Statements* and *Inappropriate Intimacy*. The "smells" defined are based on the authors own industry experience and do not have any academic citations to back them up, only example code. However, *Refactoring: Improving the Design of Existing Code (Object Technology Series)* (Fowler et al, 1999) has been cited over 9,000 times in academic works (Google Scholar, n.d.).

One of the limitations of the definitions given by Fowler is that they are nebulous. It is up to the reader to decide where the lines are drawn. For example, the *Large Class* smell is defined as:

*When a class is trying to do too much, it often shows up as too many instance variables.  
When a class has too many instance variables, duplicated code cannot be far behind.*

---

And *Inappropriate Intimacy* as:

*Sometimes classes become far too intimate and spend too much time delving in each others' private parts. We may not be pruders when it comes to people, but we think our classes should follow strict, puritan rules*

---

A reader may be left with two questions that Fowler et al (1999) does not attempt to answer:

1. How is "length" measured? If number of lines is used, does that include comments and whitespace? Does brace position affect line count?
2. How long is "too long"? How much is "too much" and "too intimate"?

Although Fowler et al (1999) alludes to these answers, the descriptions are generally vague. For example:



*As with a class with too many instance variables, a class with too much code is prime breeding ground for duplicated code, chaos, and death. The simplest solution (have we mentioned that we like simple solutions?) is to eliminate redundancy in the class itself. If you have five hundred-line methods with lots of code in common, you may be able to turn them into five ten-line methods with another ten two-line methods extracted from the original.*

---

Which suggests somewhere between ten and 100 lines is too long and that lines is the metric to use for measurement, is that counting whitespace and comments or not? How does brace position affect this count?

While Fowler *et al* (1999) provides a good starting point for "code smells", the vague definitions alone are not suitable for metrics that assess software quality as any such metric would need to define "too long" in a quantifiable way.

Fowler *et al* (1999) states that:

*One thing we won't try to do here is give you precise criteria for when a refactoring is overdue. In our experience no set of metrics rivals informed human intuition.*

---

### **1.4.1 Code Smells**

A systematic literature review of *code smells*, *anti-patterns* and *bad practices* by Sabir *et al* (2018) that looked at 78 papers (filtered by relevance from 13,769 total papers) identify that a total of 56 smells that have been reported in academic literature.

Despite the widespread labelling of the singleton pattern as a bad practice by industry developers, it is a notable omission from this list. One of the current issues with the literature is a disconnect between academia and industry.

### **1.4.2 Software Metrics**

There are many different metrics which can be used to measure various aspects of source code. These are often used as indicators for software quality. Several relevant metrics are outlined below.

#### **1.4.2.1 Length of code**

Several attempts have been made to provide metrics for size of source code. The most crude metric is Source Lines of Code (SLOC). Although there are attempts (Nguyen *et al*, 2007) to define a standard, there is no widely adopted definition of SLOC.

For use in assessing software quality and answering "how long is too long?" SLOC is poor choice as

different programming and commenting styles could result in the same logic being significantly different numbers of lines even using the same programming language. For example, brace position, whitespace, comments can all affect the number of lines in a file.

A more consistent metric is *Cyclomatic Complexity*, the total number of independent linear paths within a block of code (McCabe *et al*, 1976). A block of code with no control structures has a complexity of one, a block of code with a single of statement has a complexity of two. Unlike SLOC, code can be written with different brace and commenting styles without affecting the cyclomatic complexity.

SLOC and Cyclomatic Complexity are used for general programming, however, when using Object-Oriented Programming, there are other measurements of size include the number of classes, and number of methods in a class (Chidamber *et al*, 2007).

### 1.4.2.2 Object-Oriented Metrics.

Outside of the study of *anti-patterns* and *code smells*, metrics have been used of statistical analysis of OOP source code. One of the first suites of metrics was developed by (Chidamber *et al*, 2007) including:

- Weighted Methods Per Class (WMC), the number of methods defined in a class
- Depth of Inheritance Tree (DIT), the number of levels of inheritance per class
- Number of Children (NOC), the number of child classes a class has
- Coupling between objects (CBO), the number of other classes that a class depends on
- Response for a Class (RFC), the number of external methods that are called from methods inside the class
- Lack of Cohesion in Methods (LCOM), a measure of cohesion

Although (Chidamber *et al*, 2007) did not specifically create the metrics for assessing code quality, they observe that the metrics could be used for determining reusability:

*Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse.*

---

However, like Fowler *et al* (1999), leave it up to the individual to determine what is meant by "large numbers of methods". These metrics are statistical tallies and on their own do not offer any insight into software quality, despite statements that "too much" can be detrimental (Fowler *et al*, 1999).

### 1.4.3 How long is "too long"?

Attempts have been made to define how long methods and classes should be. For example Grady

*et al* (1994) suggests limiting the length of a method to 14 cyclomatic complexity. A number derived by plotting the complexity against how frequently the code is updated. Industry tool Scrutinizer-CI (n.d.) imposes a limit of 5 complexity per method and 35 per class for an A grade, though the reasoning for this threshold is not disclosed.

Fontana *et al* (2015) notes that:

*Many of the available smell detection tools implement a metrics-based approach, with 'fixed' or configurable threshold values. In some cases, tool providers using 'fixed' threshold values do not provide a clear rationale on how these thresholds have been devised*

---

Fontana *et al* (2015) uses existing systems to determine thresholds for various metrics in an attempt to provide a repeatable way of answering "how long is too long?" through statistical analysis of existing source code to work out averages. One conclusion they draw is that 47 methods is too many to have in a class. Although there is reasoning to back this number up, it is still a mostly arbitrary cut off point. A class with 46 methods may not be any more maintainable than a class with 47.

Although it has been shown that as a unit of code grows, the maintainability of code decreases (Yamashita *et al*, 2013), efforts such as these to quantify "too long" are indicative at best.

#### **1.4.4 Software Quality Metrics**

A survey of Object Oriented Quality Metrics by Neelamegam *et al* (2009) identified the following metrics used for assessing software quality:

- MOOSE (Metrics for Object-Oriented Software Engineering). An extension of the CK Metrics Model by the same authors.
- MOOD (Metrics for Object-Oriented Design). A set of statistical metrics which can be used to analyse various aspects of Object-Oriented Design
- QMOOD (Quality Model for Object-Oriented Design), a set of metrics specifically designed for assessing software quality.

Neelamegam *et al* (2009) concludes that:

*We surveyed a group of desirable properties for OOD quality models, and then we used them to compare the presented OOD quality models. Based on this comparison, we conclude*

that the QMOOD suite is the most complete, comprehensive, and supported suite.

---

Although each quality model is unique and use different metrics the underlying principle is the same: Source code is scanned and statistical analysis is used to assess quality. For example, QMOOD uses the following metrics:

**Table 1.1:** QMOOD Metrics

Design property	Metrics
Design Size	Design Size in Class(DSC)
Hierarchies	Number of hierarchies(NOH)
Abstraction	Average Number of Ancestors(ANA)
Encapsulation	Data Access Metrics (DAM)
Coupling	Direct Class Coupling(DCC)
Cohesion	Cohesion Among Methods in Class(CAM)
Composition	Measure of Aggregation(MOA)
Inheritance	Measure of Functional Abstraction(MFA)
Polymorphism	Number of Polymorphic Methods(NOP)
Messaging	Class Interface Size(CIS)
Complexity	Number of Methods(NOM)

Table 1.1 shows the metrics used by QMOOD, reproduced from Neelamegam *et al* (2009).

Like Chidamber *et al* (2007), the metrics are purely statistical with the inclusion of some averages to allow comparisons between classes. QMOOD takes this a step further and defines the characteristics *Reusability, Flexibility, Understandability, Functionality, Extendability* and *Effectiveness* as defined below.

**Table 1.2:** QMOOD Quality Metrics

Quality Attribute	Index Computation Equation
Reusability	$-0.25 * \text{Coupling} + 0.25 * \text{coupling} + 0.5 * \text{Messaging} + 0.5 * \text{Design Size}$
Flexibility	$0.25 * \text{Encapsulation} - 0.25 * \text{Coupling} + 0.5 * \text{Composition} + 0.5 * \text{Polymorphism}$
Understandability	$-0.33 * \text{Abstraction} + 0.33 * \text{Encapsulation} - 0.33 * \text{Coupling} + 0.33 * \text{Cohesion} - 0.33 * \text{Polymorphism} - 0.33 * \text{Complexity} - 0.33 * \text{Design Size}$
Functionality	$0.12 * \text{Cohesion} + 0.22 * \text{Polymorphism} + 0.22 * \text{Messaging} + 0.22 * \text{Design Size} + 0.22 * \text{Hierarchies}$
Extendibility	$0.5 * \text{Abstraction} - 0.5 * \text{Coupling} + 0.5 * \text{Inheritance} + 0.5 * \text{polymorphism}$
Effectiveness	$0.2 * \text{Abstraction} + 0.2 * \text{Encapsulation} + 0.2 * \text{Composition} + 0.2 * \text{inheritance} + 0.2 * \text{Polymorphism}$

Table 1.2 shows the software quality characteristics defined by QMOOD, reproduced from Neelamegam *et al* (2009).

The weightings they have chosen (For example, -0.25 for coupling in the *Reusability* metric) are arbitrary and the authors state "Weights can be experimented with to best reflect organizational objectives".

Neelamegam *et al* (2009) discusses the differences between MOOD, MOOSE and QMOOD. For the purposes of this research, the differences between the metrics are not relevant as all three metrics use a similar approach and have the same limitation with defined thresholds.

#### 1.4.4.1 Problems with statistical approaches such as MOOD

Due to the purely statistical nature of these metrics they are only indicative. Someone looking at the metrics may be able to infer some information about complexity and the overall software design the usefulness is limited:

1. They are poorly defined. The *Flexibility* metric in QMOOD does not take *Inheritance* into account. In most programming languages, overuse of inheritance creates a very rigid, inflexible design (Holub, 2010). Global variables also lead to inflexible code (Hevery, 2008). Using QMOOD, two pieces of software could receive the same score even though one uses inheritance and global variables while the other does not.
2. There is no directly actionable outcome. After calculating the metric, there is very little indication of what a developer could, or should, do to improve the quality of the source code, only an indication of where in the code improvements could possibly be made to lower the numbers.
3. Weightings are arbitrary. The score for each characteristic is useful for comparing different systems but the numbers themselves are mostly meaningless on their own.
4. Regardless of which model is used, trying to answer the question "how long is too long?" will always result in a cut off point that cannot be applied universally. When using 14 as Grady *et al* (1994) suggests, a 15th logical statement is the difference between code labelled "flexible" and code labelled "inflexible".
5. Different metrics will give different grades due to different thresholds.

### 1.4.7 Academic vs Industry assessments of code quality

While the academic focus has been on automated detection of code smells (Fontana *et al*, 2015) such as "god classes" and "long methods", industry code reviewers look at more than just statistical metrics (Hevery, 2008) to assess code quality.

Hevery (2008), a programming coach at Google, looks for specific bad practices such as the Singleton Pattern and using the new keyword in constructors. Although these practices are commonly discussed in industry, none of the software quality metrics used in academic works take these into account.

In 2008, Knack-Nielsen (2008) while writing for the website sitepoint.com, a popular resource among industry web developers, observed that:

*for it is true that global variables are often demonised and more recently the Singleton has befallen the same fate*

---

Other industry sources (Densmore, 2004; Radford, 2003; Yegge, 2004; Ronacher, 2009; Brown, 2013; Kofler, 2012; Hevery, 2008; Hevery, 2008; Sayfan, n.d.; Weaver, 2010) make the same calls to avoid using the pattern in the same manner as global variables because it introduces similar problems. Online forums and question answer sites like stackoverflow.com used by developers in industry can be seen to have the same negative opinion of the pattern (Vogt, 2008; Reddit, 2013; Adobe, 2013).

However, among academic literature the singleton pattern is never mentioned as a bad practice, anti-pattern or code smell. In 2018, ten years after the observation by Knack-Nielsen (2008) regarding industry's negative opinion of the singleton pattern, a comprehensive review of the academic literature (Sabir *et al*, 2018) identifies 56 "code smells", "bad practices" and "ant-patterns". The singleton pattern is not included in their list as it does not currently appear in the academic literature. The list does, however, contain global variables which are considered bad practice in the same manner as the singleton among industry developers (Cosentino, 2013; Hevery, 2008).

Google Programming Coach Hevery (2008) notes:

---

*There is a price to pay for such a JVM Singleton, and that price is flexibility and testability.*

---

Academic sources often contradict this. For example, using source code metrics to analyse the maintainability of the singleton pattern Abdullah (2017) writes:

---

*Maintainability metrics for implementation of Singleton is illustrated in table 6.1. It is observed that all the maintainability metrics for Singleton Design Pattern are within limit of maintainability thresholds and good scores for maintainability are observed for Singleton Design Pattern*

---

Despite specifically discussing maintainability and the singleton pattern, Abdullah (2017) never identifies that the singleton pattern has been shown to introduce maintainability issues by industry developers. This demonstrates that knowledge from industry is missing in academic literature and the limitations of statistical based metrics for source code analysis.

In *Detecting Software Bad Smells from Software Design Patterns using Machine Learning Algorithms* (Kaur *et al*, 2018) source code metrics are used analyse design patterns including the singleton notes that:

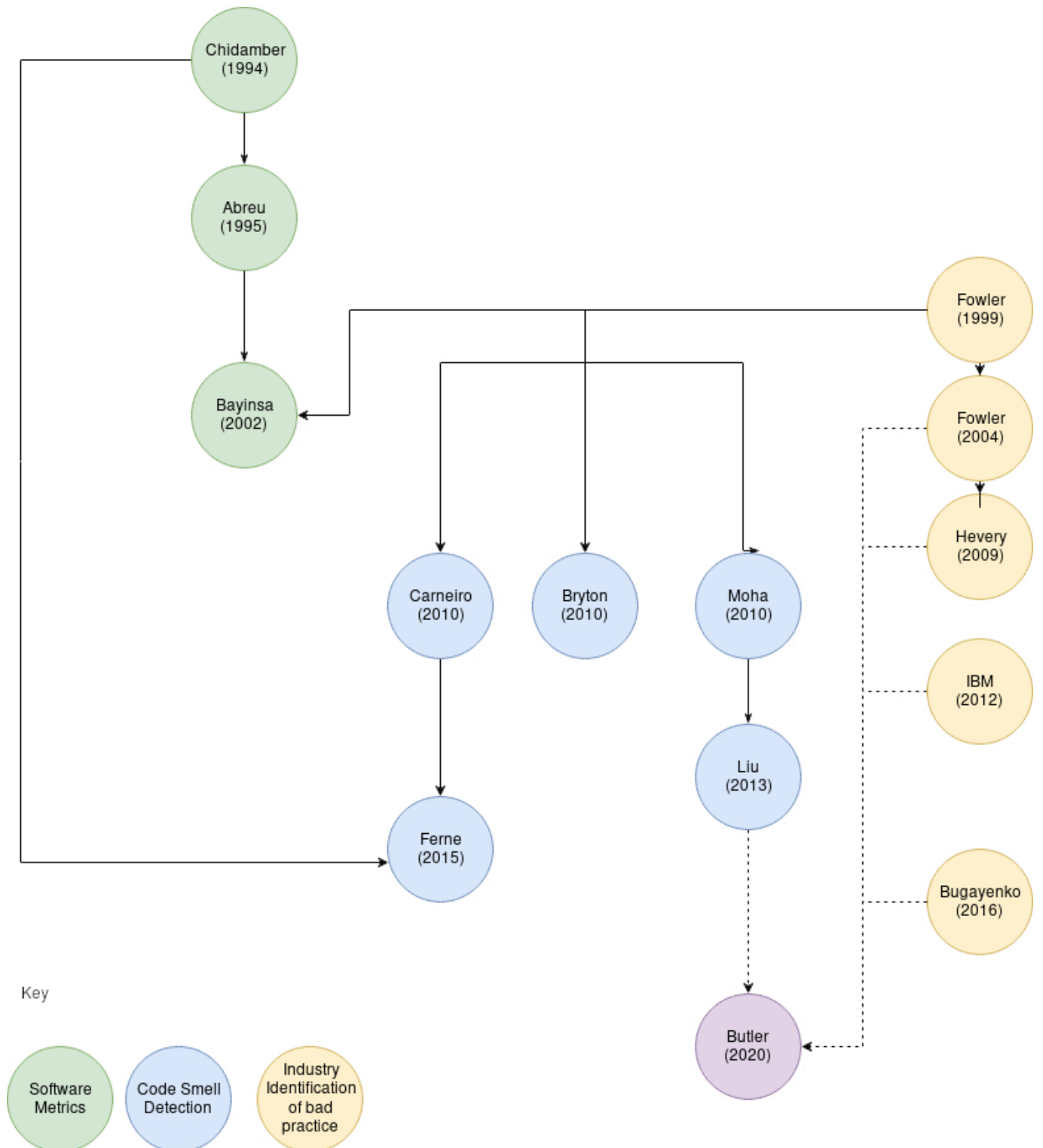
---

*In case of design patterns state strategy, adapter command, factory method and singleton lack bad smells.*

---

The paper is specifically discussing code smells but, like Abdullah (2017) never touches upon the fact industry experts discourage the singleton pattern and consider it a bad practice/code smell.

The underlying reason for this appears to be that the list of smells used in academic work originate from the industry developer Fowler *et al* (1999) and have not been added to since. Subsequent research has built on prior academic research leading to a fork in knowledge between academics and industry developers.





## Figure 1.1: Overview of academic/industry literature

Figure 1.1 shows an overview of the relationships between the literature in Academia and Industry. This chart contains only the most relevant literature to give an indication of where this research fits. A more detailed literature review is performed in this section.

Academia reached "Smell detection" via the route of software metrics like cyclomatic complexity and coupling. These were then built on using threshold based "smell detection" where too much coupling/complexity is flagged as a code smell. This is partially influenced by Industry work by Fowler *et al* (1999).

Although this approach is used in industry (Scrutinizer-CI, n.d.), industry code reviews tend to be based on identifying coding practices which make code difficult to work with (Hevery, 2008). These have been identified by programmers at companies like Google (Hevery, 2008) and IBM (IBM, 2012) as well as high profile software consultants (Bugayenko, 2016) from first hand experiences of falling into traps caused by using these programming practices.

## 1.5 Current Tools/Strageies

According to Ivo *et al* (2009) formal review and inspections of code were "recognized as important to productivity and product quality" as far back as the 1970s. Ivo *et al* (2009) also discusses the uses and merits of various static analysis tools which have been used to detect bugs and potential security vulnerabilities in code.

One such tool is FxCop by Microsoft which can be used to detect bugs, security issues and potential performance issues in code. To a very limited degree FxCop and other tools identified by Ivo *et al* (2009) detect design based issues but these tend to be very primitive such as duplicate code and variables which are declared and never used (MSDN, 2013).

There are many tools designed to detect genuine bugs, security flaws and programming practices which cause code to work in a manner it's not intended but none of these appear to target the more abstract and philosophical best practices that come from OOP theory that limit flexibility , hindering further development of the software.

### 1.5.1 List of tools for a single language

There are numerous tools for analysing code in different ways. Table 1. is a comprehensive list of static analysis tools for a single language. The scope of available tools for just one programming language demonstrates how widespread and varied static analysis tools are.

1. PHP is mature and widespread enough that many code analysis tools to have been developed for it.
2. There are thousands of open source libraries available for PHP (Packagist, n.d.) which gives large body of sample inputs for any potential future further analysis of available tools.
3. This list had already been compiled by (Seguy, 2014) and has been included here solely to show the scope of tools available for a particular language (other languages have other similar tools (Analyis Tools, n.d.)).
4. It is intended that the final stage of this project use PHP for analysis for reasons outlined in section 5.3.2.

The table includes a brief description, whether they produce a grade (A-F, 0-10, etc) and whether they look for any Object-Oriented Bad practices and whether they offer automated fixes.

This overview is based upon the list compiled by Seguy (2014).

**Table 1.3:** Review of static analysis tools

Name	What is detected?	Looks for OOP bad practices?	Provides grade?	Automate fixes?	Notes	Relevant to this research?
AppChecker	Syntax errors, undefined variables, common mistakes like assignment instead of comparison, code which can't be called, division by zero	No	Yes	No	Russian Language	No
churn-php	Statistical analysis through tallies: Cyclomatic complexity, commits	No	Yes	No		No
Eir	Security issues	No	No	No		No
Exkat	Comprehensive static analysis with over 400 rules, looks for deprecated/removed PHP features, security issues, performance issues, debugging instructions left in production code. Does not look for OOP bad practices like singletons.	No	Yes	No		No
jscpd	Duplicated code	No	No	No		No
Mondrain	Graphical representation of class coupling	No	No	No	Doesn't look specifically for bad practices but the visual output helps identify tight coupling.	Partially - can identify tight coupling
NoVerify	Syntax checker (Unused variable, undefined variable, unreachable code, etc)	No	No	No		No
Pfff	Syntax checker (Unused variable, undefined variable, unreachable code, etc), Visualisation, shows relationships between classes/methods/files.	No	No	No		No
php-analysis	Documentation does not declare what it looks for, all links to academic papers it is based on are dead. Required "corpus" files no longer available. No sample output available.	N/A	N/A	N/A		No
PHPArch	Coupling (manually configured rules)	No	No	No	Requires manual configuration but detects code rot based on rules described at the start of the project. Each class can be tagged as a certain layer, the rules are run over the lifetime of a project to check that rules have not been broken.	No
PHP-Assumptions	Strict type checking, looks for assumptions e.g. that the user entered a number, an array key is set	No	No	No		No
PhpCodeAnalyzer	PHP features that may not be available on all systems.	No	No	No		No

Name	What is detected?	Looks for OOP bad practices?	Provides grade?	Automate fixes?	Notes	Relevant to this research?
PHPCodeFixer	Deprecated/removed features	No	No	No		No
php7mar	Code that suffers from backwards compatibility issues between php versions	No	No	No		No
phpcallgraph	Call graph visualisation	No	No	No		No
PHPCPD	Duplicated code	No	No	No		No
Phan	Backwards compatibility, used classes/methods/variables/interfaces exist, Duplicated code, unreachable code, unused variables, etc	No	No	No		No
Phinder	Manually configured patterns	No	No	No	Per-project manual configuration. Rules can be added to scan for patterns e.g. var_dump or in_array with only two arguments	No
Phortress	Security issues	No	No	No		No
php-code-static-analysis	Security issues	No	No	No		No
PHP Inspections	Simple programming errors: Syntax, performance issues, assignment instead of comparison, duplicate code, infinite loops	No	No	No		No
PHP Integrator (aka Serenata)	IDE auto completion, code generation, syntax checking	No	No	No		No
PHP Integrator (aka Serenata)	IDE auto completion, code generation, syntax checking	No	No	No		No
Phlint	Syntax errors, Deprecated code, type checking, redeclaring variables, duplicate array key	No	No	No		No
PHP Lint	Syntax errors	No	No	No		No
PHP Parallel Lint	Syntax errors	No	No	No		No
phpmnd	Magic numbers	No	No	No		No
PHP Malware Finder	Potentially malicious PHP scripts	No	No	No		No
PHP Mess Detector	Duplicated code, unused variables, threshold based code size warnings	No	No	No		No
PHP Reaper	SQL injection vulnerability	No	No	No		No
PHP SA	Syntax errors, missing documentation, unused variables/constants/methods, missing constants/methods/variables	No	No	No		No
PHP Stan	Syntax errors	No	No	No		No
PHP Unlocker	SQL queries which cause table locks	No	No	No		No
PHP testability	Issues which makes code hard to test, global variables new in constructor, static variables.	Yes	Yes	No		Yes
PHP vuln hunter	Security issues.	No	No	No		No
Progpilot	Security issues.	No	No	No		No

Name	What is detected?	Looks for OOP bad practices?	Provides grade?	Automate fixes?	Notes	Relevant to this research?
Psalm	Simple bugs such as type errors	No	No	Yes		No
psecio:parse	Security issues.	No	No	Yes		No
SonarQube	Threshold based metrics, duplicated code	No	No	Yes		No
side-channel-analyzer	Security issue: side-channel vulnerabilities	No	No	No		No
TaintPHP	Security issues: XSS and SQL injection	No	No	No		No
Taint-em-all	Security issues: XSS and SQL injection	No	No	No		No
Tuli	Static type checking	No	No	No		No
Unused-scanner	Unused package dependencies	No	No	No		No
WAP	Security: XSS, validation checker	No	No	No		No
PHP VarDump Check	Looks for var_dump debugging left in code	No	No	No		No
17eyes	Coding conventions (e.g. brace position, line length)	No	No	No		No
PHP Code Sniffer	Coding conventions (e.g. brace position, line length)	No	No	No		No
EasyCodingStandard	Coding conventions (e.g. brace position, line length)	No	No	No		No
PHPCheckstyle	Coding conventions (e.g. brace position, line length)	No	No	No		No
PHP formatter	Coding conventions (e.g. brace position, line length)	No	No	No		No
Pahout	Coding conventions (e.g. brace position, line length)	No	No	No		No
PHP Doc Check	Coding conventions (docblocks/comments)	No	No	No		No
Pahout	Coding conventions (identifies coding practices which are now old fashioned due to language improvements)	No	No	No		No
PHP Doc Check	Coding conventions (docblocks/comments)	No	No	No		No
Deptrac	Checks dependencies are not used in places they should not be	No	No	No	Requires manually configuring the rules for each project	No
PHP-cfg	Visualisation: Control flow graph	No	No	No		No
PHP Coupling Detector	Checks there is no coupling where there shouldn't be	No	No	No	Requires manually configured rules for each project	No
PHP Coupling Detector	Checks there is no coupling where there shouldn't be	No	No	No	Requires manually configured rules for each project	No

Name	What is detected?	Looks for OOP bad practices?	Provides grade?	Automate fixes?	Notes	Relevant to this research?
Rector	Refactors code: rename classes across a project, upgrade php code to remove backwards compatibility issues	No	No	Yes		No
PHP Refactoring Browser	Refactors code: Renames local variables, converts local variable to instance variable, rename classes	No	No	Yes		No
Pattern Detector for PHP	Refactors code: Renames local variables, converts local variable to instance variable, rename classes	Yes	No	No	Detects singleton patterns but does not identify them as bad, only notes they exist	No
Bliss	Unknown	Yes	No	No	Claims to make code easier to manage. No documentation, no tool available, sparse website with no links.	No
Checkmarx	Security issues	No	No	Yes		No
Codacy	Security issues, coding standards	No	No	Yes		No
Code Climate	Workflow: How quickly project issues/changes are reported/requested and implemented	No	Yes	No		No
CodeScene	Threshold based metrics coupling/cohesion	No	Yes	No		No
Symfony Insight	Threshold based metrics coupling/cohesion, package/dependency management issues, deprecated code, security, performance issues, bug detection	No	Yes	No		No
RIPS	Security issues	No	Yes	No		No
Scruinizer	Threshold based metrics coupling/cohesion	No	Yes	No		No
Sider	CI tool that uses phinder backed for PHP	No	Yes	No	Per-Project manual configuration of rules required	No
Laravel Shift	N/A	No	Yes	No	Uses static analysis to automatically upgrade old laravel code to newer versions	No

Table 1.3 demonstrates the wide variety of tools available for analysing PHP code. Despite the scope of tools available, there are none that try to accomplish what this research is doing.

## 1.5.2 Relevant tools

Of these tools looked at the most relevant are Scrutinizer (Scrutinizer-CI, n.d.) and SensioLabs Insights (SensioLabs Insight, n.d.) which grade source code on their own metrics, however they do not grade the source code by looking for known bad practices.

PHP Testability looks for things like global variables and singletons but does not produce an overall grade.

The metrics used by these tools are purely statistical and take measurements of code such as *cyclomatic complexity*, the number of control structures per method and the number of methods per class. *Scrutinizer-CI* grades a class *A* if it has a cyclomatic complexity lower than 35.

Another purely statistical metric used by Scrutinizer-CI (n.d.) and SensioLabs Insight (n.d.) is *coupling*, a tally of how many external classes each class depends on.

Cyclomatic Complexity, Lines of Code and Coupling are the other metrics used by these tools are purely statistical. They do not account for bad practices and they don't tend to differentiate between programming practices such as tight and loose coupling.

These metrics could be used to give a very rough idea of how difficult a piece of software is to maintain, however the metrics are crude. For example, *Scrutinizer-CI* gives a grade for each class in the project and an overall grade. A class in the project can go from an *A* to a *B* by introducing a single if condition because the grade is based on the *cyclomatic complexity*, the number of if statements, loops and functions. It arbitrarily gives an *A* grade to any class with under 35 complexity, and any class will get a *B* grade for having just one more if statement or loop. There is no technical reason for this number, it's just one the developers chose. Although the grade is indicative of complexity, it is only an overview.

*SensioLabsInsight* detects one Object-Oriented based bad-practice: Global variables, along with a lot of other bad practices which are not related to Object-Oriented Programming, such as consistent code styling (brace position, capitalization), deprecated functions and practices known to cause bugs or security risks. Other than global variables, it does not attempt to detect bad practices such as those described by Fowler *et al* (1999) or Hevery (2008).

Other tools fall into a family called "mess detector". These exist for many languages including Java (PHP Mess Detector, n.d.) and PHP (PHP Mess Detector, n.d.). These mostly detect duplicated code, unused variables, unreachable sections however PMD does detect some bad Object-Oriented practices such as tight coupling and digging into collaborators.

In academia, Some work made on the topic of "smell detection" to detect "code smells", which are

another name for "bad practices". The most relevant research is as follows:

**Carneiro et al (2010)** discusses programmatically detecting four specific bad practices, however these detections were for much broader bad practices such as "God classes" which are classes that "do too much" and visualising why. This is a definite bad practice that is worth detecting and falls under the broader *single responsibility principle*. However, this paper is more concerned with visualising the structure of the application to identify this than detecting and potentially fixing the problem.

**Bryton et al (2002)** discusses the subjectivity of a particular bad practice called the Long Method, where a method "does too much". Detecting and addressing this presents a lot of challenges which are discussed in detail. However, what is not explored or identified is that a long method almost inevitably contains other bad practices such as *Law of Demeter* violation (Hevery, 2008; Grimm, 2014) and if-else branching rather than polymorphism (Fowler et al, 1999; Brandsma, 2009; Ferris, 2012). Because of this, they are potentially approaching the problem from the wrong end. The *Long Method* they identify is a symptom of poor design rather than a problem in its own right. It leaves an important question unanswered: *How long is too long?*, there is unlikely to be a standard or easy to calculate answer to this, however it's possible that by detecting then fixing simpler, known, and far less subjective bad practices that the problem of *Long Methods* would resolve itself because the method would be refactored and as the bad practices are removed, the method would naturally shrink in length. The *Long Method* bad practice is more indicative of the use of other bad practices than a bad practice itself. As Bryton et al (2002) even concede themselves, there is nothing inherently wrong with long methods, the problem is that long methods tend to merge responsibilities causing poor separation of concerns which, in turn, leads to long sections of code that become difficult to understand, test and modify. It's this poor separation of concerns that is the underlying issue, not the fact that a method is made up of "too many" lines of code.

**Fontana et al (2011)** This paper discusses real world "code smell detection" tools. It's slightly out of date as it doesn't include the newer generation of tools mentioned above, however the paper shows there is a very real demand for code smell detection tools. This paper is useful, however, because it shows the range of practices detected in the tools it analysed and these are the very simplistic bad practices such as duplicated code that are detected by existing tools such as FxCop mentioned above. What is very useful to this research is a list of detection tools along with a comprehensive list of the exact bad practices they detect. Of all the tools tested, none of them detect the OOP theory based "Anti-patterns" which are caused by tight coupling of components, dependency injection related bad practices such as those recognized by Fowler et al (1999), Hevery (2008), Noback (2013) and Butler (2013).



**Liu et al (2012)** is similar to Fontana et al. (2011), this paper discusses bad practice detection but with a focus on real-time detection and fixing the bad practices during code development as soon as they're created. However, the practices being detected are the same very simplistic ones which Fontana et al (2011) describe such as *long methods*, *public properties* and *duplicate code*. Again, they make no mention of the bad practices which affect pure OOP theory such as breaking encapsulation and instantiating objects in constructors instead of using dependency injection (Hevery, 2008)

**Eden et al (2006)** identifies flexibility as a very desirable trait and approaches the idea of improving code not by reducing bad practices but by measuring the flexibility of the software by examining flex points in the software. A method of measuring the flexibility in software is provided by looking at design patterns and choices made during development. Rather than focussing on negative elements in the code, his approach looks for positive aspects such as the visitor pattern and the use of interfaces rather than complex inheritance trees when designing software. This fits Hevery (2008) who identifies and explains why "loose coupling" in software is a very desirable trait in software.

Code smell detection and static code analysis for quality metrics are beginning to gain traction in both industry with tools such as Scrutinizer-CI and in academia. However, there is currently a disconnect between the tools that have been produced and the bad practices that have been identified by experts in the field.

## 1.6 Methodology

1. Aggregate documented common bad practices, "code smells" and "anti-patterns" such as those presented by Fowler *et al* (1999). As many of the bad practices are only documented by professional programmers who have documented them after encountering the problems they discovered, often these practices are not currently formally identified in academia.
  1. Design a data structure which can be used to consistently describe each bad practice. This will need to be flexible enough to store details about bad practices which may not all be able to be described in the same way and may require several iterations or extensions as new bad practices are added
  2. Using the format from (1.1) document each bad practice which has been described by authors.
  3. Categorise each bad practice by the negative traits it introduces and store the data about what traits are used along with each bad practice
  4. The data structure designed in (a) should be flexible enough to have additional information added to each bad practice in case it needs to be extended as more bad practices are discovered which might not fit the format.
2. Demonstrate that each bad practice is "bad":
  1. With examples
    1. Give referenced code examples that demonstrate issues caused by using a practice
  2. By literature review
    1. Collect opinions about each practice
    2. Construct a method of comparing the opinions based on a weighting (e.g. did the author consider alternatives?)
    3. Perform a meta-analysis of the opinions about each practice from (1)
3. With the database of bad practices, create an extensible model that allows a programmer to look through any piece of code and easily identify bad practices by following a check list/flow chart or some other easy to follow methodology for each bad practice. The model should assign a score to each bad practice based on its severity. This severity score could be based on the number of negative traits that the bad practice introduces. This score would then allow a third party to identify how easy the software is to work with relative to other software by getting an overview of the extent of bad practices it contained.
4. Create a proof-of-concept software tool that implements the model and produces the score on given inputs using a single programming language. This will be run on sample projects sourced from open source software.
5. Once the software calculates a score for a chosen programming language the score will be

evaluated against other metrics for software analysis that measure software quality and other existing metrics such as class size which are perceived to have an impact on flexibility.

6. Many bad practices can be fixed by following a step-by-step process. In *Refactoring: Improving the Design of Existing Code*, by Fowler *et al* (1999), the authors describe methods of removing the problematic code while retaining the functionality. Building upon this this research will amend the proof-of-concept tool from (4) to optionally attempt to fix or offer suggested fixes for bad practices found.

## 1.7 Chapter Review

This chapter looked at the current methods of analysing source code along with a review of existing tools and strategies. In addition it outlines the overall research goals and describes the project methodology.

To summarise the methodology, the research will be broken up into four stages:

1. Aggregation of bad practices
2. Proving that the practices identified are genuinely considered "bad practice"
3. Creating a metric for grading source code through identification of bad practices.
4. Creating a tool to automate the metric and allow for easier testing and evaluation than would be feasible with manual code reviews.

## 2. Aggregation of Bad Practices

## 2.1 Documentation of bad practices

Based on the literature review in Chapter 1, a list of known bad practices will be created and each bad practice documented. This is a necessary step prior to progressing with the research towards creating a metric that grades software based on the frequency of these bad practices.

This list was compiled through literature review of industry experts, trends and coding practices.

As noted during the literature review, industry experts were used as there is a disconnect between industry and academia.

There is not one central list of programming practices which impede flexibility. Through experience and literature review of industry experts in chapter 1.4 the following list of bad practices was compiled.

This list of bad practices list was created from practices identified by industry experts Hevery (2008), Fowler (2002), Martin (2011) and Popov (2014).

- Global variables (Radford, 2003; Densmore, 2004; Yegge, 2004; Crockford, 2006; Zakas, 2006; Hevery, 2008; Hevery, 2008; Ronacher, 2009; Weaver, 2010; Hart, 2011; Nordmann, 2011; Butler, 2013; IBM, 2012; Kofler, 2012; Svennerberg, 2012; Ferreira, 2013; Sayfan, n.d.)
- Singleton pattern (J., 2001; Densmore, 2004; Radford, 2003; Yegge, 2004; Ronacher, 2009; Brown, 2013; Kofler, 2012; Weaver, 2010; Reddit, 2013; Badu, 2008; Knack-Nielsen, 2008; Geary, 2003; Hart, 2011; Nordmann, 2011; Sonmez, 2010; Benharosh, 2015; Deshapriya, 2011; Durand, 2013; Martin, 2014; Hevery, 2008; Hevery, 2008; Hevery, 2008)
- Static methods (Mel *et al*, 1998; Neeraj *et al*, 2005; Bracha, 2007; Hevery, 2008; Sonmez, 2010; Nordmann, 2011; Schwarz *et al*, 2011; Smith, 2012; Rybak, 2013; Eberlei, 2013; Bergmann, 2013; Mindra, 2014)
- Service Locator (Hevery, 2008; Hevery, 2009; Böhlin, 2010; Butler, 2015; Johnson *et al*, 1988; Waddicor, 2014; Seeman, 2015)
- Inheritance (Ericson, 1995; Sumpton, 2010; Hurn, 2014; van Dongen, 2014; Otander, 2015; Paul, 2013; Johansson, 2015; Kegel *et al*, 2008; Buss, 2016)
- Creating an object inside a constructor (Hevery, 2008; Hevery, 2009; Böhlin, 2010; Butler, 2015; Johnson *et al*, 1988; Waddicor, 2014)
- Setter Injection (Hevery, 2009; Butler, 2013; Schindler, 2012; Muhammad *et al*, 2013; Gierke, 2013; Arendsen, 2007; Kainulainen, 2013; Paul, 2012; Fowler, 2004)
- Annotations (Bugayenko, 2016; Ahuja, 2015; Uhrig, ; Davis, 2007; Lewis, 2013; Sosnoski, 2005; Walls, 2008; Peterson, 2008; Gilstrap, 2010; Fernández, 2011; Bell, 2013; Reigler, 2014; Torchiano, 2014)

There are certainly other practices which impede flexibility and this is not intended to be a systematic review. Inevitably new bad practices will be identified by developers in the future. These practices will be used for the remainder of this research, with any further work being built in an extensible way.

At this stage, details about the practices are being collected for documentation purposes only. In Chapter 3 analysis of the practices will be performed to determine whether developers do genuinely consider them "bad

Data about each bad practice will be collected as part of this research and will be stored in an easily accessible format that can be embedded in future tools and in a format which is extensible.

The data about the bad practices will likely be used in several places:

- The model used by developers to follow step by step detection rules
- Any software program which implements the model
- The literature review section of this thesis

To avoid duplication of effort, this will require storing the data in a human readable and machine-readable manner.

## 2.2 Aims & Objectives

1. Perform a literature review to collect known practices described as bad practice.
2. Design a data format which can be used to consistently catalogue each bad practice. This will need to be flexible enough to store details about bad practices which may not all be able to be described in the same way and may require several iterations or extensions as new bad practices are added. The chosen format should be:
  - Human readable and machine readable
  - Programming language-agnostic. Ideally with the ability to include examples of the practice in a variety of languages
  - Portable without requiring specialist software to open/parse
  - Extensible, bad practices should be able to be added easily
  - Flexible enough to allow each bad practice to include optional additional information.
3. Using the format from objective 2 to catalogue each bad practice identified.
4. Categorise each bad practice by the negative higher level traits it introduces according to references. For example, tight coupling or global state.



## 2.3 Methodology

This section outlines the methodology for documenting the bad practices and the data storage format being used.

### 2.3.1 Documenting Bad Practices

Prior to this research, if a developer wanted to learn about bad practices they would need to look through various books, journals and blog posts. There was no standardised method of disseminating this information and it is often difficult to locate due to its distributed nature.

One of the objectives of this research was to aggregate and formalise knowledge of existing bad practices and make it simpler for programmers to find this information and view it in a consistent way. To meet this objective, for each identified bad practice, the following was stored:

- The name(s) given to the practice
- The names of developers who have identified the practice as being detrimental to code flexibility
- The types of problem the bad practice introduces
- How to identify the use of the bad practice
- Code samples of the practice in use
- Code samples of one or more alternative approaches which offer greater flexibility
- A step by step guide of how to remove the bad practice and replace it with an alternative

This information needed to be stored in an accessible and extensible way so that it could be added to as new bad practices emerge. The completed database needed to be stored such that it:

- Was portable
- Could be modified without specialist software
- Could be parsed using standard tools/libraries

### 2.3.2 Storing bad practices and code samples

For this project, a file-based approach had several advantages over a relational (or document) database for both managing and creating the database of bad practices:

1. It did not require writing a software layer for reading/writing data. Any text editor could be used.
2. By selecting a standardised format (JSON (ECMA International, 2017)), the files can be loaded into any third party application without requiring a database server or language specific client implementation.

3. Text-based files can easily be managed by version control software such as Git to track revisions and tools like Github can easily allow managed collaboration.
4. Embedding source code samples in files was much simpler than storing source code in a database.

Explaining bad practices required mixing code snippets within natural language text. Because of this, a text encoding method which allowed embedding source code and had basic text formatting options (paragraphs, headings, etc) was required. RTF, PDF, Word Documents, HTML and Markdown were all considered.

Markdown was chosen because:

1. It is a very simple format which can be created/edited using text editors programmers are accustomed to, it does not require any specialist software. By contrast Word Documents and PDFs require relevant software to open and the created documents are more difficult to embed in other software.
2. It can easily be opened/parsed. Future code analysis tools could easily embed the Markdown text in their applications (which would require significant extra work using RTF or a Microsoft Word document). Markdown parsers are available for many languages.
3. Source code can easily be embedded inside Markdown documents.
4. Syntax highlighting can be applied at the rendering stage. Unlike creating a PDF, Word Document or HTML file, the creator of the document does not need to be concerned with adding syntax highlighting to source code within the document as this can be done on the fly at the rendering stage.

## Supporting Different Programming Languages

Although Markdown supports embedding source code in the document using *code fences*, this feature was not utilised directly. Embedding code in the document would have made it difficult to embed examples from different programming languages. For example, if the descriptions used PHP code examples embedded in the Markdown file along with the English description, it would not be useful documentation for a tool which detected the bad practices in Java as all the code samples would have been in PHP.

Rather than storing a copy of the entire Markdown document for each programming language, the code samples were stored in separate files and embedded on the fly.

To achieve this, the Markdown format was extended to allow embedding specific examples. Although this requires an amended Markdown format, the string chosen was designed to be

readable using a standard Markdown parser and display text instead of the code.

The following was chosen:

```
[Example][4]
```

Using standard Markdown, this would reference a link to a footnote (The same document) with the text `Example[4]`. This allows someone using a standard markdown parser to know which example to look at.

Using a modified parser, this would be expanded to an embedded code sample. For example, for displaying the page using Java samples, `[Example][4]` would be replaced with the contents of the file `examples/java/4.java`. For PHP the file `examples/php/4.php` would be used. The language would be chosen by the person viewing the file and the relevant language examples is embedded.

By separating out the sample code from the English description it made extending the database to cater for different languages much easier. Sample code can be added for different languages by supplying the code. Anyone who wished to extend the project with additional code samples could use their standard editor and all the tools that the programmer is accustomed to can be used when writing the sample code.

## File/Folder Structure

Instructions on how to identify bad practices along with code samples and guides on removing the practice will be stored as Markdown files. Each bad practice will also require a name, citations, a list of negative traits it introduces and a severity rating (See section 2.3.3).

To store this information about each practice, JSON was chosen for its flexibility and widespread support in different programming languages (Gutha, 2015).

Figure 2.1 shows the JSON file that describes a bad practice will look like this:

```
{
  "name": "Using `new` in constructor",
  "severity": "5",
  "traits": ["tight-coupling", "separation-of-concerns",
```

```

"encapsulation", "law-of-demeter", "single-responsibility-principle"],
  "references": [
    "hevery-2008Aa", "hevery-2009", "bohlin-2010",
"butler-2015", "johnson-1988", "waddicor-2014"
  ]
}

```

**Figure 2.1:** Bad practice storage format

- `name` is the informal name of the bad practice
- `severity` is the severity of the practice on a cumulative scale (see section 2.1)
- `categories` is an array storing the categories the bad practice falls into
- `references` is a list of references who have described the problems the practice causes. Each reference is described in a separate `references.json`.

## Handling References

References for each bad practice were stored in a file called `references.json`. This is a JSON file of any referenced works. An example reference is described in figure 2.2.

```

"butler-2013": { // unique identifier
  "author": ["Tom", "Butler"], //Author names as an array
  "year": "2013", //Publication year
  "title": "PHP: Annotations are an Abomination", //Publication title
  "online": { // "online", "book", "journal"
    "url":
"https://r.je/php-annotations-are-an-abomination.html",
    "accessed": "2016-07-06"
  }
}

```

**Figure 2.2:** Reference storage format

These can then be converted by the application into hyperlinks, footnotes, Harvard referencing, and numerical referencing.

The markdown format was extended to include [ref:REFERENCENAME] which can be replaced. For example [ref:wulf-1976] would be replaced with (Wulf et al, 1973) by software parsing the Markdown file.

For full documentation of the Markdown extensions implemented and references.json format see *appendix I*.

## Folder Structure

Each bad practice was given its own directory and a file structure is described in table 2.1.

**Table 2.1:** Bad practice file and folder structure

Path	Description
bad-practices/	directory to store all data about bad practices
bad-practices/[practice-name].json/	The JSON file describing the bad practice, always named "badpractice.json"
bad-practices/content/[practice-name].md	Markdown file explaining how to identify and fix the bad practice.
bad-practices/examples/[practice-name]/[language]/[file].[language]	Stores each example referenced in the markdown document. For example. java/example1.java. Can be expanded by adding new folders for additional languages

Each of the files in the content directory should form a complete document if joined together. They are separated to make it easier for tools to pick and choose sections to display under different circumstances.

## Identification rules

Describing how to detect bad practices may be possible with regular expressions or another form of pattern matching, however the exact syntax that represents a bad practice will differ between targetted languages and depending on the bad practice, detecting it may require more conditional logic than can be handled by simple pattern matching.

The rules for detecting the bad practice also need to be clear to anyone reading the

documentation with explanations and possible variants included. As such, an existing or bespoke pattern language which can be understood by the computer will unlikely not be feasible and would reduce the information for developers.

Instead, natural language explanations of how to identify and remove bad practices will be included. Any software tool which intends to detect bad practices will need to implement these rules manually.

### 2.3.3 Traits & Severity

Each bad practice introduces one or more negative traits into the code. These traits are the underlying problems that make the code less flexible than alternative approaches. A practice is labeled a bad practice because it introduces one or more of these negative traits into the code. Each negative trait will be described using its own extended Markdown file in the traits directory following a similar format to the bad practices as described in table 2.2.

**Table 2.2:** Trait file/folder structure

Path	Description
traits/	directory to store all data about bad practices
traits/[trait-name].md	Description of the negative trait in Markdown format.
traits/examples/[trait-name]/[language]/[file].[language]	Stores each example referenced in the markdown document. For example. java/example1.java. Can be expanded by adding new folders for additional languages

By using a consistent file/folder structure, the list of traits can be easily extended.

### Severity Rating

Each bad practice will be given a rating based on how many of these negative traits it introduces. There are many ways this score could be calculated to account for different levels of severity of different traits. However, even within a trait such as *broken encapsulation* there are different ways of introducing the trait. Some of which will have a higher impact than others and depending on where in a system the trait is introduced will determine how much of a negative impact that negative trait has.

For example, a global variable in a class specific to a single project will have fewer repercussions

than a global variable inside a library function that's used across dozens of projects. It would be difficult for any kind of detection tool to determine the circumstantial difference and in both instances the result is lost flexibility. It's also possible that during refactoring a project specific class is promoted to a library class.

It could also be argued that some traits are worse than others. For example, *action at a distance* can often cause more issues than *broken encapsulation*. However, this is not always the case and will be different depending on where or how the negative trait has been introduced. It's possible there are some cases where *tight coupling* introduces more issues than *action at a distance* and others where the inverse is true.

Even if it could be shown that one trait is always "worse" than another, the scoring mechanism would need to reflect that. If *tight coupling* gives a severity rating of 1 and *action at a distance* is "worse", should it be given a rating of 2? Is it really *twice* as bad or should it have a rating of 1.5? A method for quantifying *negative impact* of a given trait would need to be constructed that applied accurately in all circumstances.

The difference between traits is less meaningful to a developer than the overall negative impact to a project. If one global variable is used in two places, the negative impact is less than a static method which is used in hundreds of places. From a practical perspective, the programmer is worse affected by something they are more likely to encounter.

In addition, a programmer looking at the grade their class gets will find more use in an actionable outcome, for example, *remove this global variable* than a grade.

As such, each trait will add one to the bad practice's severity rating. If a bad practice introduces *tight coupling* and *broken encapsulation* it will be given a severity rating of *two*. A bad practice that introduces four negative traits will be given a severity rating of *4*.

This approach is considerably simpler to implement and for someone looking at the number to understand. Although it avoids the nuances and issues outlined above, it still allows an indicative comparison of the negative impact of bad practices. If, for example, the *global variables* bad practice has a severity of 4 and the *static methods* bad practice has a severity of 2 it can be said that *global variables* are *worse* because someone looking at the score can see that *global variables* introduces two more negative traits than *static methods*

Future research could explore quantifiable differences between the impact of different traits.

## 2.4 Results

The JSON file structure of each bad practice can be found in *appendix II*.

A summary of each bad practice is described below, for complete examples with longer explanations, references and sample code, see *appendix III*.

### 2.4.1 Using the new keyword in a constructor.

When an object is constructed, if the constructor instantiates another object the two objects are tightly coupled. This can be fixed by injecting the dependency rather than constructing it inside the constructor.

### 2.4.2 Annotations for configuration

When configuring the application a recent trend is the use of annotations. These are embedded in the class and as such are stored at a static level. The configuration format cannot be changed without modifying the class breaking the single responsibility principle. Configuring one part of the application from another unrelated component also causes action-at-a-distance.

### 2.4.3 Global/static variables

Global variables have been recognised to cause issues since at least 1973 (Wulf *et al*, 1973). They couple components which are otherwise unrelated and suffer from action-at-a-distance whereby one part of the application can accidentally break another part of the application due to name clashes.

### 2.4.4 Singletons

The singleton pattern tightly couples components through static method calls. It impedes flexibility by design as it enforces that only one instance may be created and introduces global state.

### 2.4.5 Inheritance

Inheritance always causes tight coupling between the parent and child classes and makes it impossible to substitute the relationship at runtime. To change the relationship the child class must be rewritten.

### 2.4.6 Service locators

Service locators couple the code using them to the service locator and implicitly to any object that the service locator can provide. Instead, the actual object required should be passed directly to the



object which requires it via dependency injection.

### **2.4.7 Setter Injection**

When using setter injection, the object suffers from temporal coupling where methods on the object must be called in a specific order to work correctly. It also exposes the dependencies of the class to any collaborators, breaking encapsulation as any collaborator can replace the dependency in the object. Instead, constructor injection should be used to ensure encapsulation.

### **2.4.8 Static methods**

Static methods introduce tight coupling between components as the method's implementation cannot be substituted. Instead, an actual instance should be passed to the method which requires it.

### **2.4.9 Negative Traits**

Each bad practice is classified as "bad" as it introduces one or more negative traits into the code. These negative traits are documented in *appendix III*.

## 2.4.7 Table of bad practices

Table 2.3 shows the complete list of bad practices, the negative traits they introduce and references that describe the practices.

**Table 2.3:** Table of bad practices

Name	Severity (number of negative traits)	Negative traits introduced	References
Service Locator	4	<ul style="list-style-type: none"> <li>• Unnecessary Coupling</li> <li>• Broken Single Responsibility Principle</li> <li>• Broken Encapsulation</li> <li>• Broken Law of Demeter</li> </ul>	<ul style="list-style-type: none"> <li>• Hevery (2008)</li> <li>• Hevery (2009)</li> <li>• Böhlin (2010)</li> <li>• Butler (2015)</li> <li>• Johnson <i>et al</i> (1988)</li> <li>• Waddicor (2014)</li> <li>• Seeman (2015)</li> </ul>
Singleton	5	<ul style="list-style-type: none"> <li>• Broken Encapsulation</li> <li>• Action at a Distance</li> <li>• Global State</li> <li>• Tight Coupling</li> <li>• Broken Single Responsibility Principle</li> </ul>	<ul style="list-style-type: none"> <li>• J. (2001)</li> <li>• Densmore (2004)</li> <li>• Radford (2003)</li> <li>• Yegge (2004)</li> <li>• Ronacher (2009)</li> <li>• Brown (2013)</li> <li>• Kofler (2012)</li> <li>• Weaver (2010)</li> <li>• Reddit (2013)</li> <li>• Badu (2008)</li> <li>• Knack-Nielsen (2008)</li> <li>• Geary (2003)</li> <li>• Hart (2011)</li> <li>• Nordmann (2011)</li> <li>• Sonmez (2010)</li> <li>• Benharosh (2015)</li> <li>• Deshapriya (2011)</li> <li>• Durand (2013)</li> <li>• Martin (2014)</li> <li>• Hevery (2008)</li> <li>• Hevery (2008)</li> <li>• Hevery (2008)</li> <li>• Sayfan (n.d.)</li> </ul>
Object not initliased after constructor finishes ( <code>`initialize`</code> and <code>`set`</code> methods)	3	<ul style="list-style-type: none"> <li>• Broken Encapsulation</li> <li>• Action at a Distance</li> <li>• Temporal Coupling</li> </ul>	<ul style="list-style-type: none"> <li>• Hevery (2009)</li> <li>• Butler (2013)</li> <li>• Schindler (2012)</li> <li>• Muhammad <i>et al</i> (2013)</li> <li>• Gierke (2013)</li> <li>• Arendsen (2007)</li> <li>• Kainulainen (2013)</li> <li>• Paul (2012)</li> <li>• Fowler (2004)</li> </ul>

Name	Severity (number of negative traits)	Negative traits introduced	References
Annotations for configuration	4	<ul style="list-style-type: none"> <li>• Broken Encapsulation</li> <li>• Broken Single Responsibility Principle</li> <li>• Action At A Distance</li> <li>• Unnecessary Coupling</li> </ul>	<ul style="list-style-type: none"> <li>• Butler (2013)</li> <li>• Bugayenko (2016)</li> <li>• Ahuja (2015)</li> <li>• Uhrig ()</li> <li>• Davis (2007)</li> <li>• Lewis (2013)</li> <li>• Sosnoski (2005)</li> <li>• Walls (2008)</li> <li>• Peterson (2008)</li> <li>• Gilstrap (2010)</li> <li>• Fernández (2011)</li> <li>• Bell (2013)</li> <li>• Reigler (2014)</li> <li>• Torchiano (2014)</li> </ul>
Use of static methods	4	<ul style="list-style-type: none"> <li>• Tight Coupling</li> <li>• Broken Encapsulation</li> <li>• Unclear Dependencies</li> <li>• Single Responsibility Principle</li> </ul>	<ul style="list-style-type: none"> <li>• Mel <i>et al</i> (1998)</li> <li>• Neeraj <i>et al</i> (2005)</li> <li>• Bracha (2007)</li> <li>• Hevery (2008)</li> <li>• Sonmez (2010)</li> <li>• Nordmann (2011)</li> <li>• Schwarz <i>et al</i> (2011)</li> <li>• Butler (2013)</li> <li>• Smith (2012)</li> <li>• Rybak (2013)</li> <li>• Eberlei (2013)</li> <li>• Bergmann (2013)</li> <li>• Mindra (2014)</li> </ul>
Using `new` in constructor	3	<ul style="list-style-type: none"> <li>• Tight Coupling</li> <li>• Broken Encapsulation</li> <li>• Broken Single Responsibility Principle</li> </ul>	<ul style="list-style-type: none"> <li>• Hevery (2008)</li> <li>• Hevery (2009)</li> <li>• Böhlin (2010)</li> <li>• Butler (2015)</li> <li>• Johnson <i>et al</i> (1988)</li> <li>• Waddicor (2014)</li> </ul>
Inheritance	3	<ul style="list-style-type: none"> <li>• Tight Coupling</li> <li>• Broken Encapsulation</li> <li>• Broken Single Responsibility Principle</li> </ul>	<ul style="list-style-type: none"> <li>• Ericson (1995)</li> <li>• Sumpton (2010)</li> <li>• Hurn (2014)</li> <li>• van Dongen (2014)</li> <li>• Otander (2015)</li> <li>• Paul (2013)</li> <li>• Johansson (2015)</li> <li>• Kegel <i>et al</i> (2008)</li> <li>• Buss (2016)</li> </ul>

Name	Severity (number of negative traits)	Negative traits introduced	References
Global/Static variables	5	<ul style="list-style-type: none"> <li>• Tight Coupling</li> <li>• Broken Encapsulation</li> <li>• Broken Single Responsibility Principle</li> <li>• Action at a Distance</li> <li>• Global State</li> </ul>	<ul style="list-style-type: none"> <li>• Radford (2003)</li> <li>• Densmore (2004)</li> <li>• Yegge (2004)</li> <li>• Crockford (2006)</li> <li>• Zakas (2006)</li> <li>• Hevery (2008)</li> <li>• Hevery (2008)</li> <li>• Ronacher (2009)</li> <li>• Weaver (2010)</li> <li>• Hart (2011)</li> <li>• Nordmann (2011)</li> <li>• Butler (2013)</li> <li>• IBM (2012)</li> <li>• Kofler (2012)</li> <li>• Svennerberg (2012)</li> <li>• Ferreira (2013)</li> <li>• Sayfan (n.d.)</li> </ul>

## 2.5 Chapter Review

In this chapter, descriptions of identified bad practices have been produced and a file format for consistently storing data about bad practices has been developed.

The files created as part of this chapter can be embedded in this research as part of the thesis as well as embedded in any future tools which need to include descriptions of bad practices. The format was designed specifically with this flexibility in mind.

A paper, summarising this chapter, entitled *Seven Deadly Sins of Software Flexibility* was presented at the China-Europe International Symposium on Software Engineering Education conference in 2017 (Butler, 2017) and is available in *appendix V*. This venue was chosen as one application for this research is educating future software engineers.

### **3. Demonstrating the practices really are "bad".**

## 3.1 Introduction

Building upon the last chapter which documented known bad practice, this chapter sets out to demonstrate that these bad practices genuinely are considered "bad practice" by developers.

To facilitate this, a scoring system has been created to allow developers of most abilities to calculate analytic rigor of an article discussing a programming practice. Multiple articles can be graded and different articles compared on their analytic rigour.

The score can then be used to produce a meta-analysis of articles discussing each bad practice to look for trends. A typical trend might be: developers who consider alternative approaches more likely to prefer the practice over alternatives.

### 3.1.1 Background

*In 1950, a vote at the meeting of the British Association for the Advancement of Science showed that about half those present now embraced the idea of continental drift. [...] Interestingly, oil company geologists had known for years that if you wanted to find oil you had to allow for precisely the sort of surface movements that were implied by plate tectonics. But oil geologists didn't write academic papers; they just found oil.*

---

Bryson (2010)

The Singleton has been regarded as bad practice in industry since at least 2003 (Radford, 2003) with developers denouncing it ever since (Hevery, 2008; Sayfan, n.d.; Densmore, 2004; Radford, 2003; Yegge, 2004; Ronacher, 2009; Brown, 2013; Kofler, 2012; Weaver, 2010; Knack-Nielsen, 2008; Badu, 2008; Hart, 2011) yet where it is mentioned in academia it is only discussed as having been utilised while developing software rather than discussing whether it should or should not have been used (Alipour *et al*, 2016; Liu *et al*, 2011).

A 2018 systematic review of academic literature found 56 different "anti-patterns", "code smells" and "bad practices" which have been identified in academic literature (Sabir *et al*, 2018). Despite the widespread derision of the singleton pattern among industry developers it is not currently recognised as a bad practice in academia. As a systematic review of the academic literature did not find a single source calling the singleton pattern a bad practice, a review of academic works is not possible due to the limited data available.

In industry the singleton pattern is the most derided programming practice after global variables (Knack-Nielsen, 2008). Given the lack of discussion about the singleton pattern in academia it is unsurprising that patterns which are lesser discussed in industry do not feature in academic literature either.

It is hypothesised that the gap between academic works and industry is because bad practices are identified by people who spend 8 hours a day working on large projects where they are likely to encounter problems that academics focusing mostly on theory will not. Industry experts tend to work on large software projects which require constant maintenance and enhancement for years or even decades. They are able to determine which practices prevent them performing maintenance efficiently.

The systematic review by Sabir *et al* (2018) lists many bad practices which were first identified in industry by Fowler *et al* (1999) and were referenced by academic works shortly afterwards. While industry has moved on, academic sources have built on previous academic sources but have not gone back and incorporated new developments from industry.

Like the oil geologists, industry experts don't tend to write academic papers, many industry experts post articles on websites run by themselves or the company they work for.

### 3.1.2 Rationale

A problem with research in this area is that labeling a practice "bad" can be considered subjective. Fowler *et al* (1999), in the same book where the term "code smell" was introduced, notes that:

---

*We based our collection of refactorings on our own programming experiences.*

---

Despite listing over a dozen "code smells", these are based entirely on the author's industry experience.

Is the idea of a bad practice purely subjective or can a bad practice be defined by consensus? For example, "9 out of 10 dentists agree". If two or more developers independently reach the same conclusions about a practice that may infer that expertise is a factor and labeling a practice "bad" is not entirely subjective.

If the idea of a *bad practice* were purely subjective, different developers would disagree on whether any given practice was indeed *bad*. A purely subjective practice would result in being unable to predict a developer's attitude to a given practice. By analysing the opinions of different developers it may be possible to identify trends.

A single article is a single developer's opinion. A group of articles by different authors that independently come to the same conclusions about the same practice is a consensus.

If analysis produced a *predictive model* it would demonstrate that a consensus was reached by experts.

Do developers who consider alternative solutions reach a different conclusion than those who do not? Two or more developers may offer different contrary opinions about a practice but do those who have considered the practice in more detail reach the same conclusion?



## 3.2 Aims and Objectives

This stage of the research has its own set of aims and objectives:

### 3.2.1 Aims

1. Bring missing industry developments in this area into academia.
2. Demonstrate that industry developers attitudes towards the chosen practices are dependent on their level of analysis.
3. Create a reproducible method of gathering industry developer opinions about any given programming practice.

### 3.2.2 Objectives

1. Create a scoring system which can be used to grade the analytic rigour of an article/book/paper discussing a particular programming practice.
2. Compare the analytic rigour of different articles for the purposes of meta-analysis.
3. Compare the overall quality of discussions about a specific programming practice.
4. With a scoring system in place, perform proof-of-concept meta-analyses on practices which are well known to be described as "good" and "bad" to demonstrate that the meta-analysis methodology is fit for purpose.
5. Perform meta-analyses of remaining bad practices.

## 3.3 Methodology

The following section outlines the methodology used to demonstrate that the bad practices identified in chapter 2 are generally considered "bad practice" among developers.

### 3.3.1 Metric for comparing analytical rigour in programming articles

Differing methodological rigor in sources is a problem which exists when doing any kind of meta-analysis. When performing meta-analysis of clinical trials the Cochrane Collaboration consider methodological rigour an important part of their meta-analysis (Cochrane, n.d.) .

Rather than simply counting the number of trials which show a positive outcome and counting the number of trials which show a negative outcome, the trials are weighted on methodological rigour. For example, in a meta-analysis of a drug they may find that 3 trials show that it is an effective treatment and 8 which say that it is not. Instead of simply counting the numbers on each side, methodological rigor of each study is used as a factor when building conclusions on the overall efficacy of the treatment.

In a meta-analysis of the efficacy of homeopathic treatments it was found that trials of homeopathy with a poor methodology are much more likely to show a positive outcome whereas trials with a robust methodology are more likely to conclude that homeopathy is no better than placebo (Mathie *et al*, 2015) .

This is because methodological rigour can affect the outcome. For example, by putting the most healthy patients in the experimental group and putting the least healthy patients in the control group it's likely that the experimental group will see significant improvement over the control group regardless of whether the drug being tested has any effect (Goldacre, 2010).

For programming articles, analytic rigour can be plotted against whether the article recommends using or avoiding the practice to create a meta-analysis in a similar manner.

It should be possible to draw conclusions such as an article's analytic rigour increases, it is more likely to recommend using the practice in question.

The created metric was based on the Jadad Scale (Jadad *et al*, 1996) used for analysis of clinical trials in medicine. The Jadad Scale is a 5 point scale using a 3 question questionnaire which can be used to quickly assess the methodological rigour used in a clinical trial. The questions asked are: *Was the study described as randomized?*, *Was the study described as double blind?* and *Was there a description of withdrawals and dropouts?*. These are then used to calculate a score from zero (very poor) to five (rigorous). By citation count the Jadad Scale is the most widely used method of comparing clinical trials in the world (Olivo *et al*, 2008).

As the Jadad Scale is not applicable for anything other than the clinical trials, for the purpose of this research a new metric was created based on the principles of the Jadad scale to be used in determining the analytic rigour of any given article about a programming practice. A seven point scale was created, based upon the principles of the Jadad scale. A point awarded if the article does each of the following:

1. Describes how to use the practice.
2. Provides a code example of using the practice.
3. Discusses potential negative/positive implications of using the practice.
4. Describes alternative approaches to the same problem.
5. Provides like for like code samples comparing the practice to alternative approaches.
6. Discusses of pros/cons of the compared approaches.
7. Offers a conclusion on when/where/if the practice is suitable.

These points were chosen as they inquire whether an article's author has considered alternatives. Points 2, 3, 5 and 6 could be removed for a less granular scale, however these have been included to enable differentiation between articles which cover the topic in different depths.

In addition, these points are binary choices. Does the article include these things or not? Any two reviewers should come up with the same scores for any given article.

It is possible, though unlikely, that any article which includes point 4 also includes point 5. If this is the case, any results/conclusions gathered from the score will be the same as if point 5 wasn't included, however the added granularity may offer more detailed results. If this granularity is removed, overall trends will remain the same as long as one of the questions inquires about whether the article considers alternative approaches or not.

Using this metric, a manual page that describes a practice and provides a sample of how to use it would score two whereas an article that discussed the pros/cons of different approaches and made a recommendation would score seven.

This scale is tested in section 3.3.5 using two sets of 100 articles that produced a known outcome. It is shown that articles which scored higher on this scale were more likely to suggest using dependency injection and also more likely to suggest avoiding the singleton.

### **3.3.2. Meta-analysis**

Clinical trials can be separated by their Jadad score but this alone shows nothing about the efficacy of the treatment being analysed. To produce a conclusion the Jadad score of a trial is plotted against whether it shows the treatment to be effective or not.

By comparing the results of multiple trials, a set of trials studying the same treatment can be analysed and observations drawn such as *trials with lower Jadad scores are more likely to produce a positive result*, indicating that the stronger the methodological rigor the less likely the treatment is to be shown to be effective.

Programming articles do not produce a result, but they can offer a recommendation to use or avoid the practice being discussed. A manual page won't make a recommendation but an opinion piece will discuss if/when the practice being described should be used.

A five point scale was used to model the recommendation made by an article:

1. Always favour this practice over alternatives
2. Favour this practice over alternatives unless specific circumstances apply
3. Neutral - No recommendation (e.g. a manual page) or no conclusion drawn
4. Only use this practice in specific circumstances
5. Always favour alternative approaches

A five point scale was chosen over a three point scale as there may be cases where an article is concluded with a discussion of trade-offs. For example where an approach may be faster but less flexible an author may conclude their article with something like "use this practice unless performance is a priority".

This research focus on flexibility. If a conclusion is drawn that you should use a practice when flexibility is preferred over performance (or any other consideration) then the article would be awarded a score of 2 and considered as "Favour this practice unless performance is a paramount concern".

For research with a different goal, the focus of the analysis could be be changed to performance, security or any other metric and results gathered in the same manner.

Although this scale is potentially subjective, "favour this practice" and "avoid this practice" is a dichotomy. An article will clearly fall into one of the recommendation groups or make no recommendation at all. There is very little chance that an article's conclusion could imply both "favour this practice" and "avoid this practice" as they are contradictory points.

A three point scale: *Favour, No recommendation, Avoid* will produce the same trends on a less granular scale. Regardless of potential subjectivity of "Avoid at all costs" and "Only use in specific circumstances", trends such as "As acadmic rigour increases, authors are more likely to suggest avoiding the practice" can still be observed.

For granularity, although adding subjectivity. A 5 point scale was used instead of a three point

scale. Lower resolution results could be produced using existing data by merging any 1 and 2 point results and 3 and 4 point results. Any overall trend would remain the same.

To minimise subjectivity for points 2 or 4, the specific circumstances have to be described rather than alluded to. This is to avoid ambiguity. For example Buss (2016) writes:

*When designing a system, it's important to pick the right design principle for your model. In many circumstances, it makes sense to prefer composition over inheritance.*

---

This article only alludes to when using inheritance is preferable and provides only examples where composition is preferred. In this case the article is given a 5 despite the conclusion saying "many circumstances" rather than "all circumstances".

To score a 4 the article must explicitly define what the circumstances are. A 4 was awarded to Ericson (1995) as the author clearly states a situation where inheritance should be used over composition:

*If you aren't sure if a class should inherit from another class ask yourself if you can substitute the child class type for the parent class type. For example, if you have a Book class and it has a subclass of ComicBook does that make sense? Is a comic book a kind of book? Yes, a comic book is a kind of book so inheritance makes sense. If it doesn't make sense use association or the has-a relationship instead.*

---

### 3.3.3. Collecting data

Data was collected using Google from articles written by companies, developers and technology journalists. As a Google search for *singleton pattern* yields over half a million hits, a complete systematic review was not feasible. Instead, the first 100 relevant results from a Google search for the programming practice being analysed will be used as the sample.

A *relevant result* is defined as an article which is written by a single author or organisation describing or discussing the singleton pattern. Discussion forums, posts on social media and question & answer sites will not be included as these pages will include multiple opinions. Comments sections on articles will be omitted for the same reason. Any article which has a *Jadad* style score of zero will also be deemed irrelevant.

To avoid potential selection bias and to get an overview of what a developer searching for the topic would discover, no other inclusion/exclusion criteria were used. Future research could look into whether attitudes change over time by selecting articles between specific dates or look at websites only about particular programming languages.

Google was used to act as a randomization tool. A search returns any articles discussing the practice regardless of whether they are for or against its use.

Each article was then given a *Jadad* style score from 0-7 and a score from 1-5 for its recommendation.

A list of all URLs accessed and scores given is available in *Appendix VI*.

### **3.3.4 Additional considerations**

There are several practical issues with collecting data in this manner:

1. To minimise the effect of Google giving user-specific results based on previous searches, results were collected while logged out and using the browser's private browsing mode and closing the browser between each search term.
2. Search results will not be truly random due to the way Google's algorithm works and results will be sorted by *relevance* and the way Google sorts the results may have implicit bias: The most popular links and most cited links will appear first. Although not truly random, this gives a better overview of the zeitgeist than a genuinely randomised sample by putting the most read/cited articles ahead of less read/cited pages. Articles which are widely shared and linked to will be more likely to appear in the first 100 results.
3. A practice may have more than one common name. When this is the case, each name will be searched for and 100 results collected in total. If a practice is known by 4 different names, the first 25 relevant results for each practice were used. If a result lists both names it will only be counted once.
4. Other search engines may yield different results. Google was chosen because of its dominance and likelihood to have indexed more results. Using a search engine such as Qwant (Qwant, n.d.) which does not offer personalised results would make the results easier to replicate but may not offer as comprehensive results. Regardless of which search engine is used, results will change over time.
5. Ideally this methodology would be performed on academic literature rather than a wide-net Google search. However, as documented in chapter 1, these bad practices are currently not widely discussed in academic literature.

Regardless of these factors, results should be indicative of developers' attitudes towards the programming practice being analysed.

### **3.3.5 Test methodology**

To verify that the suggested meta-analysis methodology produces meaningful results, a meta-analysis was performed on two practices where the result can be predicted with a high degree of certainty. If the methodology works as intended, the following hypotheses should be proven true.

#### **Singleton pattern**

The singleton pattern is well known as being considered bad practice among developers (Knack-Nielsen, 2008) and will act as a good benchmark for testing the meta-analysis methodology.

#### **Hypothesis**

Before the results were collected it was expected that articles which had a higher *Jadad style score* (higher academic rigour) would be more likely to suggest avoiding the practice.

#### **Dependency Injection**

Dependency Injection is antithesis to the Singleton Pattern and is much more flexible. Although there are some practical considerations when using Dependency Injection and there is widespread discussion about the best way to implement it, it's widely considered the best approach for flexibility (Albert, 2013).

#### **Hypothesis**

Dependency Injection a well established method of increasing flexibility in code (Fowler, 2004). Because of this, it is expected that there will be few to no negative recommendations and as the *Jadad style score* increases articles should be more likely to suggest favouring dependency injection over alternative approaches.

## 3.4 Results

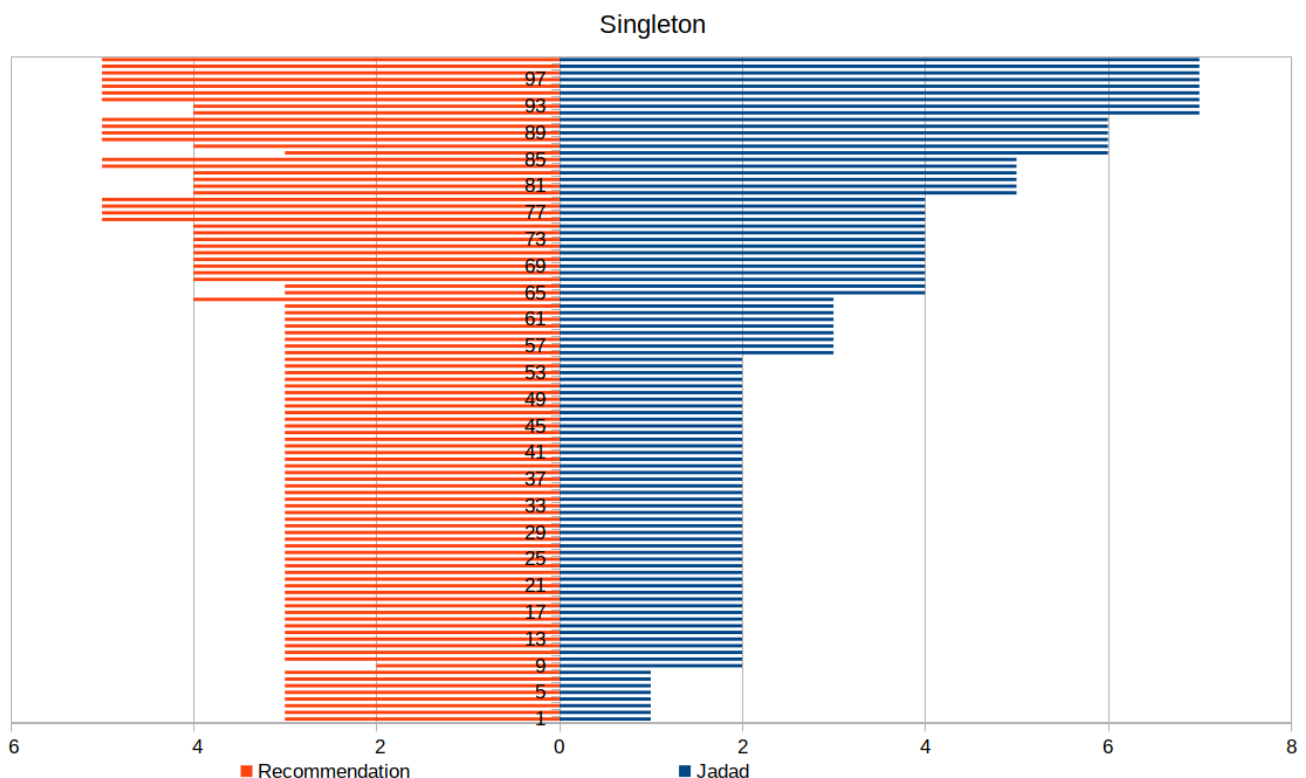
After gathering data about programming practices for the two test meta-analyses, the methodology was followed to check that it produced expected results which match the hypotheses outlined in section 3.3.5.

Raw data used for these meta-analyses is available in *appendix VI*.

### 3.4.1 Singleton

Figure 3.1 shows the results for the meta-analyses of the Singleton. Each line represents an article and the left (orange) bar for each article is the recommendation going from 5: Avoid this practice at all costs (Far left) to 1: Favour this practice over alternatives.

The right (blue) bar for each article is the Jadad style score measuring analytic rigour. A score of seven means the article describes the practice, provides code examples, discusses alternative approaches, provides like-for-like code samples, discusses the pros/cons of each approach and makes a recommendation of which approach should be used.



**Figure 3.1:** Singleton results

Article 1 has a recommendation score of 3 and a Jadad style score of 1. It does not go into detail and its recommendation is neutral; it doesn't suggest either avoiding or favouring use of the



Singleton Pattern.

Article 99 strongly recommends against using the Singleton Pattern and has an Jadad style score of 7, it compares the singleton against alternatives in detail and concludes by strongly recommending against its use (recommendation score of 5).

### **Key findings - Singleton Pattern**

- As hypothesised, articles with a high analytic rigour are considerably more like to suggest avoiding the singleton pattern.
- If a simple tally was used, the singleton pattern would appear to have a mostly neutral recommendation score. 65% of articles do not recommend for or against its use.
- The mode recommendation is neutral.
- The mean recommendation score is 3.5. From this alone it could be inferred that the singleton pattern is generally considered to be neutral, slightly discouraged but not widely avoided.
- When the Jadad style score is taken into account, every article which makes a recommendation recommends against using the singleton pattern (recommendation score of 4 or 5).
- Only 22% of articles about the singleton pattern even mention alternative approaches that can be used to solve the same problem.
- Of those that recommend against using the pattern, over half say it should be avoided at all cost.
- 55 of the 65 articles which make a neutral recommendation are manual type pages (Jadad style score of 2) which show how to use the pattern but do not weigh in on when, where or if it should be used and do not compare the pattern to alternatives.
- No articles which make a recommendation recommend using the singleton pattern instead of alternative approaches.

### 3.4.2 Dependency Injection

Figure 3.2 shows the results for the meta-analysis of Dependency Injection.

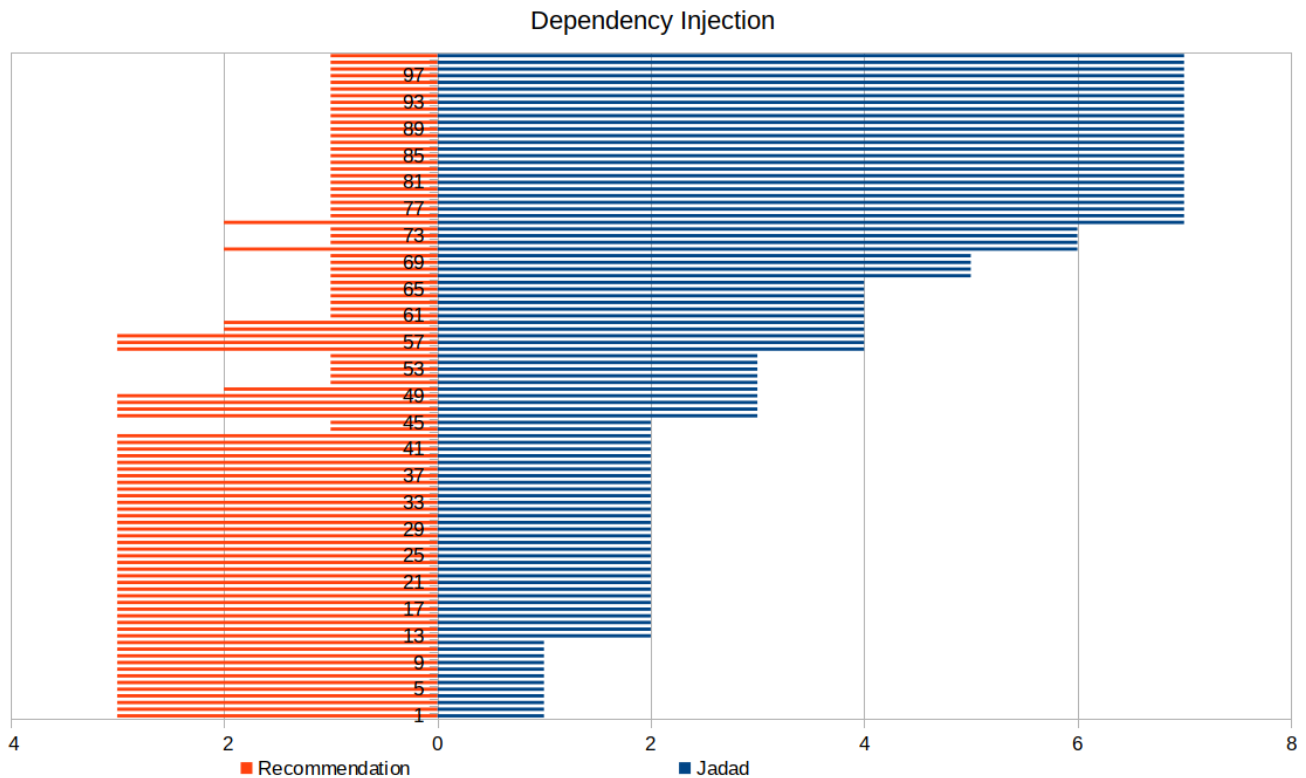


Figure 3.2: Dependency Injection results

#### Key findings - Dependency Injection

- As hypothesised, Dependency Injection is seen as overwhelmingly positive with zero articles favouring alternative approaches.
- The mean recommendation score is 1.94 which shows that even using a simple tally, the overall recommendation is that Dependency Injection is a favourable pattern among developers.
- 50% of articles suggest using dependency injection instead of alternatives.
- Every article with an analytic rigour score of 4 or higher recommends using this practice instead of alternative.

- 47 of the 50 articles with a neutral recommendation are manual style pages which show how the pattern is used but do not discuss when, where or if it should be used.
- Discounting the manual pages, only two of the remaining 53 articles make a neutral recommendation and both of those have a *Jadad* style score 3.
- As the Jadad style score increases, the probability that an article will recommend using Dependency Injection over alternatives increases.
- Only 5 of the 55 articles in favour of dependency injection (Recommendation score of  $< 3$ ) suggest there are some specific circumstances where alternatives should be used instead.

## 3.5 Conclusion

By testing the methodology with practices that the outcome can be predicted for it was possible to validate this meta-analysis methodology.

The methodology produced the expected result. It was shown that if an author considered alternative approaches they were more likely to recommend against using the Singleton Pattern. The inverse was also true for Dependency Injection.

As these were the expected results, the methodology suggested can be shown to work as intended and provide an overview of the attitudes of developers about any given practice.

This meta-analysis methodology gives more insight into the overall opinion of programming practices than a simple tally of for/against/neutral by also accounting for analytic rigour.

### 3.5.1 Key findings

1. Although a small sample size of two practices were used to test that the scores work, in both cases roughly half of articles analysed do not make a recommendation on when/where the practice should be used. For the singleton pattern only 45% of analysed articles discussed whether the pattern should be used or avoided. The remaining practices will be analysed in the following sections.
2. Any developer looking for information on a practice will find more information about *how* to use a practice than *when* or *where* the practice is applicable.
3. A sample size of 100 articles is enough to demonstrate a trend, it is unlikely that increasing the sample size will affect the results, though this hypothesis could be tested in future research.

### 3.5.2 Problems Encountered

Data collection using Google became increasingly difficult after around 80 relevant results. The number of irrelevant articles appearing in search results begin to heavily outweigh the relevant articles and there was a significant issue with duplicated content which was overcome by keeping track of author names and article titles.

Since Dependency Injection and the Singleton pattern are both widely known and discussed programming practices, finding 100 unique relevant results for lesser known practices may be

difficult.

### 3.5.3 Evaluation

The scoring system has been tested by using the methodology on bad practices where results can be predicted and the scoring system gave expected results.

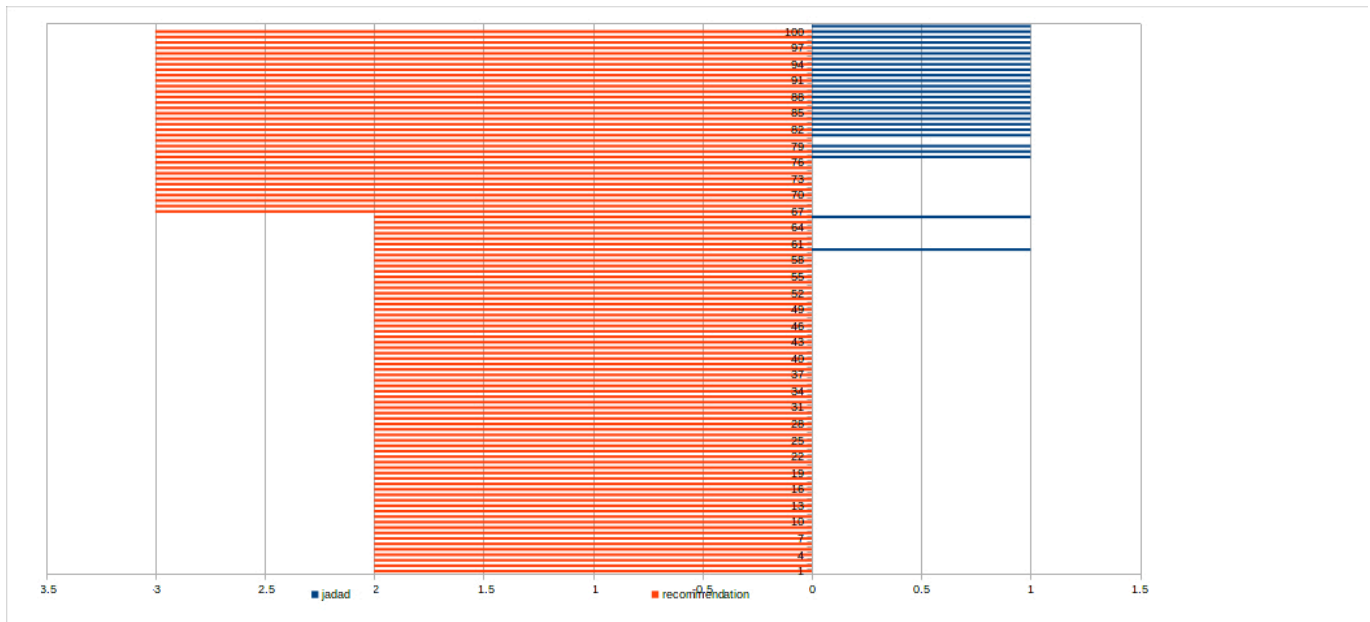
While the results look valid, one weak point is that all data was collected by the same person. Although the scoring systems have been designed to be as objective as possible by using binary points and dichotomous scales:

The seven point Jadad-style score uses binary options, for example "does it include code examples?" and "does it discuss alternative approaches?". These are unlikely to be contentious points, however the lack of additional participants leaves this question open.

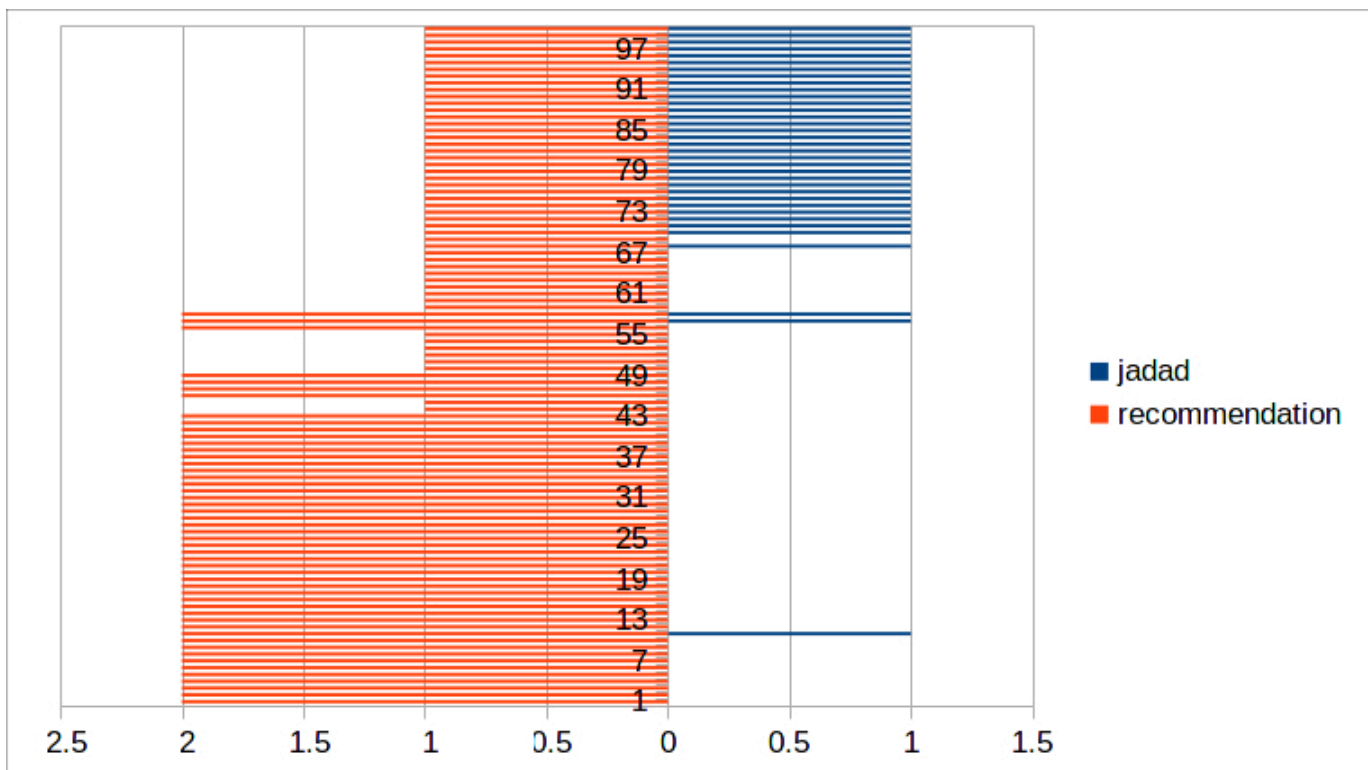
Despite this, it is possible that a different person would grade articles differently and yield different results. However, small differences in analysis of the articles would be unlikely to affect overall trends.

To test this hypothesis and to identify if possible ambiguities would affect the overall outcomes, the charts were generated using the minimal data possible:

1. The seven point jadad-style score is now a single binary option: Does the article compare the practice to alternatives?
2. The 5 point was refined to "avoid the practice", "no recommendation" and "use this practice" (points 1 and 2 were combined as were points 4 and 5).



**Figure 3.3:** Singleton results



**Figure 3.4:** Dependency Injection Results

In figures 3.3 and 3.4, the overall trends are the same even though the dataset has been minimised. These graphs show only whether the article discusses alternative approaches and whether the author recommends using or avoiding the practice. The overall

Although additional participants may grade articles slightly differently, it is very unlikely to affect

the overall trends.

### **3.5.4 Future Research**

To further validate the model a selection of participants could be used to grade a set of articles using the two scales. This would ensure that the grades given were accurate and that the point scales were as objective as possible.

This research could be continued by running the same meta-analysis on different search engines and comparing the results or looking into trends over time using article dates. For example, it may be observed that a practice is seen favourably in articles published in 1990s-2000s and then less favourably as time progresses.

This methodology could be abstracted to and used for a meta-analysis of any widely discussed topic by defining the scales for academic rigour and recommendation.

## **3.6 Results for remaining bad practices**

The previous section demonstrated that the methodology works and produces results that can be use for meta-analysis. The approach was then used to perform meta-analyses for the rest of the bad practices. Each section below contains the results, key findings and conclusions for the meta-analysis of each bad practice. Raw data for all the meta-analyses are available in *appendix VI*.

### 3.6.1 Annotations

A meta-analysis was performed for annotations using the search term "annotation configuration". It quickly became apparent that this term was mostly yielding results demonstrating how annotations were used for configuration in a specific library rather than comparing the use of annotations to alternative approaches.

This search was stopped after 20 results as most results were not relevant to the research:

- 18 of the 20 results relate to Java's popular Spring where annotations are very commonly used.
- All results were examples of configuring libraries using annotations, rather than comparing annotations to alternative approaches.

To find relevant results, which discuss the pros/cons of using annotations or alternative four new search terms were used

- annotation configuration "best practice"
- annotation configuration "good practice"
- annotation configuration "bad practice"
- annotation configuration anti pattern

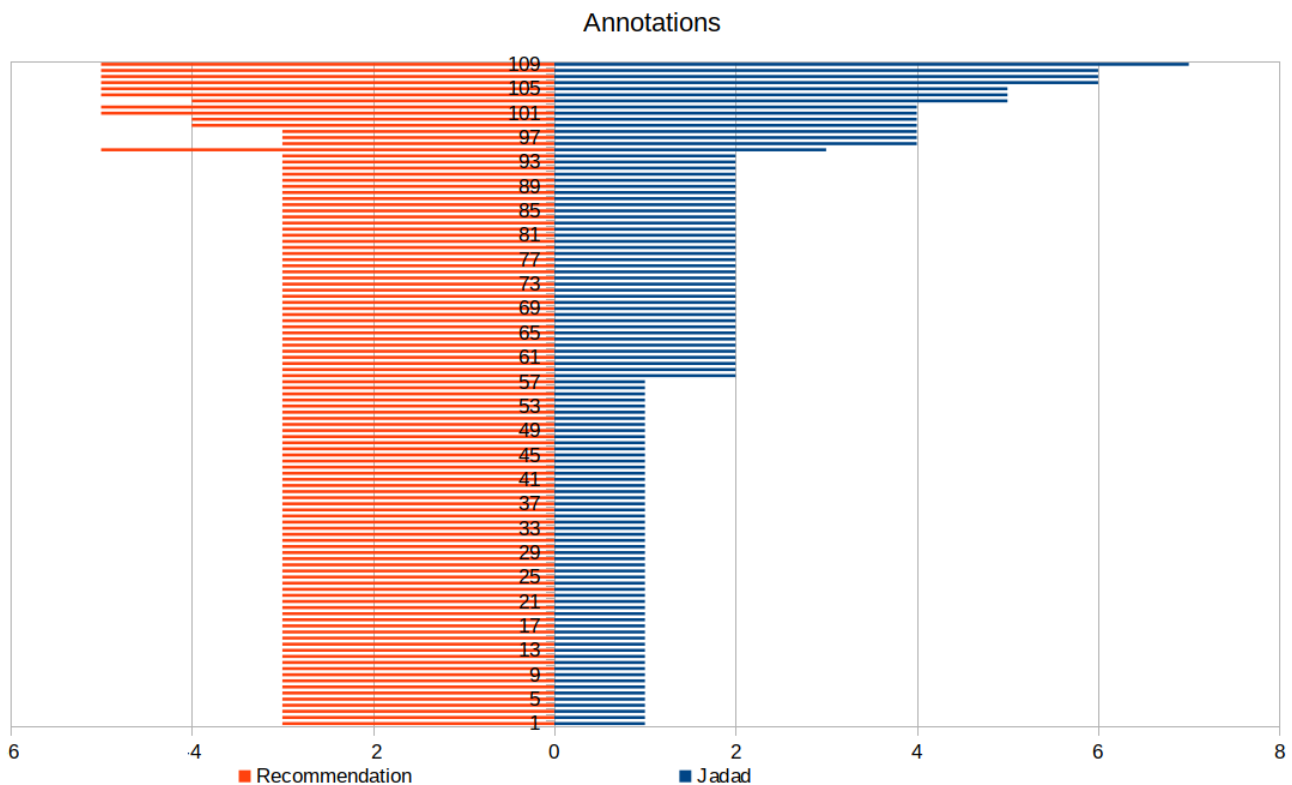
Searches were stopped after either 50 relevant results or page 10 of search results and results that appeared in more than one set of search results were only included once. In total 110 results were gathered across the four search terms.

Although these search terms all have explicit bias and will bring up results specifically discussing annotations against alternatives, searching explicitly for "good practice" and "best practice" should be biased in favour of results where authors talk favorably about annotations, however it was found that the inverse was true. Search results containing the terms "good practice" and "best practice" overwhelmingly argued against using annotations for configuration.

### Results

Figure 3.4 shows the results for the meta-analysis of Annotations.





**Figure 3.4:** Meta-analysis results: Annotations

### Key findings

- There is a clear correlation between the recommendation score and Jadad-style score. Every article with a Jadad-style score of 5 or higher recommends avoiding the practice.
- The mode recommendation score is 3 (neutral/no recommendation). A simple tally would imply that most people do not recommend either using or avoiding the practice.
- The mean recommendation score is 3.22. A metric that did not account for analytic rigour would show developer's attitude to be slightly unfavourable but nearly neutral.
- Although there are many articles discussing annotations, only 8.1% of articles discuss potential alternative approaches and only 12.7% mention negative aspects of using the practice. However, this is only slightly lower than the very well known bad practice Global Variables which has 8% and 21% respectively.
- Each of the nine articles that compares annotations to alternatives suggests using

alternatives instead of annotations.

- In total twelve articles recommend using alternatives instead of annotations and zero recommend using annotations over alternatives.
- The Jadad-style score correlates with the recommendation. The higher the Jadad-style score, the less likely the author is to recommend using annotations.

## Conclusion

Annotations are used by a large number of libraries and frameworks, however compared to the other bad practices analysed there is very little discussion surrounding when or if annotations should be used in place of alternative approaches.

There is a clear trend that when annotations are compared to alternative approaches, alternative approaches are preferred.

Further research could be carried out to examine why there is so little discussion surrounding the merit of annotations, however it is hypothesised that reasons for the lack of discussion include:

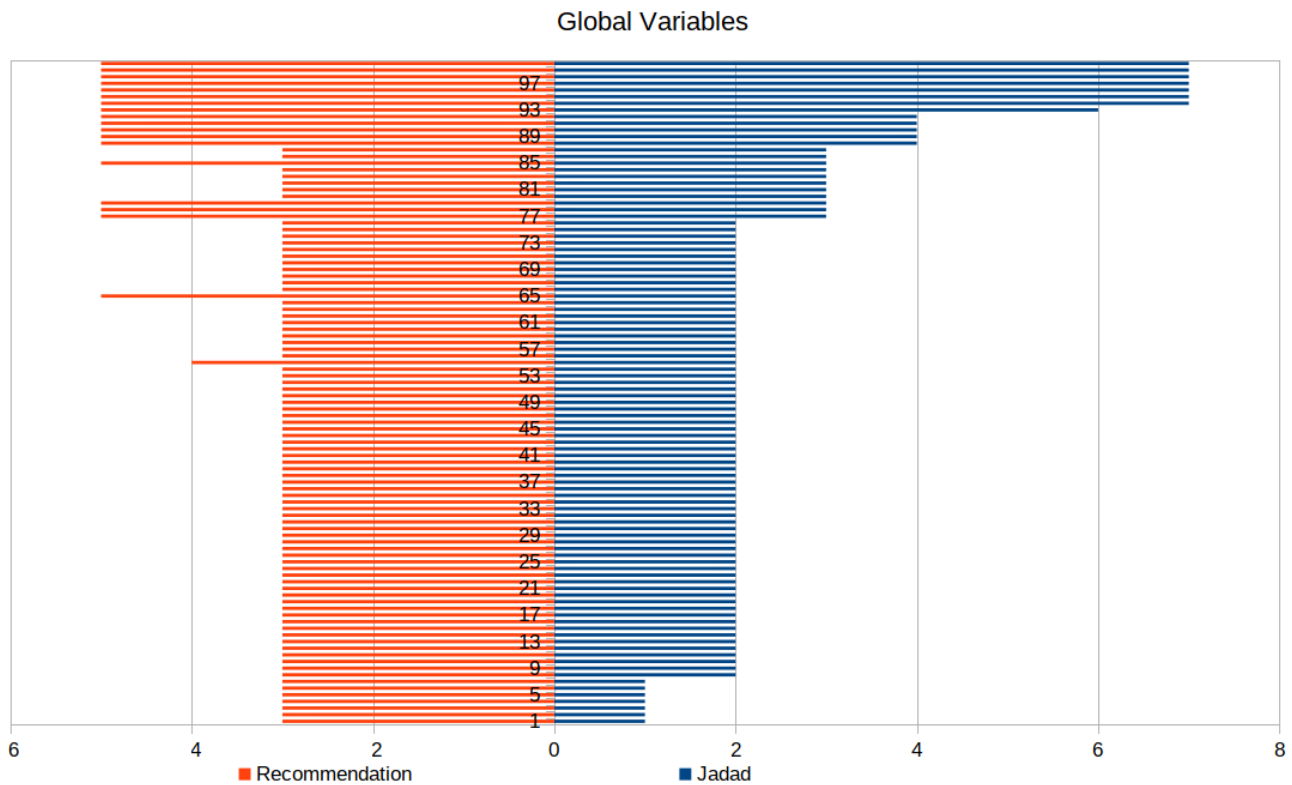
1. Annotations are not widespread and are only used in some, albeit popular, libraries such as Java's Swing.
2. Compared with other practices, annotations are modern. They have only been available since 2004 in Java (Oracle, 2014) and much more recently in other languages. Even after their introduction there would have been a delay between the language feature being introduced and use in libraries. One of the early adopters of annotations, Spring Framework, did not start using them until late 2007 (Spring, 2018) and there would have been a further delay between the library release and developers using the new version. Then developers would not have run into maintenance issues until they needed to restructure their applications after implementing them. The series of delays may be why problems are only starting to be discussed relatively recently.

### 3.6.2 Global Variables

A meta-analysis was performed for annotations using the search term "global variables".

#### Results

Figure 3.5 shows the results for the meta-analysis of Global Variables.



**Figure 3.5:** Meta-analysis results: Global Variables

#### Key Findings

- There is a clear correlation between the recommendation score and Jadad-style score. Every article with a Jadad-style score of 5 or higher recommends avoiding the practice.
- The mode recommendation score is 3 (neutral/no recommendation). A simple tally would imply that most people do not recommend either using or avoiding the practice.
- The mean recommendation score is 3.22. A metric that did not account for analytic rigour would show developer's attitude to be slightly unfavourable but mostly neutral.

- Although global variables have been described as "bad practice" since at least 1973 (Wulf *et al*, 1973) and are one of the first bad practices junior developers are taught about (Judis, 2017), only 21% of articles discussing global variables mention the negative implications of their use and only 17% recommend against using them.
- As expected, zero articles recommend using global variables over alternative approaches.
- As the Jadad-style score increases, an article is more likely to recommend against using global variables.

## **Conclusion**

Although any article making a recommendation recommends against their use, having just 17% of articles about global variables talking about them as a bad practice is surprising given how derided they are (Meyer, 1988; Hevery, 2008) and that global variables being bad practice is one of the first things often taught to junior developers (Judis, 2017). If global variables are only described negatively 17% of the time, this makes less common and more complex bad practices significantly less likely to be discussed in terms of negative or positive impact.

### 3.6.3 Inheritance

The results for the meta-analysis for Inheritance are displayed in the following chapter. However, due to the common name of the practice outside of programming terminology, this meta-analysis had to be carried out differently to others. The amended methodology is outlined below.

#### Methodology

Due to the term "inheritance" not being exclusive to programming the search term "inheritance class" was used to bring up only programming related results. Similar search terms like "inheritance programming" or "inheritance oop" would yield similar results but the page may not mention "oop" or "programming". However, any discussion about inheritance will need to mention classes.

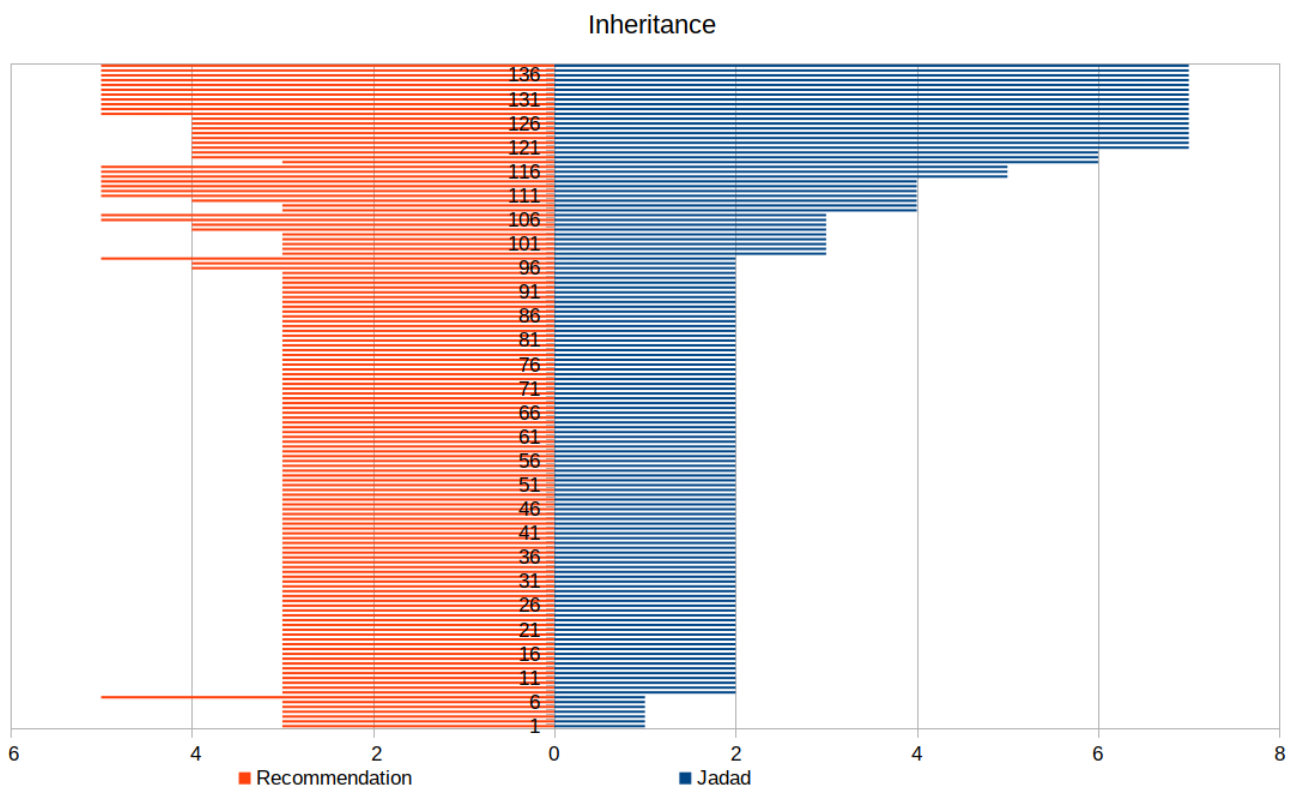
Using this search term, only 7% of results made a recommendation on whether to use inheritance or not. All 7% argued in favour of alternatives.

As 7 articles is a very small sample size, additional search terms were used to find articles which specifically compare inheritance to alternatives:

- "vs inheritance" class
- "inheritance vs" class

These terms should not introduce any bias but return only relevant results. As previously, the search keyword class was appended to ensure only discussions about programming are returned. Relevant results from the first ten pages of each of these searches were added to the data set if the URL was not already present which is why there are more than 100 articles being in the results.

#### Results



**Figure 3.6:** Meta-analysis results: Inheritance

Figure 3.6 shows the results for the meta-analysis of Inheritance.

### Key Findings

- There is a clear correlation between the recommendation score and Jadad-style score. With the exception of one neutral recommendation, every article with a Jadad-style score of 5 or higher recommends avoiding the practice.
- The mode recommendation score is 3 (neutral/no recommendation). A simple tally would imply that most people do not recommend either using or avoiding the practice.
- The mean recommendation score is 3.4. A metric that did not account for analytic rigour would show developer's attitude to be slightly unfavourable but mostly neutral.
- There are zero articles which suggest favouring inheritance over alternative approaches.
- The Jadad-style score correlates with the recommendation. The higher the Jadad-style score, the less likely the author is to recommend using inheritance.

## **Conclusion**

There is an overwhelming amount of information about how to use inheritance in various languages. However, discussions around when it should be used over alternative approaches are comparatively rare.

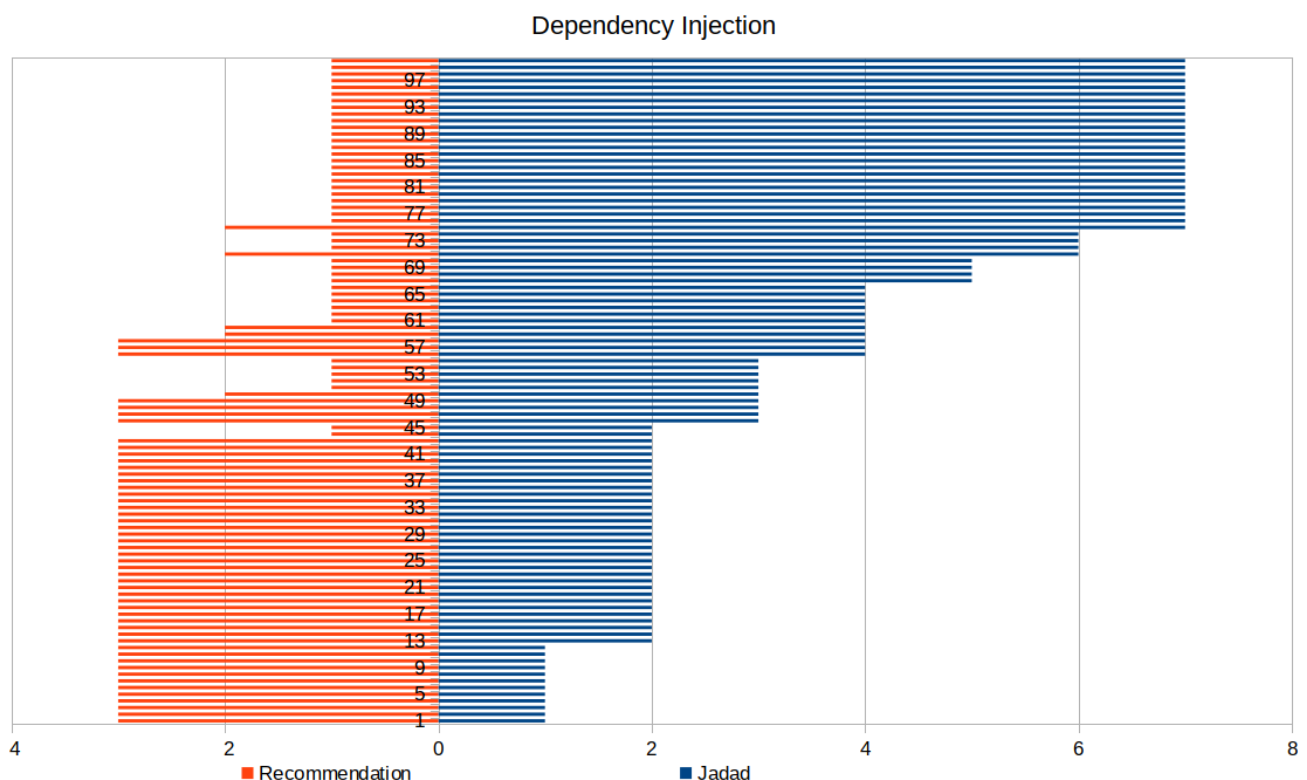
### 3.6.4 new in constructor

This practice is very difficult to search for on its own terms because there is no common name for the practice as there is with singletons or setter injection. However, this practice is the exact opposite of **dependency injection** where dependencies are injected into the class rather than instantiated inside it. Rather than searching for new in constructor the same search was completed for dependency injection. Any recommendation to favour dependency injection is a recommendation to avoid instantiating dependencies inside classes.

This approach may skew the results and the results may include articles which do not explicitly mention this practice, however as the alternative to dependency injection is creating dependencies inside the same class any results in favour of dependency injection can be considered results arguing against constructing objects in constructors.

### Results

Figure 3.7 shows the results for the meta-analysis of Dependency Injection.



**Figure 3.7:** Meta-analysis results: Dependency Injection



## **Key Findings**

*See results for Dependency Injection*

### 3.6.5 Service Locator

This section covers the results for the meta-analysis for the service locator pattern.

#### Methodology

The search keywords used were "service locator pattern" as searches for "service locator" yields results which are overwhelmingly unrelated to programming.

#### Results

Figure 3.8 shows the results for the meta-analysis of Service Locator.

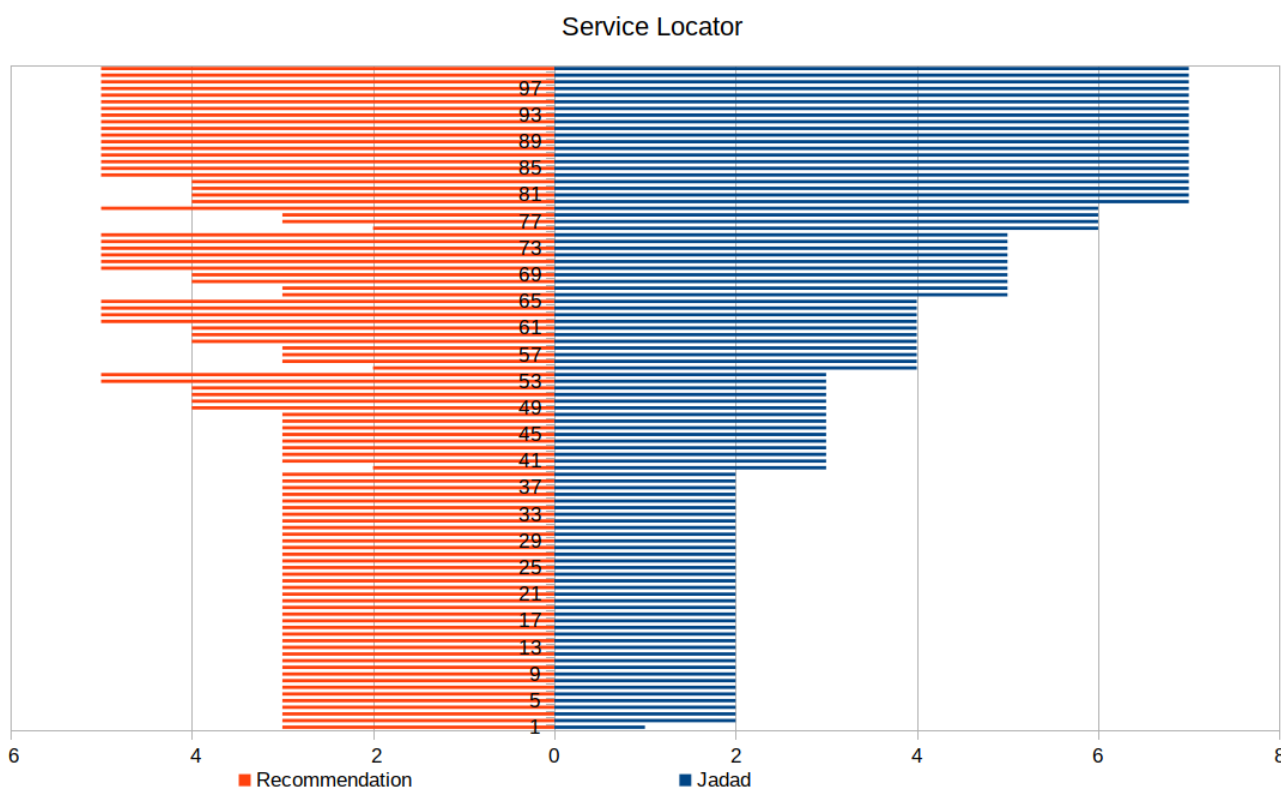


Figure 3.8: Meta-analysis results: Service Locator

#### Key Findings

- There is a clear correlation between the recommendation score and Jadad-style score. With the the exception of one neutral recommendation, every article other with a Jadad-style score of 6 or higher recommends avoiding the practice.
- The mode recommendation score is 3 (neutral/no recommendation). A simple tally would

imply that most people do not recommend either using or avoiding the practice.

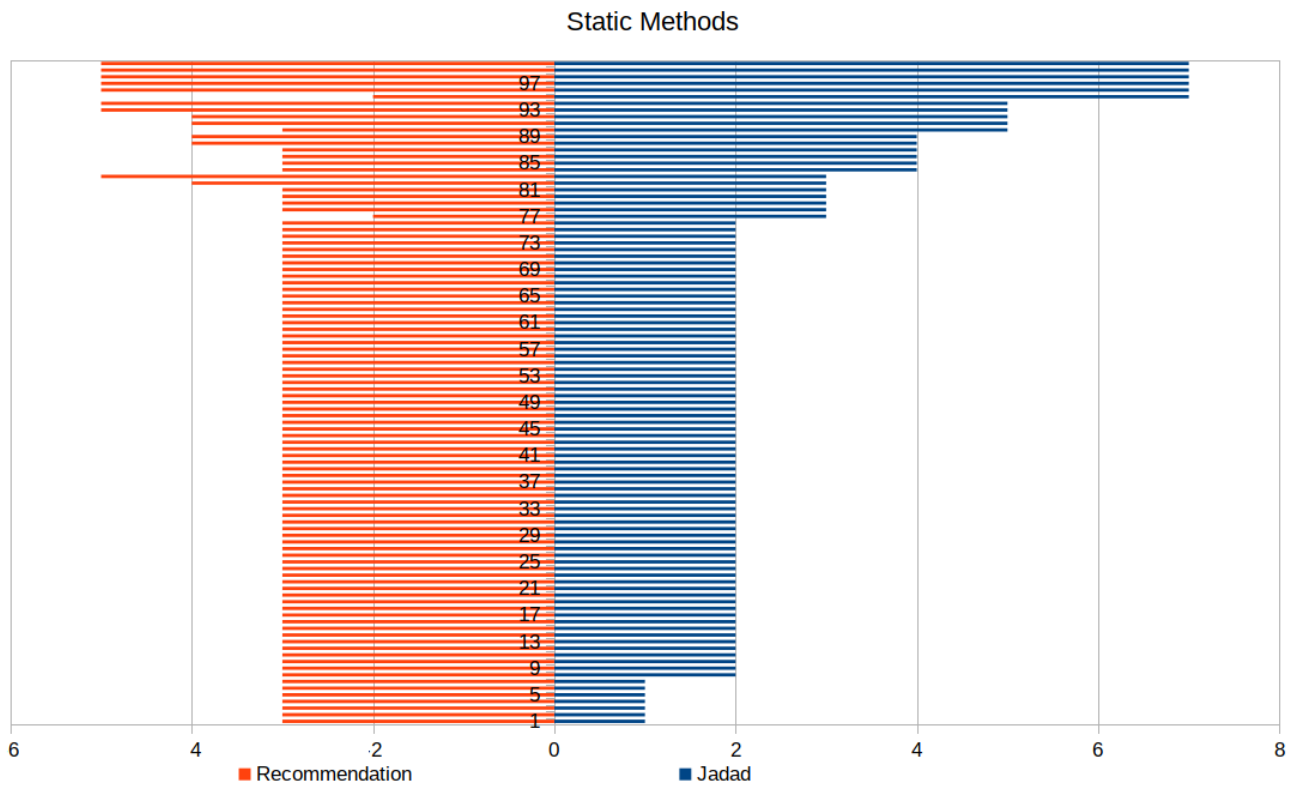
- The mean recommendation score is 3.65. A metric that did not account for analytic rigour would show developer's attitude to be somewhat unfavourable.
- The Jadad-style score correlates with the recommendation. The higher the Jadad-style score, the less likely the author is to recommend using Service Locators.
- 56% of articles about Service Locators discuss negative implications of the pattern, the highest percentage of any practice analysed.

### 3.6.6 Static Methods

This section covers the results for the meta-analysis for static methods.

#### Results

Figure 3.9 shows the results for the meta-analysis of Static Methods.



**Figure 3.9:** Meta-analysis results: Static Methods

#### Key Findings

- There is a clear correlation between the recommendation score and Jadad-style score.
- The mode recommendation score is 3 (neutral/no recommendation). A simple tally would imply that most people do not recommend either using or avoiding the practice.
- The mean recommendation score is 3.19. A metric that did not account for academic rigour would show developer's attitude to be mostly neutral.
- The Jadad-style score correlates with the recommendation. The higher the Jadad-style score,

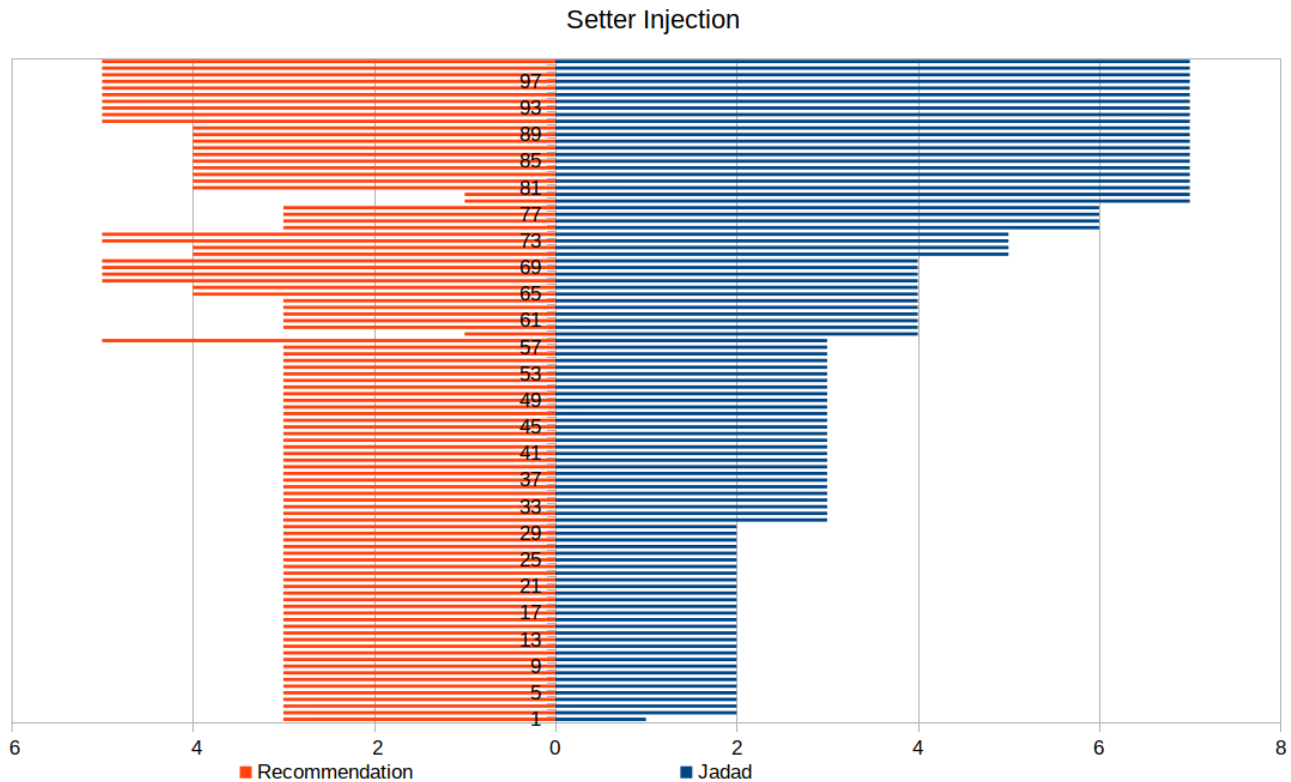
the less likely the author is to recommend using static methods.

- Considering how common static methods are, there is surprisingly little discussion about when they should be used.

### 3.6.7 Setter Injection

#### Results

Figure 3.10 shows the results for the meta-analysis of Setter Injection.



**Figure 3.10:** Meta-analysis results: Setter Injection

#### Key Findings

- There is a clear correlation between the recommendation score and Jadad-style score. However, one article with a Jadad-style score of 6 and one with 7 recommend using Setter Injection over constructor injection. Further research could be done to determine where the point of disagreement is or whether it's a specific programming language or framework being discussed in these articles.
- The mode recommendation score is 3 (neutral/no recommendation). A simple tally would imply that most people do not recommend either using or avoiding the practice.
- The mean recommendation score is 3.37. A metric that did not account for analytic rigour would show developer's attitude to be mostly neutral.

- The Jadad-style score correlates with the recommendation. The higher the Jadad-style score, the less likely the author is to recommend using setter injection.
- 58% of articles discuss alternative approaches, the most of any practices analysed.

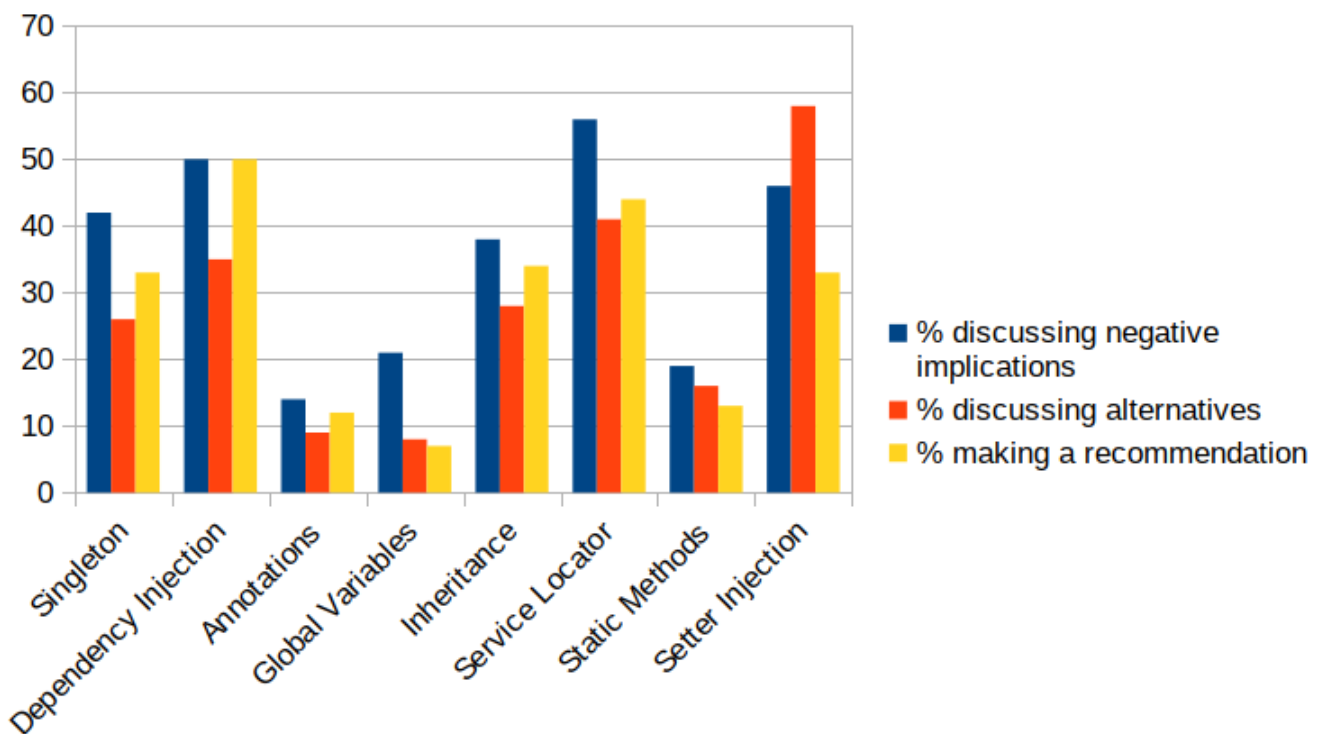
### 3.7 Meta-analyses overall conclusions

Most programming practices are taught using examples, but very few articles regarding any practice discuss alternative approaches or when/where a given practice should be used over another.

This is potentially a serious problem for students and junior developers as they are taught practices without also being taught negative side effects of using those practices or alternative solutions to the same problem.

This is similar to teaching students of carpentry to use a jigsaw without teaching them about hand saws or chainsaws and where each one is useful.

*if all you have is a hammer, everything looks like a nail \*- Proverb\**



**Figure 3.11:** Depth of discussion about each practice

Figure 3.11 shows each practice broken down by the number of articles that discuss negative implications of the practice, discuss alternative approaches and make a recommendation.

The following conclusions were made based on over 800 articles being analysed across 8 bad practices:



- Regardless of the practice being discussed, authors who consider alternative approaches come to different conclusions than those who do not.
- Despite global variables being widely considered bad practice by even junior developers and known to cause issues since 1973 (Wulf et al, 1973) only 16% of articles about global variables recommend against their use and only 8% discuss alternative approaches.
- Of the 847 articles analysed, 33% mention the negative implications of the practice being discussed, while 26% discuss alternatives and 27% make a recommendation of when/where to use the practice. Although as demonstrated in Figure 3.10, this varies significantly by the practice being discussed.
- Although it was expected that simpler practice like global variables would receive more discussion regarding the problems, advanced level practices such as service locators have a higher percentage of articles discussing their negative aspects than simpler practices such as global variables and inheritance. It is hypothesised that this is because:
  1. People using these more advanced practices are using them to solve a specific problem and are more likely to consider alternatives
  2. Professional programmers are not interested in talking about basic functionality like global variables and inheritance and would rather discuss more advanced concepts. Lower level programmers who are still using global variables are less likely to write articles.

### 3.7.1 Possible further research

Further research could be performed on the data already gathered. The following research questions are proposed:

1. If articles were broken down by publish year, are developers becoming more or less favourable of a practice over time?
2. Does language choice affect the outcome? If articles were broken down by programming language in each article would different languages yield different results. The existing data suggests it doesn't as the current language agnostic results are internally consistent but a proper analysis may yield some surprises.

3. As roughly two thirds of articles about programming practices teach only how to use it and not where or if it should be used, does this have an impact on developers progression? What would be the effect of teaching developers up front that these practices introduces maintainability problems?

## 3.8 Chapter Review

This chapter outlined the problem of subjectiveness in the definition and identification of bad practices, and outlined a methodology for determining whether developers generally considered given practices "bad".

As a result of this chapter, the following was produced:

1. A metric for grading articles about a programming practice on their academic rigour.
2. A methodology for performing a meta-analysis for articles about any given programming practice.
3. An academic paper entitled *A Methodology for Performing Meta-analyses of Developers Attitudes Towards Programming Practices* was presented at the Proceedings of the 2019 Computing Conference (Butler, 2019) to describe the methodology for performing the meta-analyses. This paper is available in *appendix VII*.
4. For each bad practice, 100 articles were analysed and results used for meta-analyses. The raw data gathered for the meta-analyses in this chapter is available in *appendix VI*.
5. Results of the meta-analyses were generated.

## 4. Creating a metric

## 4.1 Introduction

The previous chapter showed that the identified bad practices genuinely are considered "bad" by developers. These can now be used as part of a metric which scans source code for these practices.

A metric was created for identifying bad practices in source code and grading it. Source code is analysed and two results are returned:

1. A grade for the software. The grade is a score from 0-100 based on the quality of the software with 100 meaning no bad practices identified and 0 meaning that every class contains bad practices.
2. A list of classes/lines which contain bad practices and instructions for how a developer could remove the practices.

The metric can be used to compare the flexibility of two different pieces of software and in isolation to identify where code flexibility can be improved by highlighting known bad practices.

## 4.2 Aims and Objectives

### 4.2.1 Aim

Create a metric for measuring the previously identified bad practices found in source code. The metric will need to account for the size of a project and severity of each bad practice.

### 4.2.3 Objectives

1. Calculate weightings for each bad practice based on the severity of each bad practice.
2. Design a metric which grades software based on frequency of bad practices.
3. Test the metric to ensure it produces a spread of results.

## 4.3 Methodology

This section outlines the methodology used to create the metric for grading source code flexibility.

### 4.3.1 Introduction

A base metric was created and refined during the development process. The actual grades generated are not important as long as one software grade can be compared to another.

### 4.3.2 Software Size

A meaningful metric cannot be a simple tally of the number of bad practices. A piece of software could be given a score of 5 indicating that it contains 5 bad practices. This alone is not useful when comparing different software.

Using a tally, two different pieces of software could be graded 5, however if one contains 100 lines of code and the other contains 10,000 lines of code the scale of the problem is different between the pieces of software. A developer working on the first has a higher probability of encountering maintainability issues than a developer working on the second larger project.

For this reason the size of a project will be factored in to the overall score of the software.

#### Measuring software size

To account for project size the question *how large is the project?* must be answered.

There are several methods for measuring software size (as outlined in the literature review in chapter 1) including:

1. Source Lines of Code (SLOC). A count of the number of lines of source code. This method can be unreliable as coding style can affect the number of lines. For example, brace position, comments and whitespace.
2. Cyclomatic Complexity. A count of the number of if statements, loops and functions. This is consistent across coding and commenting styles and gives a much more accurate indication of code size.
3. Number of methods/number of classes. The building blocks of an Object-Oriented application are classes.

## Number of classes

Number of classes was chosen for the metric for the following reasons:

1. Number of classes is a consistent, unambiguous, easy to calculate figure for any project which doesn't change with coding style choices such as brace positions, whitespace, comments and other coding convention differences.
2. Classes are the building blocks of an object-oriented program and every project will have them.
3. Several bad practices exist at a class level. For example, in relation to properties or constructors. Other bad practices exist due to coupling between classes and the practice will affect the class regardless of the size of the class.
4. A class' API is what makes it maintainable or not. Bad practices such as using singletons, global variables or inheritance make the entire class difficult to maintain and move between projects. For example, if a class uses setter injection the whole object can exist in an incomplete state and introduce bugs, regardless of the size of the class.
5. SLOC and Cyclomatic Complexity would produce inconsistent results. Bad practices which exist at a class level such as use of inheritance or new in constructor would be diluted by large classes. For example, a 2000 line class which uses inheritance would potentially see a lower score than a 200 line class with the same bad practice, despite having the same effect on the overall maintainability/portability of the code.

### 4.3.3 Severity

Are all bad practices equally bad? Are annotations as bad as Global Variables. If not, how can the difference be quantified?

As discovered earlier in the the research, bad practices are bad because they introduce one or more negative traits such as *Action at a distance*, *Tight Coupling*, *Broken encapsulation*, *Broken Single Responsibility Principle*, etc.

Annotations only introduce four of these problems, whereas global variables introduce all of them.

Because of this, global variables can be considered quantifiably worse than annotations in the context of this research.

Each bad practice was given a *severity* score based on how many of these traits were introduced as follows:

**Table 4.1:** Bad practice severity ratings

<b>Bad practice</b>	<b>Negative traits introduced</b>	<b>Severity</b>
Annotations for Configuration	Broken Encapsulation, Single Responsibility Principle, Action At A Distance, Unnecessary Coupling	4
Global/Static Variables	Tight Coupling, Broken Encapsulation, Single Responsibility Principle, Action At A Distance, Global State	5
Inheritance	Tight Coupling, Broken Encapsulation, Single Responsibility Principle	3
New In Constructor	Tight Coupling, Broken Encapsulation, Single Responsibility Principle	3
Service Locator	Tight Coupling, Broken Encapsulation, Single Responsibility Principle, Law of Demeter	4
Setter Injection	Temporal Coupling, Broken Encapsulation, Action at a Distance	3
Singleton	Tight Coupling, Broken Encapsulation, Global State, Action At a Distance, Single Responsibility Principle	5
Static Methods	Tight Coupling, Broken Encapsulation, Single Responsibility Principle, Unclear Dependencies	4

This rating could potentially be further refined. Is *Broken Encapsulation* worse than *Tight coupling*? This could be a question for further research, however this would be very difficult to quantify in an unopinionated way. As such, each negative trait will be weighted identically for this metric.

#### 4.3.4 What to grade

As bad practices generally operate at a class level, grading individual classes is beneficial. To make feedback meaningful and enable users to identify the source of problems, each class will be given an individual grade. From this, users will be able to identify which classes contain bad practices and which do not

The overall grade for the project will be an average class score. One class with a low score in a large project will still result in a high overall project score.



### 4.3.5 Grading range and visualisation

There are numerous methods to express the grades of classes or projects. For example, and not limited to, A-F, 1-5 stars, 0-10, 0-100 (percentage). Each of these grading ranges would be feasible but 0-100 was chosen as it gives a finer level of granularity. A-F gives only six (or five when the *E* grade is omitted) grade bands, 0-10 gives only 11. With 0-100 one project can be graded at 82 with another at 83. This additional granularity should make comparing projects easier.

1-10 with decimal points was considered e.g. 8.3 or 8.2 which would be equivalent to 82 or 83, but this requires more characters to express the same score and as such 0-100 was the grade which was eventually chosen.

By contrast, 1-1000 (or any higher number) is less common and likely less intuitive. If finer granularity is required a decimal can still be added to a 1-100 grade.

Using a numerical scale, as long as decimals can be added to refine the granularity, the actual range does not matter as long as it is made clear to users what a high score is and what a low score is.

#### Visualisation

An advantage of A-F is that is a grading system used frequently in education and users would likely be familiar with the fact that *A* is a *better grade* than *F*. With 0-100 (or 0-10) there is ambiguity *Does 100 mean 100% e.g. a perfect score? or Does 0 mean zero issues detected?*

To reduce this ambiguity, when grades are displayed, they will be displayed with visual cues:

1. Colour coding. Low scores (<30) will be displayed in red, medium scores in yellow, high scores in green and perfect scores in bright green. The numbers for the bands could be adjusted, however the colour is only indicative. If someone sees green and a score of 87 they should be able to infer that higher is better.
2. Progress bar. By including a progress bar of 0-100, coupled with colour coding, it should be quickly apparent that 100 is a high score and 0 is a low score.

### 4.3.6 Grade Calculation

To make it possible for users to identify where issues are detected each class will be individually graded. This individual class score can be utilised by users to identify the classes where

improvements can be made.

Each class score is calculated by identifying bad practices in the code weighted by severity and size.

The number of methods was taken into account as it will give more of a spread of results and give more useful data to users. A class with 1 method and 1 bad practice should not give the same score as a class with 10 methods and 1 bad practice as practically, a much lower percentage of the code being flagged is genuinely an issue.

Initially, for each bad practice the processes outlined in figure 4.1 was followed to calculate the score.

```
Score = 0;
for each bad practice identified:
    Score = Score + IssueSeverityRating;
ClassGrade = 100 - ((Score/NumClassMethods)*100)
```

**Figure 4.1:** Initial grade calculation

Where:

The contents of `ClassGrade` is the grade for the class.

`IssueSeverityRating` is the severity rating of the issue identified.

`Score` is a tally of severity ratings for each bad practice identified (The same bad practice can be identified multiple times in the same class)

`NumClassMethods` is the number of methods in the class.

This process starts off assuming the class is perfect (100 score) and subtracts points based on the severity of the issue and the size of the class.

This method was tested using the top 20 PHP projects listed on packagist (Packagist, n.d.) the most popular repository for hosting PHP libraries (TutsPlus, n.d.).

### 4.3.6 Model refinement

After examining the results, most projects had numerous classes with a score of 0 even though they only had a single negative trait.

These zero score classes were usually where classes had few methods and used inheritance or using new in constructor. Because the grades were weighted on number of methods, any class with inheritance and few methods was graded zero.

Regardless of the class size, inheritance has the same effect of tight coupling on the *class*. For this bad practice (among others) it makes little sense to weight based on class size because the bad practice can only exist at one specific point in each class and introduces tight coupling throughout the class.

Other bad practices, like static methods can exist multiple times in a class and at different locations should and therefore should be weighted by method.

As there are now two different types of weighting (class level and method level) the metric was changed to incorporate this.

Class level bad practices were given 50% of the grade and method level bad practices were given the other 50%. The result of this is that even if a class has all the class level bad practices, if it has no method level bad practices it will still get a score of 50%. On the other hand, classes which have zero class level bad practices can at worst score 50%. For a class to score less than 50% it must contain both class level bad practices and method level bad practices

50% for each was initially chosen as a metric as there is no clear justification for 60/40, 70/30 etc. After testing 50/50 it was discovered that this gave a good spread of results with a lot more granularity than applying all bad practices at method level.

The bad practices which are applied at class level are: - New in constructor - Inheritance

The updated metric was calculated as shown in figure 4.2:

```
MethodScore = 0;
ClassScore = 0
for each class level bad practice identified:
    ClassScore = ClassScore + IssueSeverityRating;
for each method level bad practice identified:
    MethodScore = MethodScore + IssueSeverityRating;
ClassGrade = 50 - (((MethodScore/2)/NumClassMethods)*50);
```

$$\text{ClassGrade} = \text{ClassGrade} + (50 - (\text{ClassScore}/2) * 50)$$

**Figure 4.2:** Refined grade calculation

To keep scores for method level bad practices the same as they were previously, all tallies were halved because each was only being applied to 50% of the total.

The result of this is that there are effectively two different metrics from 0-50 being added together to get the final grade. One 0-50 for class level bad practices and one 0-50 for method level ones.

### 4.3.7 Further refinement

Due to the severity of inheritance (4 severity rating) the above still did not produce meaningful results in some instances.

Any class with inheritance would be calculated as shown in figure 4.3:

$$50 - ((4/2) * 50)$$

**Figure 4.3:** Initial inheritance calculation demonstration

This calculates to -50 due to how severely inheritance is weighted. Even rounding up to 0 this posed a problem: A class with inheritance will always get 0% for its class level bad practices.

A class which had inheritance would have the same score as a class with inheritance and new in constructor despite one class being quantifiable worse: two bad practices instead of one.

Several different weightings were tested and the calculation shown in figure 4.4 was chosen:

$$\text{ClassGrade} = \text{ClassGrade} + (50 - (\text{ClassScore}/5) * 50)$$

**Figure 4.4:** Refined class grade calculation

/5 was chosen as:

- Any class with inheritance will lose 40% marks
- A class with inheritance and new in constructor will lose 50% marks

The weighting could be fairer such that a class with inheritance lost 25% and lost another 25% for new in constructor but this did not accurately reflect the severity of class level bad practices.

From the previous iteration where inheritance was an immediate 0% score making inheritance into a 75% score did not reflect the severity of the bad practice.

Instead, by using /5 and inheritance costing 40%, it better expresses the severity to the user.

### **A note on numbers**

One important consideration is not that each class is given a completely meaningful number but that the flexibility of one class can be directly compared to the flexibility of another. The actual number being displayed matters less than a metric that gives more granularity. For example, if every class was either 0 or 100 that would be less useful than a metric which graded classes at any point in between that.

Any number associated with a class is always going to be arbitrary without the context of something else to compare it to. The only time the number is useful is when comparing two classes, two projects, etc which have been graded using the same metric.

As such, the result of these numbers can be further refined without any detriment to the metric or any tool which uses it. The important consideration is that any comparisons done can only happen between classes (or projects) which are scanned using the same revision of the metric.

### **4.3.8 Project score**

The project score is simply the mean class score and calculated using the process shown in figure 4.5:

```
Total = 0;
for each class
    Total = Total + ClassGrade
ProjectGrade = Total / NumberOfClasses
```

**Figure 4.5:** Project score calculation

By using a mean, project size is taken into account. A project with a low number of classes with issues will get a lower score than a project with a large number of classes. This is useful as it gives an indicative figure for the percentage of the project affected by bad practices.

## 4.4 Results

The completed metric, labelled **Insphect**, was tested on the top 20 PHP projects on packagist at the time of writing.

The "Top packages" list from packagist was chosen as the packages are by different authors, different sizes and perform a variety of different tasks. This should be representative of general PHP libraries.

Note: Packages were omitted if they:

- Only contained interfaces (no classes)
- Were polyfill packages. These tend to be single class/single method utility packages which act as backward/forward compatibility tools. As such their design/function is very limited.
- The phpunit/phpunit package. As most packages contain PHPUnit tests, any class which extends PHPUnit is ignored as tests should not be scanned for flexibility. However, the tool developed detects phpunit classes as tests so it cannot be scanned. This is currently a limitation of the tool which may eventually be worked around.

**Table 4.2:** Results table for 20 projects

Project	Insphect	Number of classes
doctrine/lexer	100	1
sebastian/exporter	100	1
webmozart/assert	33.33	2
phpdocumentor/reflectioncommon	100	2
doctrine/instantiator	73.67	3
symfony/event-dispatcher	76.43	9
guzzlehttp/promises	81.9	9
sebastian/diff	88.76	13
symfony/process	66.74	15
symfony/finder	58.06	21
guzzlehttp/guzzle	67	30

Project	Insphect	Number of classes
phpdocumentor/typeresolver	84.86	32
phpdocumentor/reflectiondockblock	71.84	34
myclabs/deep-copy	86.32	35
doctrine/inflector	45.77	39
symfony/translation	72.64	53
symfony/http-foundation	68.99	65
symfony/console	70.25	73
monolog/monolog	66.19	99
phpspec/prophecy	68.58	178

Table 4.2 shows the raw scores generated by the metric for the top 20 packages on packagist.

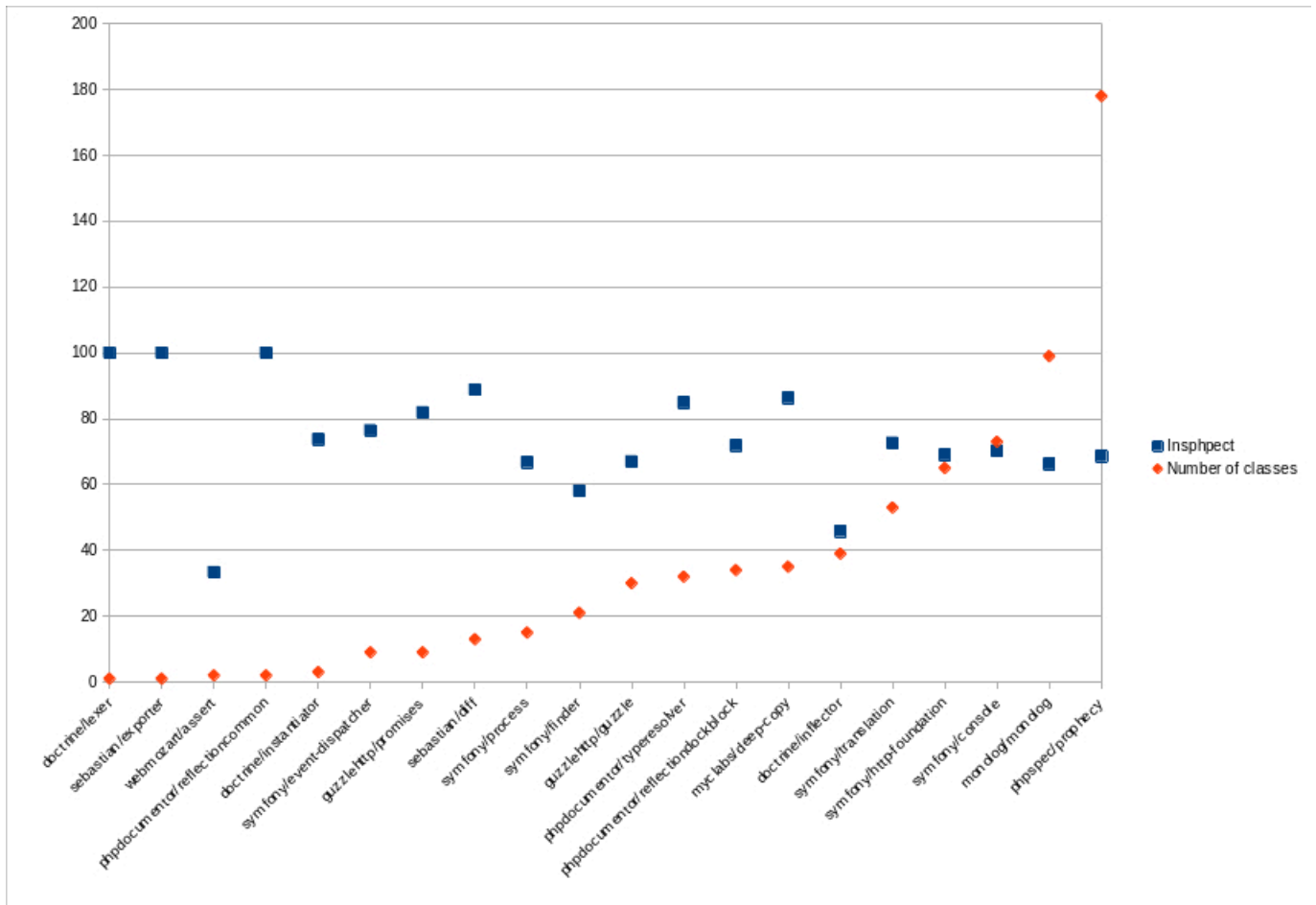
#### 4.4.1 Preliminary Evaluation

A preliminary evaluation was conducted to test the suitability of the metric before a complete evaluation by other developers.

This evaluation was performed to determine whether the grades generated gave an even spread of results and to check that the results were not biased by project size.

The results show a range of different projects of varying sizes by different authors and this resulted in a wide spectrum of scores. As expected, project scores do not directly correlate with project size.





**Figure 4.6:** Graph of results and number of classes

This is by design, the score should not correlate with project size and shows that the metric works similarly for both large and small projects.

The lowest score was 33.33 and the highest score was 100. Although this may indicate that the range is not wide enough as there are no results in the sample which are below 33, this is also to be expected. As the project score is the mean of all classes, to get a low score a project would have to have issues in all or most classes in the project.

Even in a project where half the classes scored 0, the mean would be much higher.

### Sample size

A greater number of projects could have been analysed for the preliminary evaluation, however this is evaluating whether the metric produces a meaningful spread of results the sample size does not matter as much as the randomisation of the sample.

In total 714 classes were analysed across the 20 projects, 510 of the classes contained issues and 1078 issues were detected in total. The mean class grade was 74.04 with the lowest at 0 and

highest at 100.

A more complete evaluation will be performed at a later stage by asking other developers to upload their work and evaluate the results themselves.

#### **4.4.2 Conclusion**

The metric gives a reasonable spread of results and is not biased towards large or small projects or projects by particular authors. There is likely room for improvement but as a proof-of-concept it is ready to be tested and used by real developers in order to gather their feedback.

## 4.5 Chapter review

In this chapter a metric was developed, and revised, for grading software based on the frequency of bad practices it contains. Preliminary results, after revising the calculation, showed that the metric produced a meaningful spread of results and could be used to measure the flexibility of different software projects by notifying the user of bad practices in the source code.

## 5. Testing the metric by creating a tool

## 5.1 Introduction

While developing the metric in the last chapter, software was developed alongside the metric, in the first instance to test that the metric produced meaningful results without having to manually analyse thousands of lines of code and secondly to enable other developers to test it for themselves.

## 5.2 Aims and Objectives

1. Build a software tool that analyses source code for previously identified bad practices.
2. The tool should use the metric which is to be developed in parallel to the software and produce a grade.
3. In addition to the overall software grade, the software should display which classes/lines contain the issues and information about why the issue was flagged up and a generic explanation of how to fix the issue.
4. If possible, as a proof of concept, the tool could be extended to include automated corrections.

## 5.3 Design

A tool designed to test and evaluate the metric was created alongside the metric. This was done in parallel in order to test that the metric produced a meaningful spread of results rather than grading everything at extremes.

As such, the metric was modified during development of the tool.

### 5.3.1 Web based or application.

A web based tool was chosen that users did not need to download and install an application, they can upload a project or provide a git repository URL to have it analysed by the tool.

There are both advantages and disadvantages of a web based application as shown in table 5.1.

**Table 5.1:** Pros/Cons of a web-based tool

Pros	Cons
Does not require downloading or installing any software	User data is sent to the server. The user may not be comfortable uploading their code to a service to which they are unfamiliar
Users can try the software with minimal effort or risk, they don't need to install software from an unknown source	Relies solely on the server, too many requests at once will slow the site down and since this may be fairly computationally expensive may require a server with a strong CPU.
Platform independent, the software does not have to be developed for windows, mac or linux	
Evaluation and error detection is easier as all results are stored on the server rather than the user's own PC	
It's potentially easier to design a user friendly and aesthetically pleasing HTML/CSS website than a desktop application	
The software can be improved without users needing to update to the latest version	

Due to the positives outweighing the negatives, a web-based tool decided upon though this project could just as easily have been built as a desktop application.

### 5.3.2 Language choice

The tool could have been built for any Object-Oriented language. Although any language could be

used to build the tool and as a target language for analysis, it was anticipated that it would be significantly easier to analyse code in the language it was written in. For example, a tool written in Java to analyse Java code or a tool written in Javascript to analyse Javascript, etc as the tool would be able to utilise existing parsers, class loading mechanisms and even execute the code to trace what it is doing.

As such, the language chosen was the language the tool was written in and the target language which is being analysed.

The three candidate languages were Java, Javascript and PHP.

Javascript was avoided because its object model is significantly differed to other Object languages. It does not support strong typing and Inheritance and class loading are handled very differently.

PHP was chosen over Java because:

1. It is open source and would have been possible to make a custom build of the PHP interpreter for the tool if required.
2. There are thousands of existing projects which could be used to test the metric.
3. PHP allows easily controlling how classes are loaded via autoloaders. This enabled easier development as custom code could be easily injected into classes on the fly, just before they are parsed. Classes can be substituted with doubles and mocks on the fly in a way that is considerably more difficult in Java.
4. PHP is used for many web based projects and could be easily used to build a web-based tool. PHP already includes tools for parsing PHP code and using the same language for the tool as the language being analysed potentially makes the tool more efficient to build.
5. The developer has significant experience in the language.

### **Implications of using PHP**

The language choice, regardless of which was ultimately chosen, may have an impact on eventual findings. The following could be impacted by language choice:

- Selection bias of respondents. If the tool is developed for a particular language, developers of that language will be over-represented in any trials.
- Language nuances. Due to the differences in languages, some languages may require things to be done in certain ways. For example, PHP requires extending the `Exception` class to create custom exceptions. Projects may be build differently in different languages due to

restrictions imposed by the language itself.

However, these differences will not affect the bad practices being identified or direct results of the tool. Feedback can still be gathered from users of other languages by presenting pre-generated reports.

Due to the scale of the project, building a tool to identify bad practices across multiple different languages was infeasible. For the reasons listed above, PHP was chosen. Regardless of which language was ultimately decided upon these issues would be present.

In future the tool could be developed for multiple languages and results compared between users of different languages and projects written in each language. This is beyond the scope of this research at this stage.

### **5.3.3 Specification**

From a user's perspective, the website is fairly basic:

1. The user either uploads a zip file or provides a git repository URL.
2. The sever extracts/downloads the files.
3. The source code is scanned for the previously identified bad practices and a report is generated.
4. The report will list all the classes in the project to allow users to see which classes contain issues.
5. Clicking a class will display the code for the selected class and highlight any issues in the code.
6. The tool will also include background information on the project and a way of reporting bugs.
7. To collect feedback, a survey will be included as part of the website.

The challenging part of the build is the back-end that analyses the code for bad practices. The user will not see this and will only be given the generated report.

### **5.3.4 Report format**

The report will display a grade for the project and classes. The first page of the report will clearly display the project score along with a list of classes in the project along with their grade.

Clicking on a class will take the user to another page which displays the grade for the class, all the code for that class along with highlighting the issues that were detected. Lines with issues will be highlighted to the user and clicking the line will display information about the issue that was



detected.

This format is inspired by reports generated by Scrutinizer (Scrutinizer-CI, n.d.) and PHPUnit's code coverage report (PHPUnit, n.d.) which provide different information but are also used to analyse projects.

### 5.3.5 Methodology

The tool was built loosely using Agile techniques with a focus on Test-Driven-Development. All code was built as individual units with its own test prior to being put together to construct the final project.

Each stage of the software development process (excluding the GUI) followed these steps:

1. Tests were written to describe what the unit should do.
2. Code was written until all tests passed.
3. If the unit needed extension or modification during development, new tests were added and tests re-run to ensure nothing was broken.

The following modules were developed:

### 5.3.6 Unit 1 - Utility class 1: Navigating code

A set of tools were created for parsing and navigating around source code. A class which tokenized PHP code and could move to the next/previous token, match brackets and navigate around to the next class/function or other block was created.

Sample tokenizer usage is shown in figure 5.1.

```
$tokenizer = new Tokenizer(token_get_all($file));  
//Extract class name  
$className = $tokens->next('T_CLASS')->next('T_STRING')->string();
```

**Figure 5.1:** Sample Tokenizer usage

This would extract the string `ClassName` from the code shown in Figure 5.2 by moving to the corresponding tokens.

```
<?php
namespace Example;
class ClassName {
}
```

**Figure 5.2:** Sample Tokenizer usage (b)

The complete code and unit test suite is available on request and will be made open source after this research is published.

### 5.3.7 Unit 2 - Utility class 2: Calculate namespace

Because PHP supports local aliases for class names, the tool will need to work out the Fully-Qualified-Class-Name as it is referred to in global scope.

A class was written to determine the global, Fully-Qualified-Class-Name for any given local class name as shown in figure 5.3.

```
namespace Example;
use \Library\Some\Class as SomeClass;
class ClassName {
}
```

**Figure 5.3:** Sample local class aliasing in PHP

In this scope, the class called `SomeClass` is. A tool, `NamespaceResolve`, was written to convert the local alias to the global class name as demonstrated in figure 5.4

```
$source = file_get_contents('file.php');
$resolver = new NamespaceResolve($source);
```

```
$globalName = $resolver->resolve('SomeClass'); // \Library\Some\Class
```

**Figure 5.4:** Tool for resolving global class name

### 5.3.8 Unit 4 - Project

A unit was created for handling projects. This unit is used to load a set of files and has the ability to attach issues to classes in the project.

A `Project` instance is given a directory and recursively loads all files with a `.php` extension, it can then be used to get information about the files in the project or associate an identified issue with a particular class.

Sample `Project` API is shown in figure 5.5.

```
$project = new Project('/path/to/files');  
//Returns an array of all classes in the format Name => Tokens  
$classes = $project->getClasses();  
//Returns an array of all files in the format Path => File  
$classes = $project->getClasses();  
//Gets the file by class name  
$file = $project->getFileForClass($className);  
//Registers an identified issue  
//Issues are things like global variables and singletons identified by  
earlier research  
$issue = new Issue($className, //Name of class with issue  
                  $lineNumber, //Line number where issue was  
detected  
                  $method, //Name of method which detects  
issue (Can be left blank)  
                  $descriptionOfIssue, //Description of issue e.g. "New in  
constructor"  
                  $stringOfCode, //The line or block of code  
containing the issue e.g. $this->object = new Object();  
                  $typeOfIssue, //Consistent type e.g.  
NEW_IN_CONSTRUCTOR for use in the metric  
                  $pathToDocumentation //Path to a file containing the
```

```
explanation of why this is an issue
        );
$project = $project->registerIssue($id, $issue)
```

**Figure 5.5:** Project class usage

### 5.3.9 Unit 5 - Scan for bad practices

A unit was created for each of the bad practices. Each unit is independent from the others and follows the interface shown in figure 5.6.

```
interface Rule {
    public function run(Project $project): Project;
}
```

**Figure 5.6:** Rule interface

Each Rule has access to the project, scans each file or class for the bad practice it is intended to identify and returns a new bad practice. Each Rule looks follows the interface and sample usage is shown in figure 5.7.

```
class GlobalVariables implements Rule {
    public function run(Project $project): Project {
foreach ($project->getClasses() as $className => $tokens) {
            if ($this->lookForGlobals($tokens) == true) {
                $project = $project->registerIssue(uniqid(),
new Issue($className, $lineNo....));
            }
        }
    }
}
```

**Figure 5.7:** Sample rule implementation

A Rule instance was created for each bad practice.

As each bad practice uses its own class which follows this structure, the tool can be very easily extended with new bad practices.

### 5.3.10 Unit 6 - Combining the rules

A unit was created to tie together the work done to this point. An instance of `Inspect` takes a project and some rules and scans the project for any assigned rules.

Sample usage is shown in figure 5.8.

```
$inspect = new Inspect();  
$inspect = $inspect->addRule(new Rule\GlobalVariables());  
$inspect = $inspect->addRule(new Rule\StaticMethods());  
$inspect = $inspect->addRule(new Rule\NewInConstructor());  
$project = $inspect->scan($project);
```

**Figure 5.8:** Sample `Inspect` instance

This approach was chosen as it is beneficial for testing as it is possible to test this process with just one rule enabled.

### 5.3.11 Unit 7 - Metric

After scanning a project with unit 6, the `$project` instance contains a list of identified issues.

The next stage was creating a unit for grading the project.

Sample usage is shown in figure 5.9.

```
$result = new Result($project);  
//Calculates an overall grade for the project  
$grade = $result->getGrade();  
//Returns an array in the format ClassName => Grade  
$classRatings = $result->getClassRatings();  
//Returns the percentage of classes with issues
```

```
$percentIssues = $result->getPercentageIssues();
```

**Figure 5.8:** Code for generating the grade of a project

The metric itself was changed several times during the course of development. This is documented in the previous chapter.

### **5.3.12 Unit 8 - Class Issues**

The next unit was designed to extract the issues for a particular class. This class can be used in the process of generating a report for a particular class.

It includes methods for extracting all the code for a class formatted as HTML, with issues highlighted.

### **5.3.13 Unit 9 - GUI**

The final stage was putting a GUI on top of the existing code. This required some minor changes to some of the other units for tracking progress to display a progress bar to the user while the uploaded code was being analysed.

The GUI design was changed over several iterations through discussions with the supervisor.

### **5.3.14 Unit 10 - Automated corrections**

One of the projects overall additional objectives is exploring the idea of automated corrections. This was implemented for one the bad practice, *new in constructor* and partially implemented for *static methods*.

This was technically very challenging as it required rewriting the code to remove the bad practice.

## **5.4 Implementation**

The web based tool, named Insphpect and launched on <https://insphpect.com/> was built in parallel to the metric in order for the metric to be tested.

For portability, Insphpect was built inside a containerised environment using Docker. The project can be moved to any machine running docker and launched with `docker - compose up`.

The `docker - compose . yml` configuration file is included in the source code and uses the following services:

- nginx web server
- PHP
- certbot (for SSL certificates)

The tool uses the following third party libraries:

- PHPUnit (PHPUnit, n.d.) unit testing suite
- level-2/transphorm (Transphorm, n.d.) Template engine
- level-2/dice (Transphorm, n.d.) Dependency Injection Container
- RadialBar (RadialBar, n.d.) for drawing circular progress bars

Screenshots of Inspspect

Figure 5.9 shows the home page of Inspspect. Users can enter a Git repository URL or click the "upload zip file" link to upload zipped source code.

### Inspect a repository



Or click [here](#) to upload zip file

### Insphpect: Smarter code reviews

Insphpect is an automated code review tool which identifies inflexibilities in PHP code and helps you write better software.

#### What does Insphpect do?

Insphpect analyses your code for design patterns which are known bad practices and introduce negative traits such as tight coupling and global state.

#### How is it different from Scrutinizer/phpmd/etc?

Insphpect looks for software traits that lead to poor flexibility. The aforementioned tools mostly use size based metrics like cyclomatic complexity (the number of if statements, loops and functions) to grade your classes.

For example, in Scrutinizer A class with 34 complexity gets an A and a class with 35 complexity gets a B. This 35 cut-off is arbitrary and only indicative of problems in your code.

Although it's a *bit more complex than that*, such tools grade your code on simple tallies of the weight of your code (number of methods, cyclomatic complexity number of coupled classes, etc).

Instead of using statistical metrics which focus on code weight, insphpect looks for specific design patterns which are known to made code difficult to maintain and extend. For example, singletons and global variables. It uses these, rather than the simply weighing the size of your classes to grade your code.

#### Who is Insphpect for?

Insphpect is useful for both novice and experienced developers who write object-oriented PHP code. It can be used as an educational tool and as a way of improving your existing software.

When Insphpect encounters a known antipattern, it highlights and tells you why it's a poor design choice as well as making recommendations on what you should do instead. In some cases it can even generate a patch to remove the antipattern\*.

#### What does Insphpect not do?

Insphpect focuses only on code *flexibility*. It does not attempt to identify bugs or bad practices related to other desirable software traits such as performance, user experience, or security.

It does not currently perform continuous integration and only offers on-demand scans of your code.

\* This feature should be considered alpha

### Sample analysis

Below are some screenshots and sample reports you can look at to see what Insphpect does.

Not a PHP Programmer? Your feedback matters too, [click here!](#)

### Screenshots

Transphorm/Parser/CsToXPath

93.27

Rating	Issue	Method	Count
100	GlobalStatic variables	ns	12
100	Use of static methods	processor	40

Code

```

<code>
</code>

```

### Sample reports

Choose one of the projects below to view sample reports:

- [Level2/Transphorm](#)
- [symfony/finder](#)
- [symfony/Routing](#)
- [guzzle/guzzle](#)

Please note: These reports are not automatically updated and do not reflect the most recent versions of the code.

**Figure 5.9:** Screenshot of Insphpect home page

Once a user has uploaded a project or provided a Git repository URL, a report is generated as shown in figure 5.10. An overall project score is displayed at the top with each class from the project listed in a table below. Each class is given a score and colour coded for user friendliness.



## Result

### Summary

Report generated for <https://github.com/Level-2/Transphorm> (Commit: ad9c6663b5ca7448afb20135d0a215730249e702)

On 07/03/2020 14:00:36

12 issues in 7 of 58 classes.

12% of classes contain issues.



Class Name		Rating
Transphorm\Property		100
Transphorm\Pseudo\Attribute		100
Transphorm\Hook\ElementData		100
Transphorm\Hook\PropertyHook		100
Transphorm\Hook\PseudoMatcher		100
Transphorm\Hook\PostProcess		100
Transphorm\Property\Bind		100
Transphorm\Property\Content\Pseudo\Headers		100
Transphorm\Property\Content\Pseudo\Attr		100
Transphorm\Property\Content\Pseudo\BeforeAfter		100
Transphorm\Property\Content		100
Transphorm\Property\Content\Replace		100
Transphorm\Property\Repeat		100
Transphorm\Property\Content\Pseudo		100
Transphorm\Property\Display		100
Transphorm\Pseudo\Not		
Transphorm\Rule		
Transphorm\Pseudo\Nth		
Transphorm\TSSFunction\Json		
Transphorm\TSSFunction\Attr		
Transphorm\TSSFunction\Template		

**X**

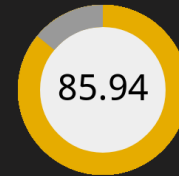
This tool is currently proof-of-concept. Your feedback and evaluation is valuable in helping to improve it and ensure its reports are meaningful.

Please click here to complete a short survey to tell us what you think. It should take less than 5 minutes and help further this research project!

**Figure 5.10:** Screenshot of sample report

Each class can be selected and when a single class is clicked on, the class' score is displayed with the code from the class below. Any lines with flexibility issues are highlighted in red as shown in figure 5.11.

Transphorm\Parser\CssToXPath



### Detected issues

Issue	Method	Line number
Global/Static variables	NA	12
Use of static methods	processAttr	40

### Code

Click highlighted lines for details

```

1 <?php
2 /* @description      Transformation Style Sheets - Revolutionising PHP templating *
3 * @author            Tom Butler tom@r.je *
4 * @copyright         2017 Tom Butler <tom@r.je> | https://r.je/ *
5 * @license           http://www.opensource.org/licenses/bsd-license.php BSD License *
6 * @version           1.2 * */
7 namespace Transphorm\Parser;
8 class CssToXPath {
9     private $specialChars = [Tokenizer::WHITESPACE, Tokenizer::DOT, Tokenizer::GREATER_THAN,
10        '!', Tokenizer::NUM_SIGN, Tokenizer::OPEN_SQUARE_BRACKET, Tokenizer::MULTIPLY];
11     private $translators = [];
12 + private static $instances = [];
13     private $functionSet;
14     private $id;
15
16     public function __construct(\Transphorm\FunctionSet $functionSet, $prefix = '', $id = null) {
17         $this->id = $id;
18         self::$instances[$this->id] = $this;
19         $this->functionSet = $functionSet;
20
21         $this->translators = [
22             Tokenizer::WHITESPACE => function($string) use ($prefix) { return '/' . $prefix . $string; },
23             Tokenizer::MULTIPLY => function () { return '*'; },
24             '!' => function($string) use ($prefix) { return '/' . $prefix . $string; },
25             Tokenizer::GREATER_THAN => function($string) use ($prefix) { return '/' . $prefix . $string; },
26             Tokenizer::NUM_SIGN => function($string) { return '[@id=\' . $string . '\']'; },
27             Tokenizer::DOT => function($string) { return '[contains(concat(\' \', normalize-space(@class), \' \'), \' . $s
28             Tokenizer::OPEN_SQUARE_BRACKET => function($string) { return '[' . $string . ']'; }
29         ];
30     }
31
32     private function createSelector() {
33         $selector = new \stdClass;
34         $selector->type = '';
35         $selector->string = '';
36         return $selector;
37     }
38
39     //XPath only allows registering of static functions... this is a hacky workaround for that
40 + public static function processAttr($attr, $element, $hash) {
41     $attr = unserialize(base64_decode($attr));
42
43     $functionSet = self::$instances[$hash]->functionSet;
44     $functionSet->setElement($element[0]);
45
46     $attributes = [];
47     foreach($element[0]->attributes as $name => $node) {
48         $attributes[$name] = $node->nodeValue;
49     }
50 }

```

**X**

This tool is currently proof-of-concept. Your feedback and evaluation is valuable in helping to improve it and ensure its reports are meaningful.

Please click here to complete a short survey to tell us what you think. It should take less than 5 minutes and help further this research project!

Figure 5.11: Screenshot of sample class

## Code

Click highlighted lines for details

```

1<?php
2/* @description      Transformation Style Sheets - Revolutionising PHP templating   *
3 * @author           Tom Butler tom@r.je                                         *
4 * @copyright        2017 Tom Butler <tom@r.je> | https://r.je/                   *
5 * @license          http://www.opensource.org/licenses/bsd-license.php   BSD License *
6 * @version          1.2                                                       */
7namespace Transphorm\Parser;
8class CssToXPath {
9    private $specialChars = [Tokenizer::WHITESPACE, Tokenizer::DOT, Tokenizer::GREATER_THAN,
10        '*', Tokenizer::NUM_SIGN, Tokenizer::OPEN_SQUARE_BRACKET, Tokenizer::MULTIPLY];
11    private $translators = [];
12    - private static $instances = [];

```

## Why this impedes flexibility

## Global variables

Note: A future update will differentiate between `private static` variables and `public static` or `global` variables as `private static` variables do not cause as much of a problem.

## Summary

- Hidden dependencies
- Broken encapsulation
- One component can accidentally overwrite data required by another component (action at a distance)
- You can only every have one copy of the variable
- Adding code requires knowing exactly what variables are already in use
- When working in teams, name clashes can be easily introduced
- Global state makes it difficult to reuse the code. E.g. having two files open at the same time would require writing the code twice, three times for three files, etc.

## Background

The identification of global variables as a bad practice dates as far back at least as far back as 1973[1] and are one of the most widespread and well known *bad practices* related to flexibility. This is likely due to being available in almost every programming language, ease of use and speed to learn. They also cause severe problems in code and it's very easy to get caught out by using them, even in a small application.

Global variables are widely labelled "bad practice" and have been for some time, for example back in 1999 Kernighan wrote:

*"Avoid global variables; wherever possible it is better to pass references to all data through function arguments"*

Kernighan[2]

And Hevery[3] states:

*"I hope that by now most developers agree that global state should be treated like GOTO."*

This attitude is widespread and Sayfan[4] sums up the problem:

*"Whenever shared mutable state is involved, it is easy for components to step on each other's toes."*

Although "global variables are bad" is a common thing to here, for novice developers it's not immediately obvious why this is. However, the reasons have been covered frequently by developers of varying prominence. While writing about designing the Eiffel programming language, [5] stated several problems with global variables:

*"Since global variables are shared by different modules, they make each of these modules more difficult to understand separately, diminishing readability and hence hampering maintenance."*

*"As global variables constitute a form of undercover dependency between modules, they are a major obstacle to software evolution, since they make it harder to modify a module without impacting others."*

*"They are a major source of nasty errors. Through a global variable, an error in a module may propagate to many others. As a result, the manifestation of the error may be quite remote from its cause in the software architecture, making it very hard to trace down errors and correct them. This problem is particularly serious in environments where incorrect array references may pollute other data."*

## Action at a distance

This problem is commonly referred to as *action at a distance* and described by Hevery[6] as:

*"Spooky Action at a Distance is when we run one thing that we believe is isolated (since we did not pass any references in) but unexpected interactions and state changes happen in distant locations of the system which we did not tell the object about. This can only happen via global state."*

## Broken encapsulation

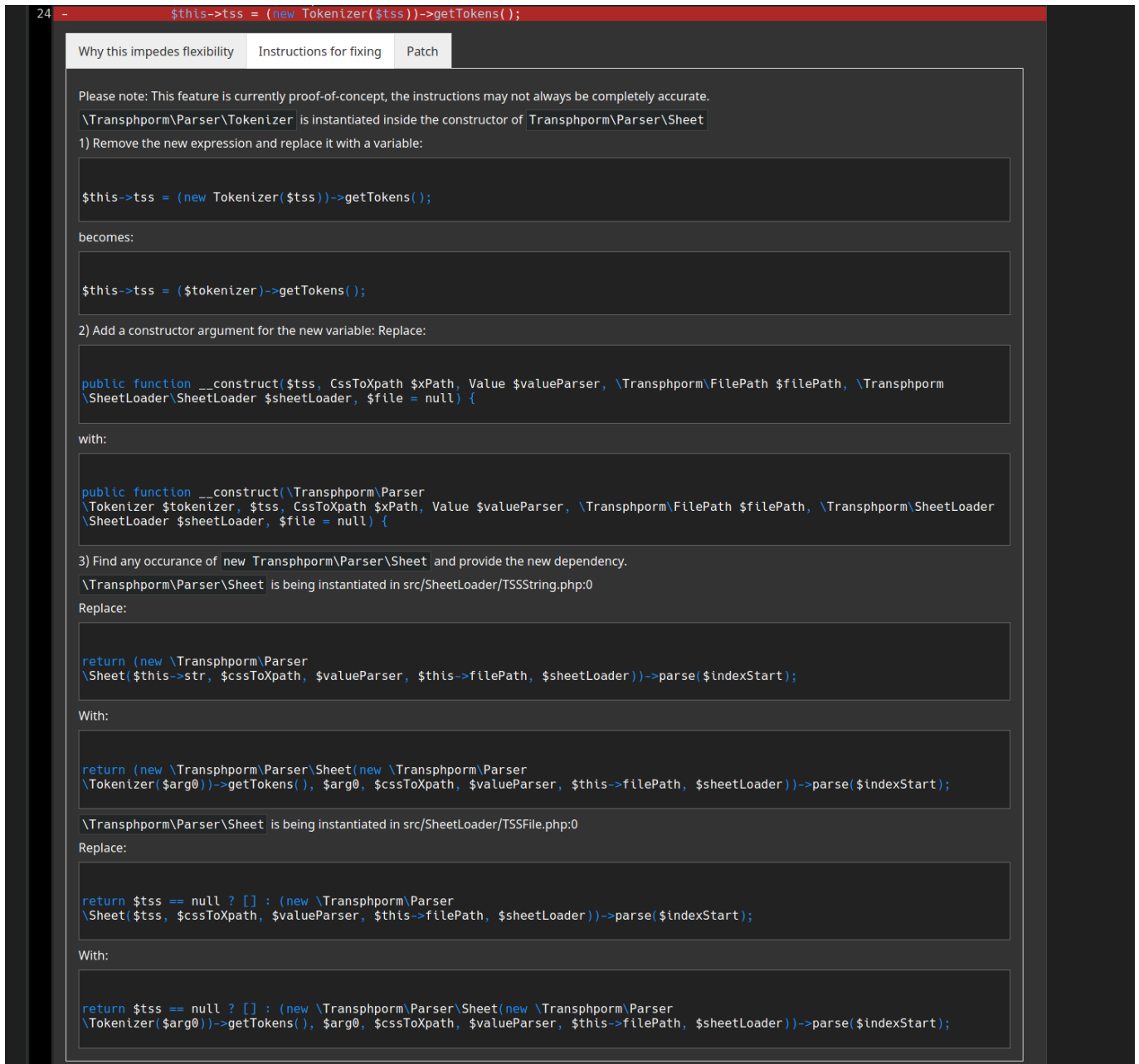


This tool is currently proof-of-concept. Your feedback and evaluation is valuable in helping to improve it and ensure its reports are meaningful.

Please click here to complete a short survey to tell us what you think. It should take less than 5 minutes and help further this research project!

To give users a better understanding of the issue, as shown in figure 5.12, each red line can be clicked on to expand an explanation of the issue detected. The data used to populate this is the data collected in chapter 2.

**Figure 5.12:** Screenshot of sample class with issue expanded



For some bad practices, Inspec generates step by step instructions which are specific to the code supplied (shown in figure 5.13). It is also able to generate a patch which can be applied to the project to remove the bad practice (shown in figure 5.14). Due to the nature of bad practices, this patch can affect files beyond the class which has been clicked on by the user.

**Figure 5.13:** Screenshot of automated fix instructions

Please note: This feature is currently proof-of-concept, this patch may not work, please don't blindly apply it.

```
diff --git a/src/Parser/Sheet.php b/src/Parser/Sheet.php
index 06384a2..3834958 100644
--- a/src/Parser/Sheet.php
+++ b/src/Parser/Sheet.php
@@ -15,13 +15,13 @@ class Sheet {
     private $file;
     private $rules;

-    public function __construct($tss, CssToXPath $XPath, Value $valueParser, \Transphorm\FilePath $filePath, \Transphorm\Sheet
+    public function __construct(\Transphorm\Parser\Tokenizer $tokenizer, $tss, CssToXPath $XPath, Value $valueParser, \Transph
         $this->XPath = $XPath;
         $this->valueParser = $valueParser;
         $this->filePath = $filePath;
         $this->sheetLoader = $sheetLoader;
         $this->file = $file;
         $this->tss = (new Tokenizer($tss))->getTokens();
+        $this->tss = ($tokenizer)->getTokens();
     }

     public function parse($indexStart = 0) {
@@ -115,4 +115,4 @@ class Sheet {
         return $return;
     }
-}
+}
\ No newline at end of file
diff --git a/src/Parser/Tokenizer.php b/src/Parser/Tokenizer.php
index af6d1a3..c0f3f22 100644
--- a/src/Parser/Tokenizer.php
+++ b/src/Parser/Tokenizer.php
@@ -37,16 +37,11 @@ class Tokenizer {
     const MULTIPLY = 'MULTIPLY';
     const DIVIDE = 'DIVIDE';

-    public function __construct($str) {
-        $this->str = new Tokenizer\TokenizedString($str);
+    public function __construct(\Transphorm\Parser\Tokenizer\Comments $comments, $str) {
+        $this->str = $tokenizedString;

         $this->tokenizeRules = [
-            new Tokenizer\Comments,
-            new Tokenizer\BasicChars,
-            new Tokenizer\Literals,
-            new Tokenizer\Strings,
-            new Tokenizer\Brackets
+        ];
+        $comments;
     }

     public function getTokens() {
@@ -62,4 +57,4 @@ class Tokenizer {
         return $tokens;
     }
-}
+}
\ No newline at end of file
diff --git a/src/Parser/Value.php b/src/Parser/Value.php
index 14097e0..e4c8527 100644
--- a/src/Parser/Value.php
+++ b/src/Parser/Value.php
@@ -47,7 +47,14 @@ class Value {

     public function parse($str) {
-        $tokenizer = new Tokenizer($str);
+        $tokenizer = new Tokenizer(new \Transphorm\Parser\Tokenizer\Comments,
+            new Tokenizer\BasicChars,
+            new Tokenizer\Literals,
+            new Tokenizer\Strings,
+            new Tokenizer\Brackets
+        ], new \Transphorm\Parser\Tokenizer\TokenizedString($arg0), $arg0);
+        $tokens = $tokenizer->getTokens();
+        $this->result = $this->parseTokens($tokens, $this->baseData);
+        return $this->result;
diff --git a/src/Pseudo/Not.php b/src/Pseudo/Not.php
index 384d0b2..d9c2f20 100644
--- a/src/Pseudo/Not.php
+++ b/src/Pseudo/Not.php
@@ -24,7 +24,14 @@ class Not implements \Transphorm\Pseudo {
     private function notElement($css, $xpath, $element) {

         foreach ($css as $selector) {
-            $tokenizer = new \Transphorm\Parser\Tokenizer($selector);
+            $tokenizer = new \Transphorm\Parser\Tokenizer(new \Transphorm\Parser\Tokenizer\Comments,
+                new Tokenizer\BasicChars,
+                new Tokenizer\Literals,
+                new Tokenizer\Strings.
```

Figure 5.14: Screenshot of generated patch

## 5.4.1 Technical challenges

The majority of the bad practices use static analysis. For example, looking for object instantiation in a constructor is fairly trivial:

1. For each class, find the constructor.
2. Look for a new keyword.

However, for service locators and setter injection, the process was significantly more complex. The code is re-written on the fly to log all calls to a method.

For example, the code shown in figure 5.14 is rewritten on the fly, to this prior to being executed to become the code shown in figure 5.15.

```
class Car {
    private $engine;
public function __construct(Engine $engine) {
    $this->engine = $engine;
}
public function drive() {
    $drive = $this->engine->drive();
    return $drive;
}
}
```

**Figure 5.14:** Code prior to being rewritten

```
class Car {
    private $engine;
public function __construct(Engine $engine) {
    Insphpect::logCall(__CLASS__, '__construct');
    $this->engine = $engine;
}
public function drive() {
    Insphpect::logCall(__CLASS__, 'drive');
}
```

```
        $drive = $this->engine->drive();
        return Insphpect::logReturn(__CLASS__, __METHOD__, $drive);
    }
}
```

**Figure 5.15:** Code after being rewritten by Insphpect

This was done for every class and mock dependencies were created. In this example a mock version of the Engine class is created on the fly and how its return values are used are logged.

The code is then executed in a separate process for each class so that each class can be run in isolation with its own set of mock dependencies created.

Once the trace is complete it is then possible to see:

- If a dependency is used in more than one method
- If an object is a service locator (if it is used to just return another object)

Re-writing the code on the fly was particularly challenging because there are so many variations of code. For example, for logging return values it's not always simple due to complex expressions such as the return statement shown in figure 5.16.

```
return $check ? 1 : -1;
```

**Figure 5.16:** Complex return statement

Which is rewritten to the code shown in figure 5.17:

```
return Insphpect::logReturn(__CLASS__, __METHOD__, $check ? 1 : -1);
```

**Figure 5.17:** Complex return statement after being rewritten by Insphpect

The first attempt to handle this wrapped everything from the return statement to the following semicolon. However, there were still edge cases where this is not the case. For example a closure

as shown in figure 5.18.

```
return function($a, $b) {  
    return $a+b;  
};
```

**Figure 5.18:** Alternative complex return statement

Insphect needs to rewrite this code to become as it is shown in figure 5.19.

```
return Insphect::logReturn(__CLASS__, __METHOD__, function($a, $b) {  
    return $a+b;  
});
```

**Figure 5.19:** Alternative complex return statement after being rewritten on the fly

Rewriting code on the fly had to be done carefully and required a significant amount of testing on various real projects before the error rate was low.

The top 20 PHP projects from Packagist were scanned during testing and none of them now causes errors when rewriting the code, however there are likely still edge cases where re-writing the code on the fly will not produce valid code.

## 5.4.2 Known limitations

The software is feature-complete however there is still room for improvement. The tool currently lacks the following features:

- Privacy. Anyone can see any code uploaded to the website if they know (or guess) the URL.
- Automated fixes/Patch generation. This is partially implemented for two of the bad practices, however it is not perfect and may result in invalid code. There is a notice on the site telling users that the feature is experimental.



- The tool cannot scan itself or any library it uses in the back-end. Because the tool runs modified versions the code it has to load the classes into memory to run them. PHP does not allow loading two classes with the same name so the tool cannot currently scan itself.

Future iterations could fix these issues and improve the tool, however it is in a state where it is useful to general programmers so will be released with these limitations.

# 6. Evaluation

## 6.1 Introduction

In chapters 3 and 4, a metric, and tool which enables grading source code based on that metric, were developed. This chapter outlines the techniques used to evaluate the metric and software and presents the results of the evaluation.

## 6.2 Evaluation techniques used by other software metrics

During the literature review, the following existing software metrics were analysed. To evaluate the metric that was developed for this research, the evaluation techniques used when these metrics were created were looked at as inspiration for evaluation techniques.

**MOOD** (Abreu *et al*, 1995) introduces the metric but does not evaluate it. A later paper *An Evaluation of the MOOD Set of Object-Oriented Software Metrics* (Harrison *et al*, 1998) by different authors, provides a more robust evaluation of the metrics. This paper uses nine commercial pieces of software to evaluate the metrics and compare them to other metrics such as total classes, total methods and total attributes. The paper ends with the sentence " Without further empirical validation, we cannot be sure that it is worth paying attention to these metrics." implying their evaluation technique is inconclusive. They also note that "their utility will continue to be questioned until a sufficient number of empirical validations have been performed at a systems level to establish causal relationships between the metrics and external quality attributes of systems, such as reliability,maintainability, testability, etc"

**MOOSE** (Chidamber *et al*, 2007) does not provide any kind of external validation or evaluation. The paper outlines the metric and compares it to criteria supplied by Weyuker *et al* (1988). However, (Chidamber *et al*, 2007) cites Cherniavsky *et al* (1991) which concludes "It is shown that a collection of nine properties suggested by E.J. Weyuker is inadequate for determining the quality of a software complexity measure". The MOOSE metrics are evaluated against these nine properties but here is no other external evaluation and no evaluation on the suitability of the proposed metrics or comparison to alternatives.

**QMOOD** (Bansiya *et al*, 1997). When this metric was presented, it was not externally validated at all. As these metrics are purely statistical (e.g. number of classes) there is no scope for evaluation in the context of disagreeing or agreeing with the results. The metrics are presented as-is without any external validation of their usefulness. A later paper (Bansiya *et al*, 2002) does provide some model validation. This validation only "Verifies quality attribute values are in valid ranges". There is no external evaluation, survey or opinions on the suitability of the metric by anyone outside the authors of the metric.

None of these metrics were externally validated or evaluated at the point they were introduced,

they are evaluated on their own merit with their own criteria.

An important distinction in metrics is noted by *An Evaluation of the MOOD Set of Object-Oriented Software Metrics* (Harrison et al, 1998):

*We also need to distinguish between internal attributes of a product or process (those attributes which can be measured purely in terms of the product itself), and external attributes of a product or process (those attributes which can only be measured with respect to how the product or process relates to entities in its environment [1]). Managers are often particularly interested in measuring external attributes such as reliability and maintainability. However, OO software metrics are often based on internal (low-level) attributes, under the assumption that they are related to external (high-level) attributes*

---

By Harrison's definition, flexibility is an *external attribute* while the metrics such as QMOOD above are measuring *internal attributes*. Evaluation techniques used for these *internal attributes* are may not be useful when applied to *external attributes* such as flexibility.

### 6.2.1 Other academic approaches

*Towards a Framework for Software Measurement Validation* (Kitcchenham et al, 1995) provides an overview of how to validate software measurements and notes that software metrics should exhibit the following properties:

1. Be based on an explicitly defined model of the relationship between certain attributes (the relationship between the internal and external properties)
2. be based on a dimensionally consistent model.
3. exhibit no unexpected discontinuities.
4. use units and scale types correctly.

The metric produced as part of this research can be tested against each of these criteria:

*Be based on an explicitly defined model of the relationship between certain attributes*

---

The created metric links the internal attributes (bad practices) to the external attribute (flexibility). This is done explicitly using the severity rating of each bad practice. As such, the metric meets this criteria.

*be based on a dimensionally consistent model.*

---

This is elaborated as:

*For example, a cost model often uses size to predict effort. A feature of this type of model is that the unit of the output variable is different from the unit of the input variable(s). Thus, the model must include constants with appropriate units to convert between the different units.*

---

The produced metric, uses such constants in the form of severity ratings for each bad practice to convert the input variables (size of the project, number of bad practices) into the output. The metric fulfills this criteria.

*exhibit no unexpected discontinuities.*

*Valid indirect measures should not exhibit unexpected discontinuities; that is, they should be defined in all reasonable or expected situations.*

---

The example given is where an input is zero where the metric generally expects a positive number. The produced metric implicitly handles these cases as outputs can either be 100 or 0 depending on whether bad practices were detected or not.

There is no way for the metric to produce a number outside of the range 0-100, therefore this criteria is met.

*use units and scale types correctly.*

---

Kitcchenham *et al* (1995) defines units as measurable attributes and scale types as nominal, ordinal, interval and ratios. An example given is categories such as "Major, Minor and Negligible". How units are applied to these scales must be consistent.

*Thus, in the context of nominal and ordinal scale measures where our measures are mappings to arbitrary labels, we suggest a "unit" is needed to ensure that such measures are used consistently.*

---

The metric created maps bad practices / project size (a unit) to the scale output. The output on a scale of 0-100 is an arbitrary label but it is based on a unit and the mapping is consistent. Therefore, this criteria has been met.

The metric produced meets all of Kitcchenham *et al* (1995) criteria. Kitcchenham *et al* (1995) also notes that to validate the mappings and scales that experiments such as "asking a random section

of individuals" is a meaningful method of validating the measurements.

*To corroborate a measure, we can perform experiments to see whether people agree that an attribute exists or whether a mapping to a value captures their understanding of the attribute. For example, if we want to know whether an entity exhibits a particular attribute we can ask a random selection of individuals to classify a set of entities according to a set of possible categories (for nominal scale measures) or to rank the set of entities with respect to the attribute of interest.*

---

To validate the scale produced, a selection of individuals can be asked whether their view of flexibility matches the output of the metric.

## 6.3 Evaluation techniques for this project.

The following evaluation strategies were considered:

1. Comparing the results to other similar metrics (Like Harrison *et al* (1998))
2. Asking real developers to evaluate the flexibility of some software and comparing it to the metric's results.
3. Having real users try the software and answer a questionnaire to see if they agreed with the score and recommendations given (As suggested by Kitchenham *et al* (1995)).

Strategies (1) and (3) were chosen and (2) was discounted.

## 6.4 Real developer evaluations

The software could have been evaluated by getting real developers to give their view of flexibility of some existing software and grade it, then compare their grade to the results generated by the metric.

This could potentially yield answers to questions such as "Is X more flexible than Y?" from both developers and the metric that was developed.

This was discounted for the following reasons:

1. If using existing software, for a developer to accurately grade the flexibility of some code, it would be significantly easier for programmers already familiar with the code they are grading. This would introduce significant selection bias towards authors/users of the code being analysed who may assume the software is flexible because they are used to using it, even potentially used to working around flexibility issues the software has.

2. The question of "what makes code inflexible" was already answered earlier in this research via meta-analysis, as such, developer opinions (100 per bad practice) have already been identified and evaluation with real-users would need to cover this again.
3. For people unfamiliar with the software, it would require a significant time investment to look through code and rate it.
4. Finding experienced developers to spend significant time grading software will be difficult and likely result in a very small sample size as there is nothing in it for them other than participating in the research.

Instead, blind trials could be performed on some sample projects created specifically for this purpose. This would have introduced its own set of issues:

1. It would be very difficult to create sample projects that are representative of real world projects. Real world projects can take months or years worth of development and different projects by different authors have different programming styles which would be hard to replicate. Creating realistic sample projects is not feasible.
2. Recruitment of respondents would be difficult as it would require a large time investment to look through the code.
3. This would likely result in a tiny sample size, from which it would be difficult to infer any meaningful conclusions.

As such, although this strategy may have the potential to be the most robust form of evaluation, the limitations mentioned previously made it unfeasible and this evaluation strategy was not used. As the less than 5 minute questionnaire eventually used only yielded 75 responses, the sample size of people for this much more time consuming, and higher skill requirement, evaluation technique would have been very small.

## **6.5 Compared to other metrics**

Insphect was used to grade 20 pieces of existing software. The same 20 pieces of software were graded using other metrics for comparison.

This method was chosen as it's relatively easy and may yield interesting correlations. However, it is

noted that any correlation or lack of correlation may not give any meaningful insight as the metrics will be grading the software in different ways. It may be that a good grade in one metric always gives a good grade in another, if so, that would be a useful point for further research.

During the literature review 72 different static analysis tools were briefly reviewed. Most of these were deemed irrelevant to the research as they did not analyse code for the same kind of issues and most did not produce a grade.

Three metric based tools were found to grade software quality and produce an overall project grade. However, between the literature review and evaluation Symphony Insights is now commercial software and cannot be used without a paid license. As such, it was omitted from the evaluation.

The following tools which provide an overall grade were used for evaluation:

- Insphect (this metric)
- Scrutinizer-CI
- SonarQube (Specifically SonarCloud)

Scrutinizer uses threshold based metrics and was reviewed in detail during the literature review. SonarQube provides similar threshold based metrics but also has a "Maintainability" grade, which is described as "SonarQube helps you find your Code Smells and understand what's wrong. Then it shows you how to fix the problem." (SonarQube, n.d.) which is very similar to Insphect.

### **6.5.1 Methodology**

Insphect was already used to analyse the top 20 packages from packagist. These grades were compared to SonarQube's and Scrutinizer-CI's scores for the same packages. The results were plotted to look for correlation.

A sample size of 20 was used because Scrutinizer took 15-20 minutes for small projects and 20-25 minutes for larger projects. Gathering results for 20 projects already took a significant amount of time.

This selection method was chosen as the top 20 packages are popular, of varying sizes and by different authors. This acts as a randomisation technique that is biased towards the most popular packages.

Each of the packages was then scanned with the three different tools to produce a grade.

Correlation between the grades, or lack thereof, may not be meaningful because the software tools are all looking for different aspects of the software. However, if the tools all roughly agree it



would be a starting point for further research to understand why they agree and why there may be correlation is between bad practices and aspects like code size.

Scrutinizer and Inpspct provide an overall project score to grade the quality of the software.

SonarQube provides four different metrics:

- Reliability, detects bugs
- Security, detects security issues
- Maintainability, detects code maintenance issues
- Duplications, detects duplicated code

The only metric which can be reasonably compared to Inpspct is the *Maintainability* score. As such, the others were omitted from the results.

Number of classes was also included in the results as a project's score may correlate with project size. Tiny projects may be less likely to include issues in any tool due purely to probability.

## 6.5.2 Results

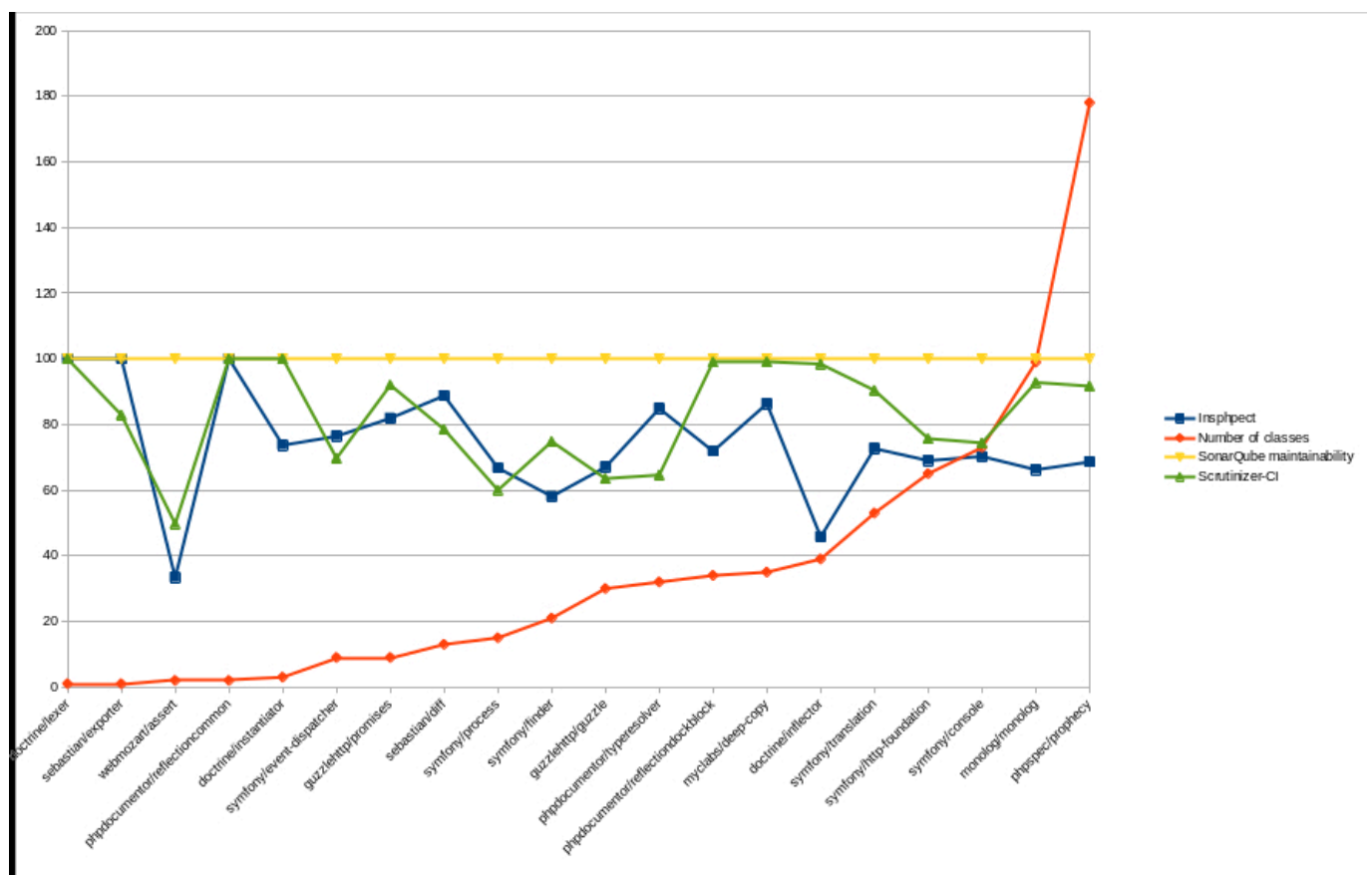


Figure 6.1: Results of each tool, Normalised

Figure 6.1 shows normalised results for each tool and the size of the project.

SonarQube graded all 20 projects as A while Scrutinizer and Insphect showed variation in their scores. This may be due to limited granularity using an A-F scale however, neither Scrutinizer or Insphect correlate with SonarQube. Scrutinizer and Insphect each gave three perfect scores. A larger sample size may yield a wider variation from SonarQube but both Scrutinizer and Insphect grade one or more project in the 40-50% band which should be visible in an A-F scale.

It could be hypothesised that popular packages are more likely to be well written and get higher scores. To test this, a further 20 projects were uploaded to SonarQube:

- 20 projects were chosen at random from packagist using the "random packages" page.
- The packages were scanned 6 months after the initial tests in case the A grades given previously were a temporary bug in the software at the time the results were gathered.

The 20 projects analysed by SonarQube were:

- def-studio/dock
- vyuldashev/monolog-loki
- signifly/laravel-domain-commands
- takeit/amp
- vietanh/lform
- snowcookie/generate-schema
- singcl/php-mvc
- wanghouting/lt-dev-tools
- 6phere/php-websocket
- hschottm/contao-textwizard
- memio/spec-gen
- zzzcms/mysql-dispatch
- armonia-tech/phalcon-migration
- yuk1/job
- joshuand1990/excel-helper
- dakalab/birthday
- b61/laravel-reportable
- contaoblackforest/contao-calendar-tags-bundle
- caaqil/instagram-api
- m2demo/module-m2-extension

All 20 packages were also given an A grade for maintainability. It is not clear why this grade is A in all 40 projects uploaded as no other score has been recorded. Regardless of the reason, the outcome for this research is the same: The score from SonarQube is not relevant for the research.

In contrast, both Insphect and Scrutinizer gave three perfect scores of 20 projects analysed.

Two of the three perfect scores given by Insphect were on projects with only one class where any issues related to coupling would be difficult to introduce, therefore several of the checks are irrelevant.

Scrutinizer gave two of its three perfect scores to different projects to Insphect so they disagree on what gets a perfect score more than they agree.

This can be explained due to Scrutinizer's threshold. Scrutinizer gives poor grades to large classes, as such any project with few classes but many lines will be graded poorly by Scrutinizer. On the other hand, as Insphect looks for bad practices, the size of the class is not taken into account.

### 6.5.3 Conclusions

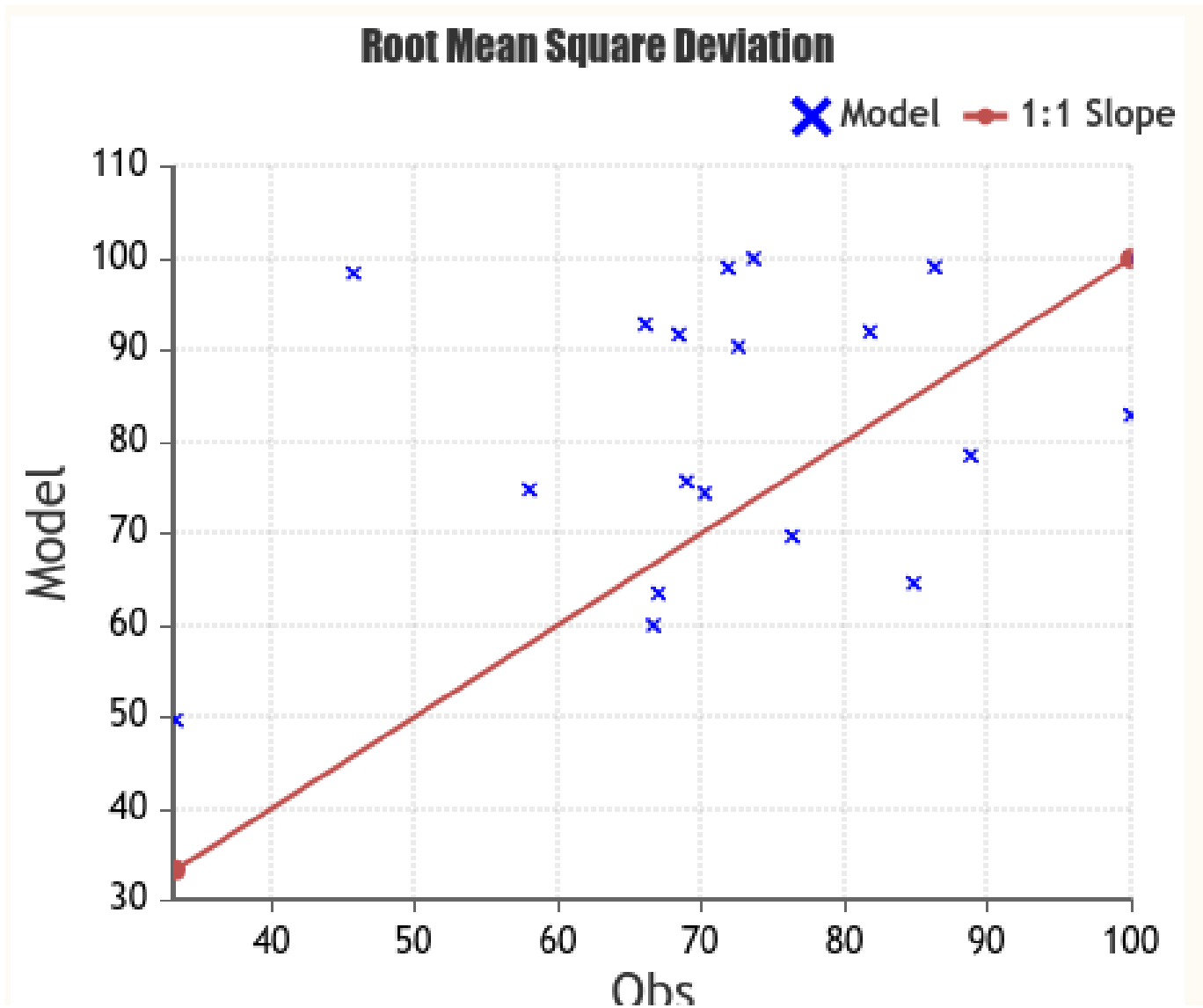
None of the three result sets correlate with project size, at least when the project size is  $> 2$  classes. Due to the way these tools work, small projects are less likely to contain issues and therefore are more likely to get higher scores.

SonarQube gives the least meaningful results on this dataset with every project getting an A grade. This makes comparisons difficult and usefulness to users limited. As the other two tools show significant variance it is unclear if this is a bug in SonarQube at the time of

It is unclear if there is a correlation between Scrutinizer and Insphect. There appears to be a small correlation, but there are several libraries in which Scrutinizer and Insphect strongly disagree with each other. For example, Scrutinizer gives doctrine/inflector a 9.84 (out of 10) while Insphect grades it 45.77 (out of 100).

The Normalised Root Mean Square Deviation (NRMSD) is 19.42. Using NRMSD, the closer the result is to zero, the higher predictability the model being tested has. For this comparison, a score of zero would mean knowing an Insphect score, you could always accurately predict the corresponding Scrutinizer score.

At 19.424 this may indicate a minor correlation. Though the average difference between the scores is 19.42.



**Figure 6.2:** Normalised Root Mean Square Deviation of Insphect and Scrutinizer-CI

As shown in figure 6.2, plotting the RSMD may show a minor correlation but demonstrates the significant outliers. A much larger sample size would be required to draw any meaningful conclusions, however there is no conclusive correlation in the results collected.

## 6.6 Bad practice frequencies

After gathering results in section 6.5, the results from all 20 projects were analysed together to identify which bad practices were detected most often.

Table 6.1 shows each tool analysed and the number of times each bad practice was identified.

**Table 6.1:** Frequency of bad practices

	new in constructor	annotations for configuration	global variables	service locator	static methods	inheritance	setter injection	singleton
doctrine/lexer	0	0	0	0	0	0	0	0
sebastian/exporter	0	0	0	0	0	0	0	0
webmozart/assert	0	6	0	0	99	1	0	1
phpdocumentator/reflection-common	0	0	0	0	0	0	0	0
doctrine/instantiator	0	8	2	0	6	0	0	0
symfony/event-dispatcher	1	0	3	0	4	1	0	0
guzzlehttp/promises	1	6	1	0	27	3	0	0
sebastian/diff	1	2	1	0	1	3	0	0
symfony/process	2	0	5	0	5	8	0	1
symfony/finder	1	0	2	0	12	16	0	0
guzzlehttp/guzzle	4	0	8	1	90	9	0	0
phpdocumentor/type-resolver	2	16	0	0	0	7	0	0
phpdocumentor/reflection-docblock	3	0	25	0	32	0	1	1
myclabs/deep-copy	6	0	0	0	2	7	1	0
doctrine/inflector	0	0	0	0	26	4	0	1
symfony/translation	4	0	8	0	7	33	0	0
symfony/http-foundation	6	1	21	0	47	33	2	2
symfony/console	23	6	15	0	40	35	0	0
monolog/monolog	8	10	15	0	31	56	0	0
phpspec/prophecy	35	0	4		26	103	0	1
<b>Total</b>	<b>97</b>	<b>55</b>	<b>110</b>	<b>1</b>	<b>455</b>	<b>319</b>	<b>4</b>	<b>7</b>

The most commonly identified bad practice identified was **static methods** with **inheritance** in second place.

It is observed, and expected, that different projects follow different programming styles and contain different practices due to the authors chosen approaches.

These numbers alone do not give meaningful data as different projects are different sizes. Table 6.2 shows the same data weighted by class. Each cell is the number of occurrences divided by the number of classes in the project, an average number occurrences per class within the project.

Class level practices such as inheritance will never score above one as inheritance cannot occur more than once per class.

**Table 6.2:** Frequency of bad practices (average per class)

	new in constructor	annotations for configuration	global variables	service locator	static methods	inheritance	setter injection	singleton
<b>doctrine/lexer</b>	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
<b>sebastian/exporter</b>	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

<b>webmozart/assert</b>	0.00	3.00	0.00	0.00	49.50	0.50	0.00	0.50
<b>phpdocumentator/reflection-common</b>	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
<b>doctrine/instantiator</b>	0.00	2.67	0.67	0.00	2.00	0.00	0.00	0.00
<b>symfony/event-dispatcher</b>	0.10	0.00	0.30	0.00	0.40	0.10	0.00	0.00
<b>guzzlehttp/promises</b>	0.08	0.46	0.08	0.00	2.08	0.23	0.00	0.00
<b>sebastian/diff</b>	0.08	0.15	0.08	0.00	0.08	0.23	0.00	0.00
<b>symfony/process</b>	0.13	0.00	0.33	0.00	0.33	0.53	0.00	0.07
<b>symfony/finder</b>	0.05	0.00	0.09	0.00	0.55	0.73	0.00	0.00
<b>guzzlehttp/guzzle</b>	0.13	0.00	0.25	0.03	2.81	0.28	0.00	0.00
<b>phpdocumentor/type-resolver</b>	0.06	0.50	0.00	0.00	0.00	0.22	0.00	0.00
<b>phpdocumentor/reflection-docblock</b>	0.08	0.00	0.69	0.00	0.89	0.00	0.03	0.03
<b>myclabs/deep-copy</b>	0.17	0.00	0.00	0.00	0.06	0.19	0.03	0.00
<b>doctrine/inflector</b>	0.00	0.00	0.00	0.00	0.67	0.10	0.00	0.03
<b>symfony/translation</b>	0.06	0.00	0.12	0.00	0.10	0.48	0.00	0.00
<b>symfony/http-foundation</b>	0.08	0.01	0.29	0.00	0.64	0.45	0.03	0.03
<b>symfony/console</b>	0.29	0.08	0.19	0.00	0.50	0.44	0.00	0.00
<b>monolog/monolog</b>	0.08	0.10	0.15	0.00	0.31	0.56	0.00	0.00
<b>phpspec/prophecy</b>	0.18	0.00	0.02	0.00	0.14	0.54	0.00	0.01

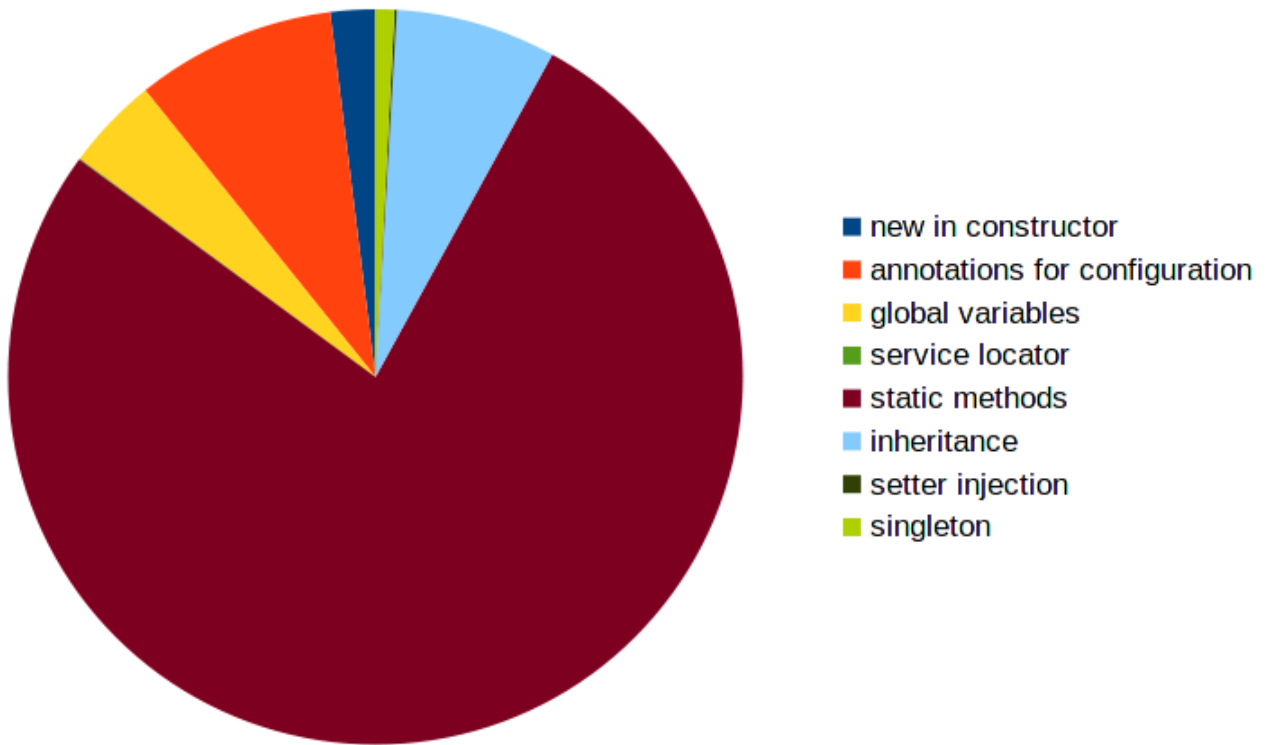
Table 6.2 shows how many times per class on average each bad practice appears in each project.

The most immediately surprising result is that webmozart/assert contains 49.5 static methods on average per class, nearly twenty times as many as the next project, guzzlehttp/guzzle with 2.8. This shows a very different coding style between webmozart/assert and the other projects.

A similar coding style trend can be seen with inheritance. On average across all projects, 27% of classes use inheritance but this is over 40% in Symfony projects. symfony/finder uses inheritance in 73% of its 22 classes.

Overall, static methods are by far the most common bad practice encountered in this data set. This may be expected as static methods can occur more than once per class while inheritance cannot.

Figure 6.3 shows the mean frequency of each bad practice across all twenty projects.



**Figure 6.3:** Mean frequency of bad practices across all projects

The most frequently detected issue is static methods. The second most common issue is Inheritance, followed by a much smaller occurrence of the other practices. Both static methods and inheritance are very easy to implement and static methods can be used by those more familiar with procedural programming to produce a procedural application in classes (Hevery, 2008).

Table 6.3 shows the the grade each project was given and the percentage of marks lost for each practice in each project. Each cell contains the percentage of marks lost due to occurrences of each practice. For example, webmozart/assert lost 8.33% from its grade due to using annotations.

**Table 6.3:** Which bad practice caused grade reductions

	Score	new in constructor	annotations for configuration	global variables	service locator	static methods	inheritance	setter injection	singleton
symfony/finder	58.78	2.27	0	0.31	0	11.36	27.27	0	0
monolog/monolog	66.93	3.63	1.18	3.62	0	3.17	20.36	0	0
phpspec/prophecy	69.04	6.59	0	0.73	0	1.48	18.48	0	0.27
symfony/process	66.77	5.88	0	0.46	0	4.39	18.82	0	2.94
webmozert/assert	55.28	0	8.33	0	0	26.07	10	0	0.25
symfony/translation	74.91	1.77	0	1.91	0	2.98	15.53	0	0
symfony/http-foundation	72.07	3.04	0.02	1.81	0	6.17	15.11	0.98	0.13

symfony/console	70.16	9.56	1.42	0.88	0	2.87	14.59	0	0
guzzlehttp/guzzle	67.97	4.12	0	1.04	3.72	15.75	9.23	0	0
guzzlehttp/promises	62.58	3.13	3.13	0.37	0	23.3	7.5	0	0
sebastian/diff	88.76	3.13	0	0.39	0	0.22	7.5	0	0
phpdocumentor/type-resolver	82.85	2.94	3.03	0	0	0	8.24	0	0
myclabs/deep-copy	86.67	2.56	0	0	0	2.32	8.2	0.24	0
doctrine/inflector	45.77	0	0	0	0	45.94	5.86	0	2.44
symfony/event-dispatcher	78.39	4.17	0	4.09	0	10.02	3.34	0	0
doctrine/lexer	100	0	0	0	0	0	0	0	0
sebastian/exporter	100	0	0	0	0	0	0	0	0
phpdocumentator/reflection-common	100	0	0	0	0	0	0	0	0
doctrine/instantiator	64.78	0	8.89	2.77	0	23.55	0	0	0
phpdocumentor/reflection-docblock	69.6	3.58	0	11.86	0	12.01	0	0.41	0.18
Total		54.1	26	29.94	3.72	180.24	162.76	1.63	6.21

The highest single reduction of marks is doctrine/inflector's 49.94% reduction in grade due to use of static methods. Despite webmozart/assert having a much higher number of static methods, the ratio of static methods to object methods is higher in doctrine/inflector, causing it a higher reduction in grade as a higher proportion of methods throughout the project are static than not.

Overall, static methods caused the highest loss of marks with inheritance in second which is unsurprising as static methods and inheritance are the most commonly detected bad practice in this sample.

## 6.7 User evaluation

Insphect.com was launched in March 2020 along with an anonymous questionnaire offered to users of the site. The questionnaire is available as *appendix VIII*.

To attract a wide spectrum of PHP developers rather than just contacts of the author, the tool was advertised in the following locations:

- Reddit (Butler, 2020)
- Hacker News (also known as ycombinator) (Butler, 2020)
- PHP Today (Butler, 2020)

In addition to the author's website and facebook page which are both followed by a general programming audience.

Initially the results were intended to be collected on the 1st June 2020. However, by this date, despite over 400 projects having been analysed, only 63 people had completed the survey.

This was extended until the 1st August 2020 in an attempt to achieve a larger sample size.

The primary method for generating more responses was a guest post on Sitepoint.com, a popular



resource for web developers. As sitepoint have over 100,000 followers on both facebook and twitter, this was an ideal place to advertise the tool to a wider audience and sitepoint.com accepting the article demonstrates an interest from industry. The guest article, entitled *How to Ensure Flexible, Reusable PHP Code with Insphpect* is available in *appendix IX*.

Despite sitepoint.com posting an article about Insphpect (Butler, 2020) which generated nearly 1,000 hits to the Insphpect website and 25 tweets (Twitter, 2020) about the project, this only resulted in an additional 12 questionnaire responses, bringing the total to 75.

Although a larger sample size would have been preferred, the data collected is still valuable as contains the opinions of real users of varying skill levels about the project.

A study by Macefield *et al* (2009) found that with a sample size of 20, 98.4% of usability issues were detected. In a similar study, Faulkner *et al* (2003) found that with a sample size of 5 users, 55% of usability issues were detected. By increasing the sample size to 20, 95+% of usability issues were detected.

From these studies, it can be inferred that a sample size of 75 should be enough to identify close to 100% of usability issues. However, as usability is not the only factor it cannot be inferred that 75 respondents are enough to answer questions about the suitability of the metric or accuracy of grades given.

Although it is clear that as sample size increases, confidence in the answers also increases. It is not clear what an optimal sample size for a study like this may be.

According to Riley *et al* (2020):

*The sample size of the development dataset must be large enough to develop a prediction model equation that is reliable when applied to new individuals in the target population. What constitutes an adequately large sample size for model development is, however, unclear.*

---

Though Riley is discussing sample sizes in the context of clinical studies, not software evaluation, the concept of a prediction model may be beneficial here. If there is a clear trend in the results then the sample size may be considered accurate. Outliers are always to be expected, but if the majority of respondents agree with each other, this helps validate the results.

For example, if a question with answers *Strongly Disagree - Strongly Agree* had the same or similar number of responses for each answer there would be no predictability. Whereas, if 95% of answers were in the same answer, for example, 95% of respondents selected *Strongly disagree* it would be possible to predict that the next respondent is much more likely to respond *Strongly*

*Disagree* rather than any of the other options.

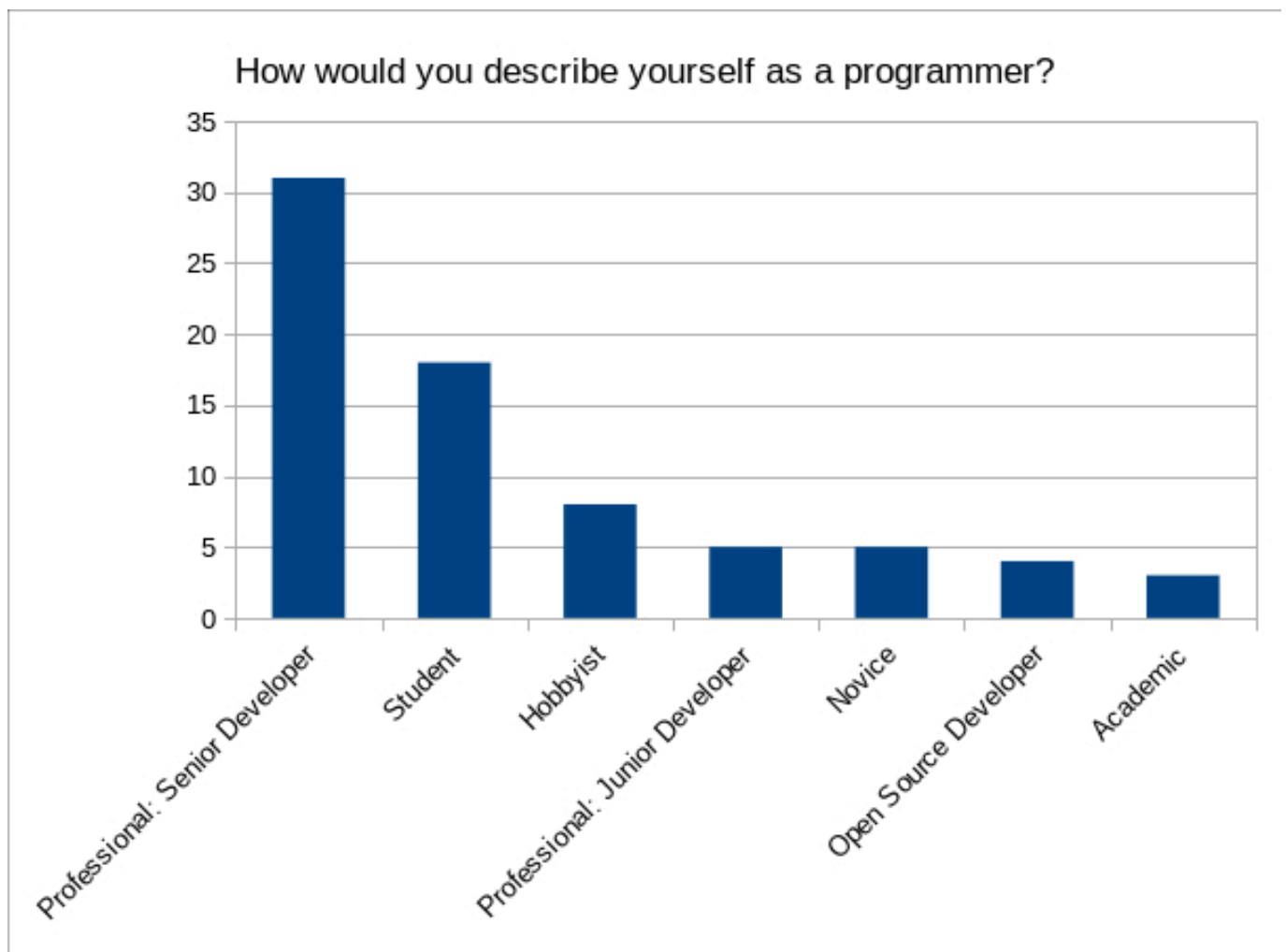
If the 75 respondents produce results with predictability, then 75 can be considered an adequate sample size for this research.

### 6.7.1 Results

The data for users who completed the questionnaire are summarised below. Raw data for the results is available in *appendix VIII*.

All questions were optional and several were multiple choice (multiple choice questions are noted on the respective charts).

Question 1. How would you describe yourself as a programmer?



**Figure 6.4:** Question 1. How would you describe yourself as a programmer?

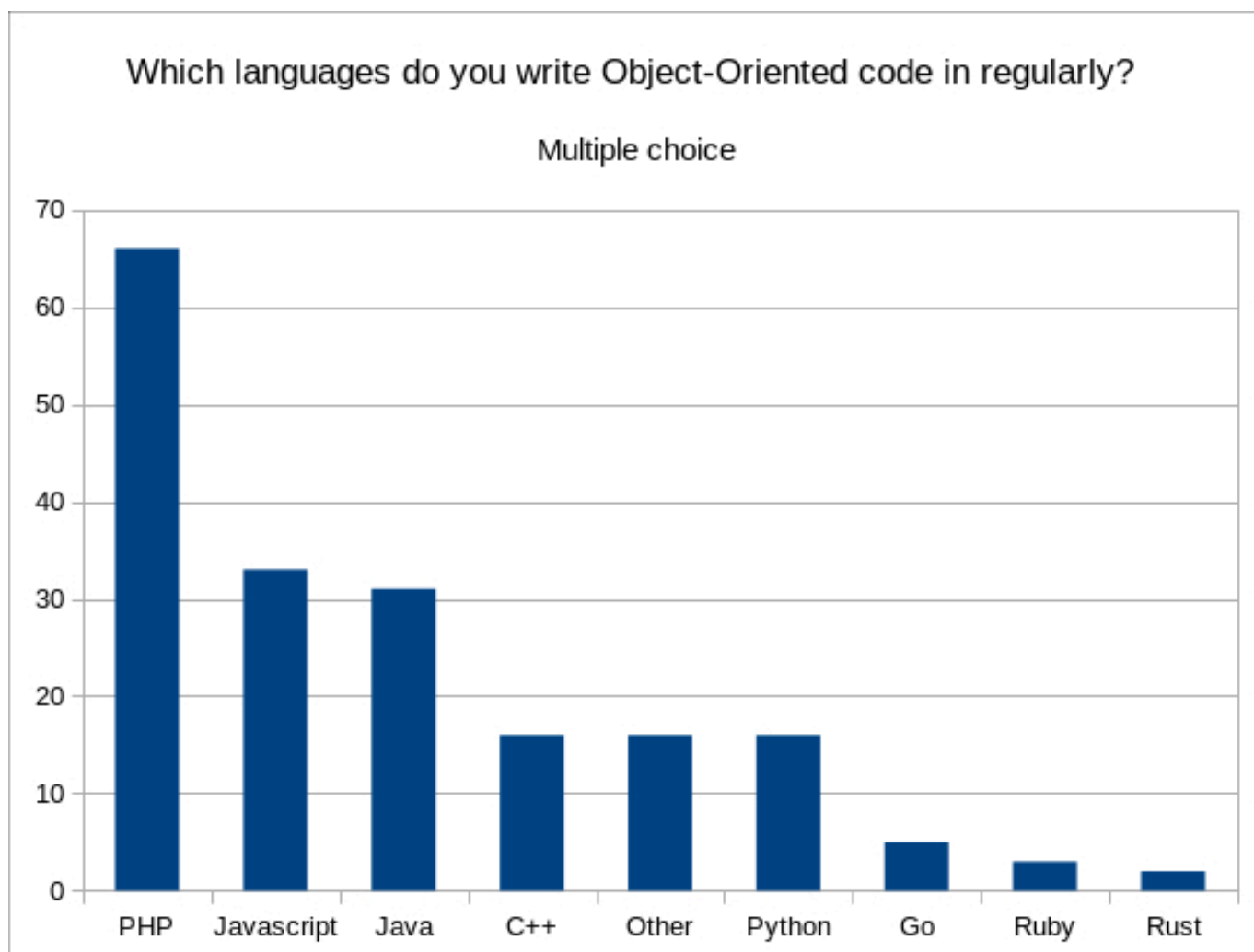
This question was asked so that the data could be broken down by programming ability. Experts should find the tool useful for different reasons than novices. The majority of respondents were

senior developers. Due to the locations the website was advertised, this is unsurprising and senior developers are the most important for judging the usefulness of the results and metric.

Predictability is not a factor for this question as personal information is being asked rather than their opinion on the research.

Most respondents are senior developers. This is beneficial for later questions as senior developer opinions on whether they agree with the grade are more meaningful than that of novices.

Question 2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.



**Figure 6.5:** Question 2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.

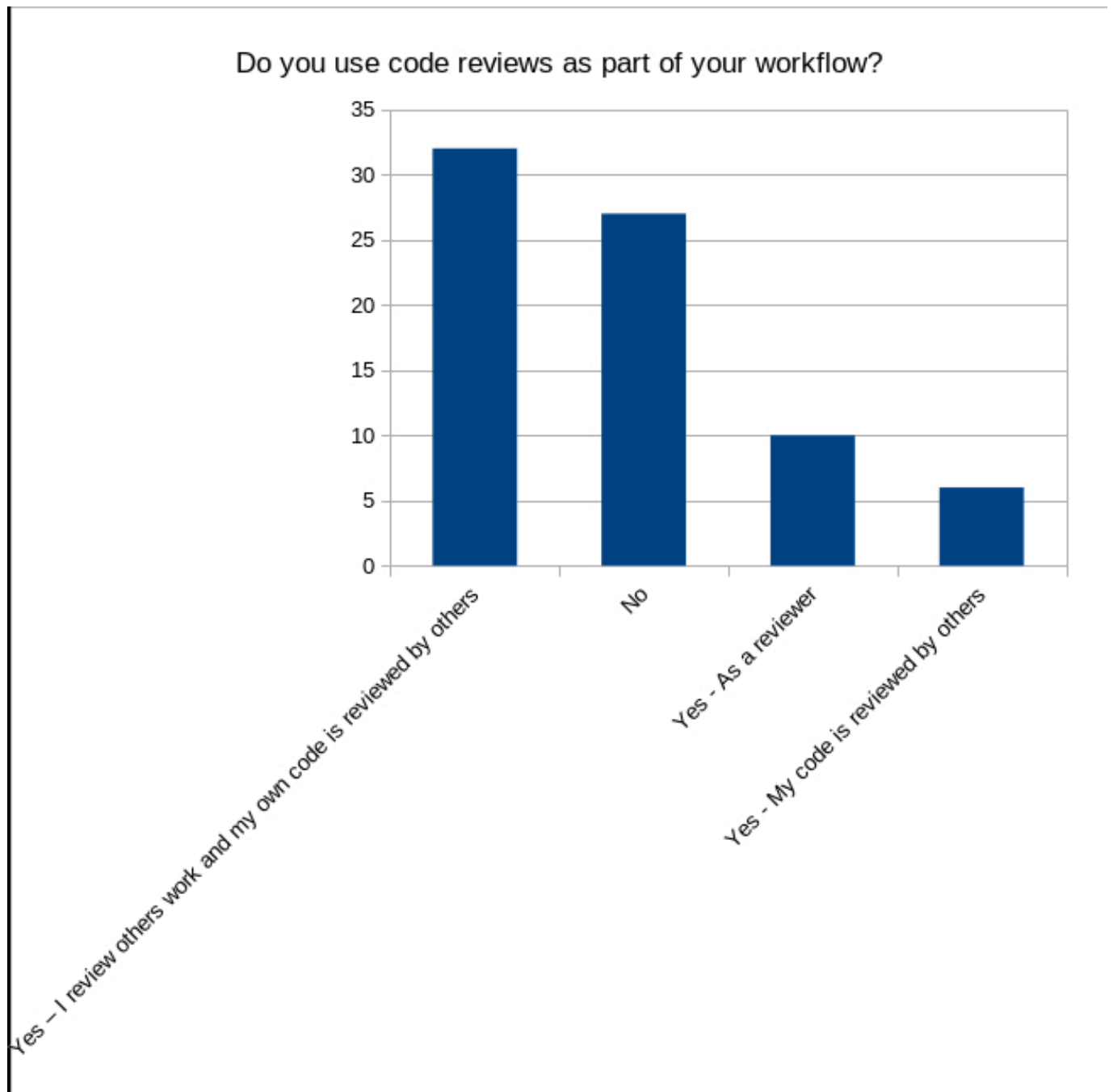
This question was asked so that results can later be broken down by languages used. For example, do Java developers have different opinions on the results than PHP developers?

As the tool being evaluated grades PHP code, and the tool was advertised primarily on PHP

focused programming websites, it's unsurprising that the majority of respondents chose PHP.

Predictability is not a factor for this question as personal information is being asked rather than their opinion on the research.

Question 3. Do you use code reviews as part of your workflow?

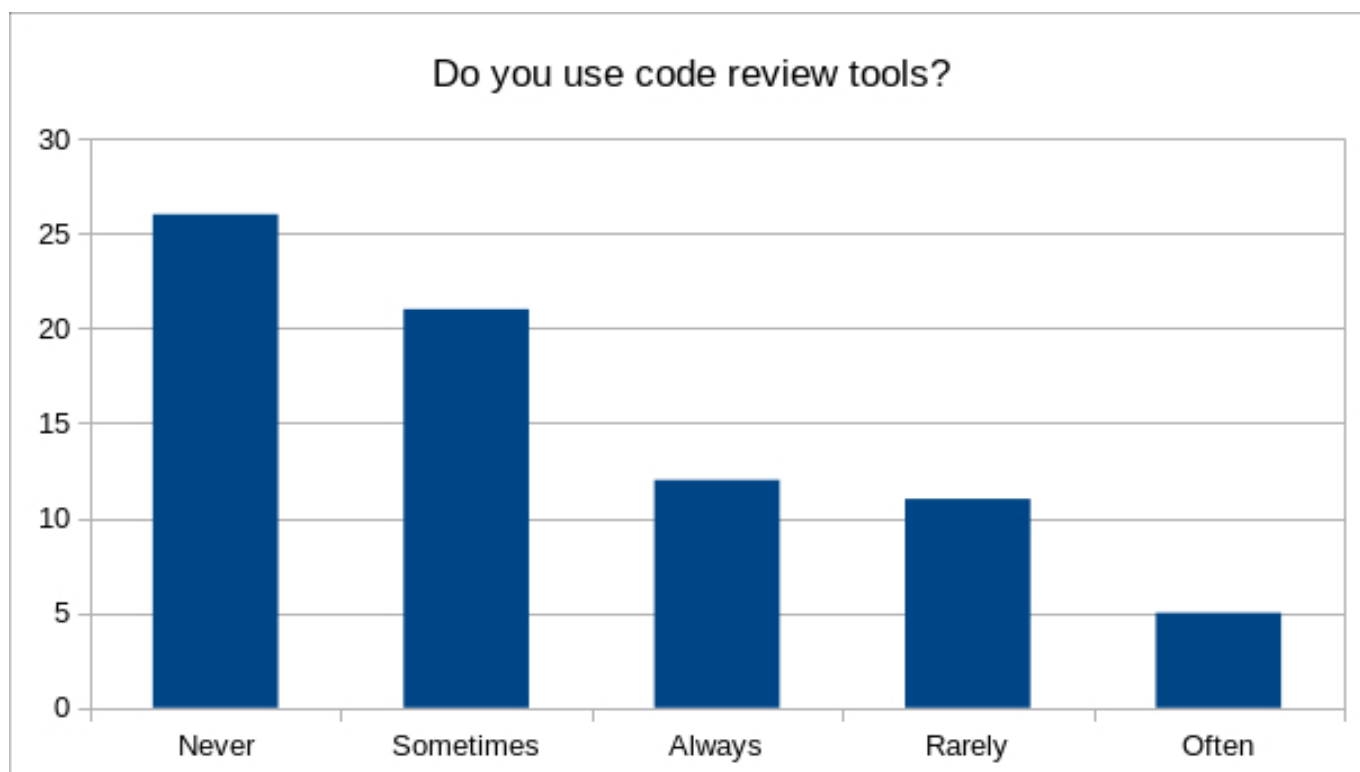


**Figure 6.6:** Question 3. Do you use code reviews as part of your workflow?

This question was asked so that results can later be broken down by those who currently use code reviews and those who do not. As the metric is an automated code review, it is anticipated that those who use code reviews will find the results more useful than those who do not.

Predictability is not a factor for this question as personal information is being asked rather than their opinion on the research.

Question 4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?

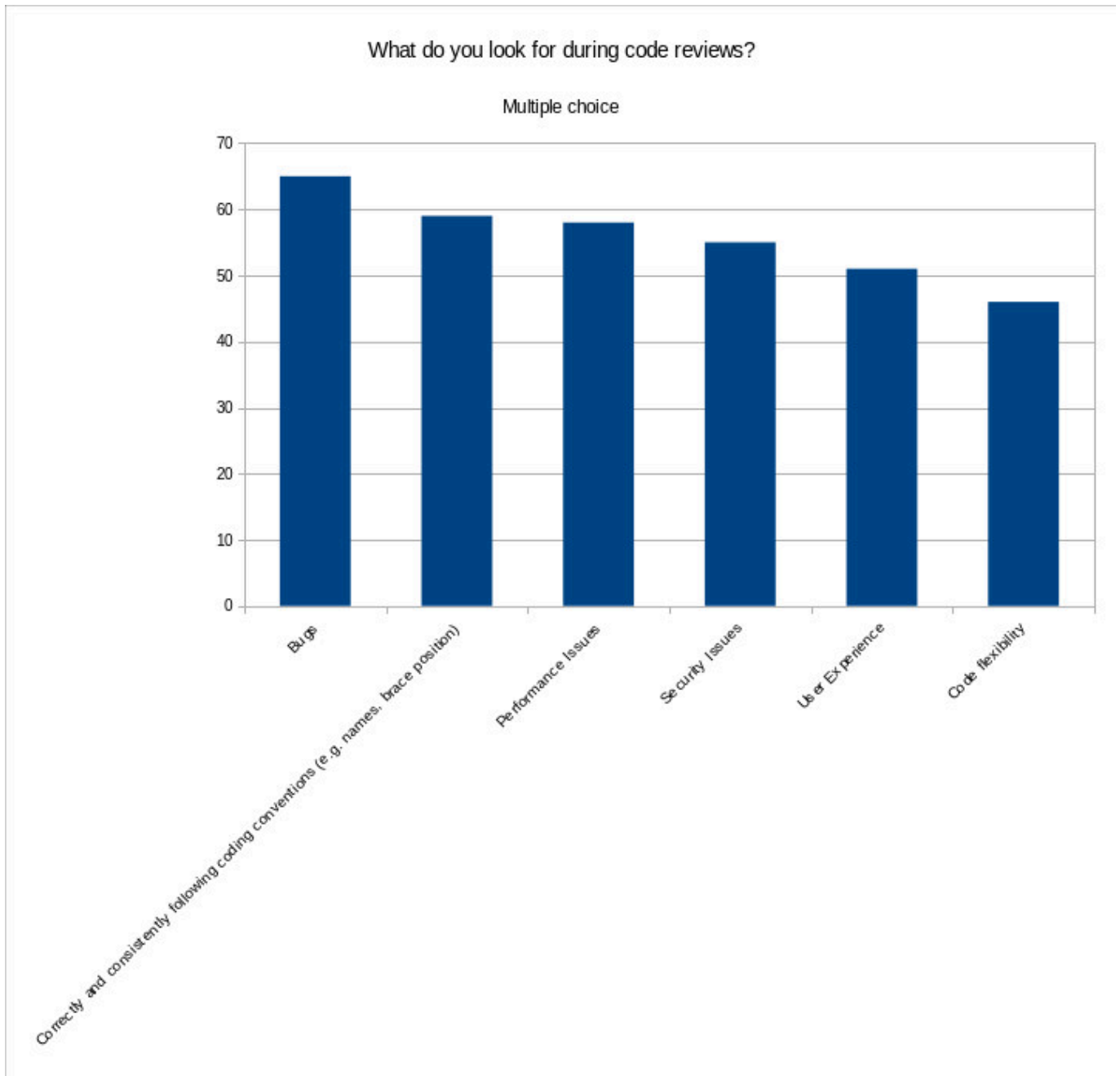


**Figure 6.7:** Question 4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?

This question was asked so that results can be broken down by those who currently use similar tools.

Predictability is not a factor for this question as personal information is being asked rather than their opinion on the research.

Question 5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.

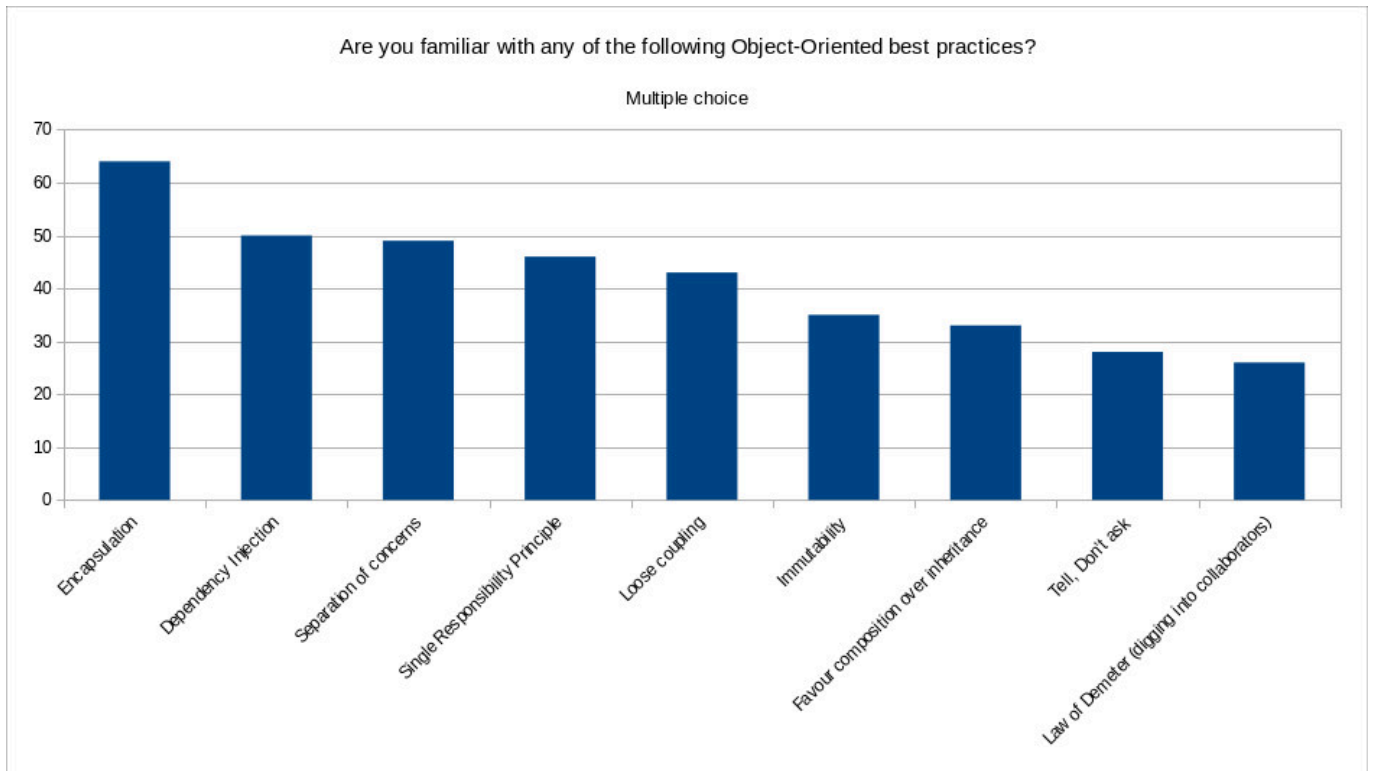


**Figure 6.8:** During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.

This question was asked to see if respondents look for flexibility during code reviews so that the opinions of those who currently look for flexibility can be isolated.

Predictability is not a factor for this question as personal information is being asked rather than their opinion on the research.

Question 6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply



**Figure 6.9:** Question 6. Are you familiar with any of the following Object-Oriented best practices?  
Please tick all that apply.

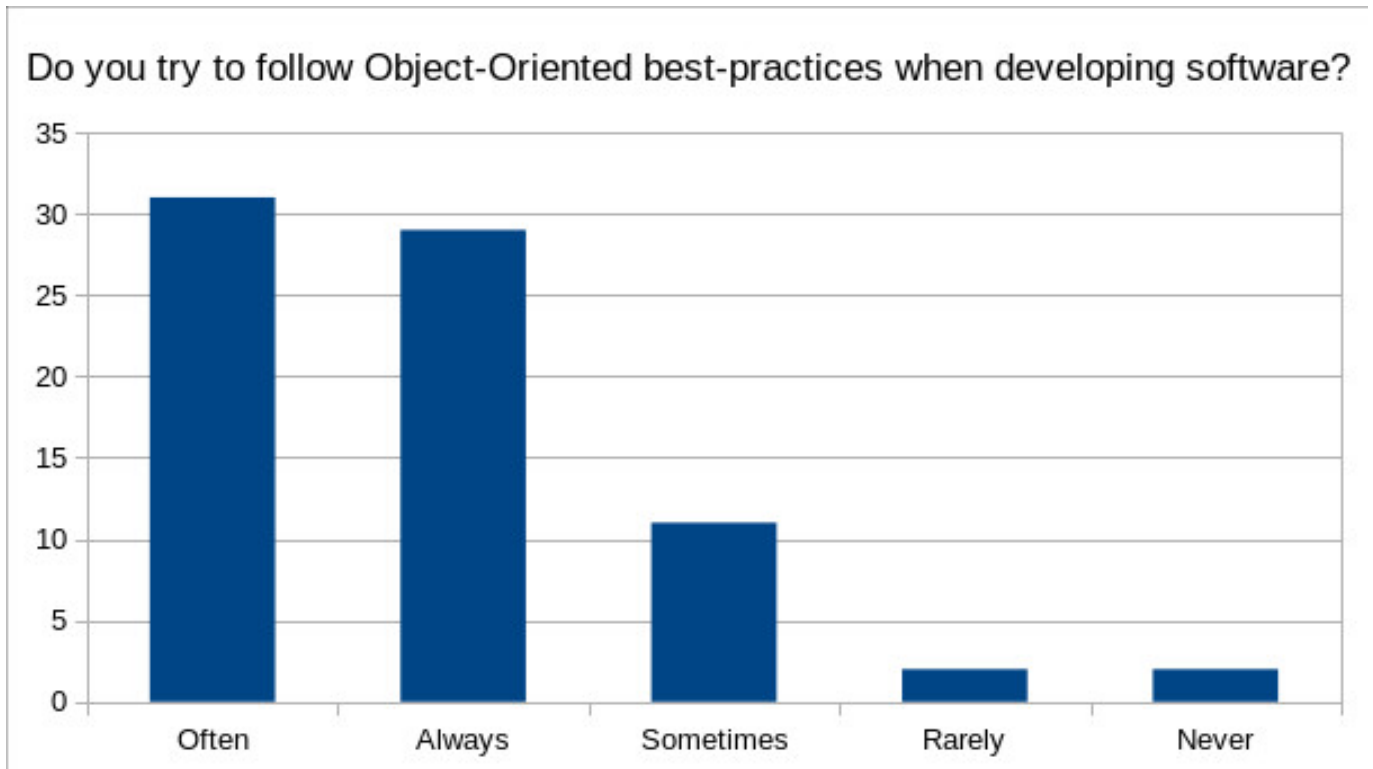
This question was asked to determine which best practices developers are familiar with. This was included so that respondents could be filtered by whether they follow OOP best practices or not.

Most respondents are familiar with Encapsulation with less than half familiar with Law of Demeter. As an understanding of these concepts is a requirement for understanding why bad practices (which go against these best practices) it is useful to know whether respondents are familiar with best practices rather than bad practices.

This is a secondary assessment of ability in addition question 1.

Predictability is not a factor for this question as personal information is being asked rather than their opinion on the research.

Question 7. Do you try to follow Object-Oriented best-practices when developing software?



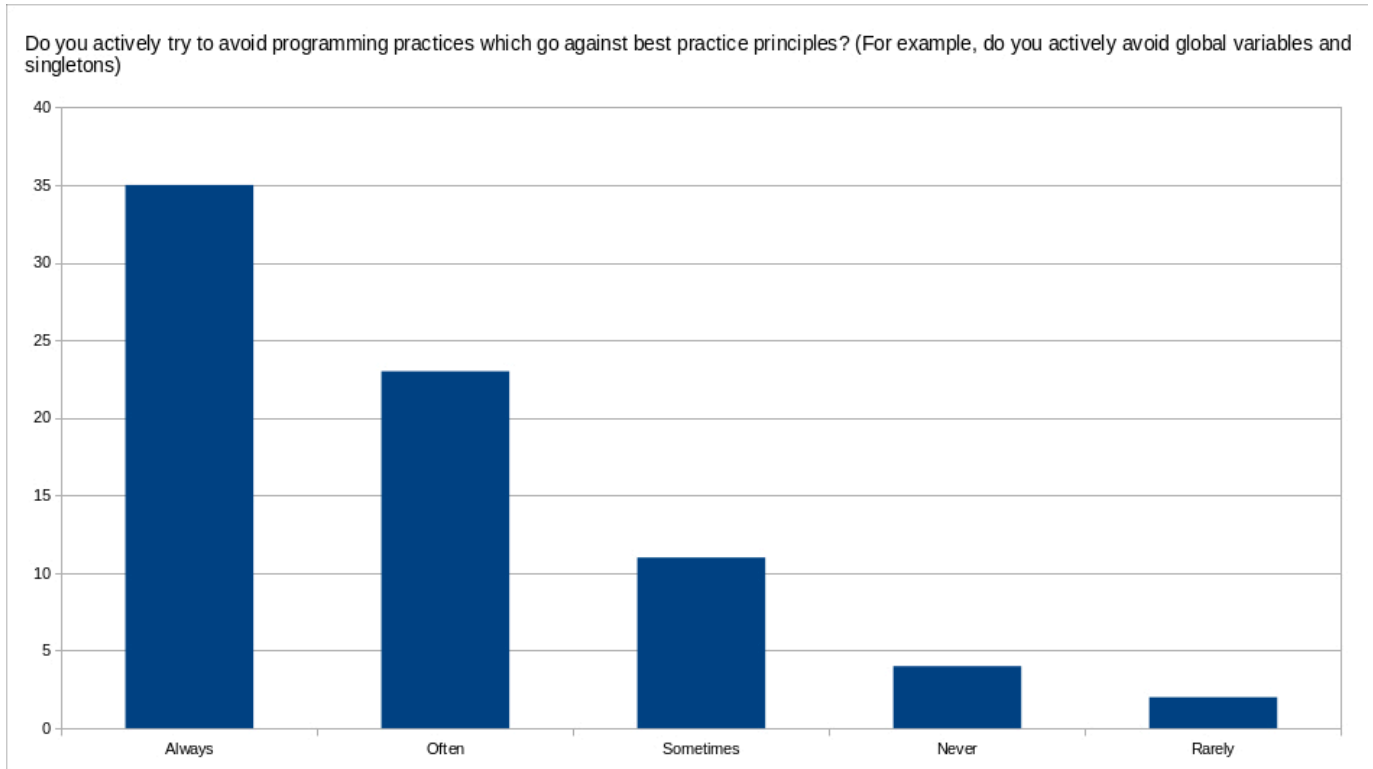
**Figure 6.10:** Question 7. Do you try to follow Object-Oriented best-practices when developing software?

This question was asked to determine whether respondents actively follow object-oriented best practices. Those who do not are unlikely to find the metric useful.

It is predicted that those who answer never or rarely to this question will rate the usefulness of the metric lower (Questions 13, 14 and 15)

Question 8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)



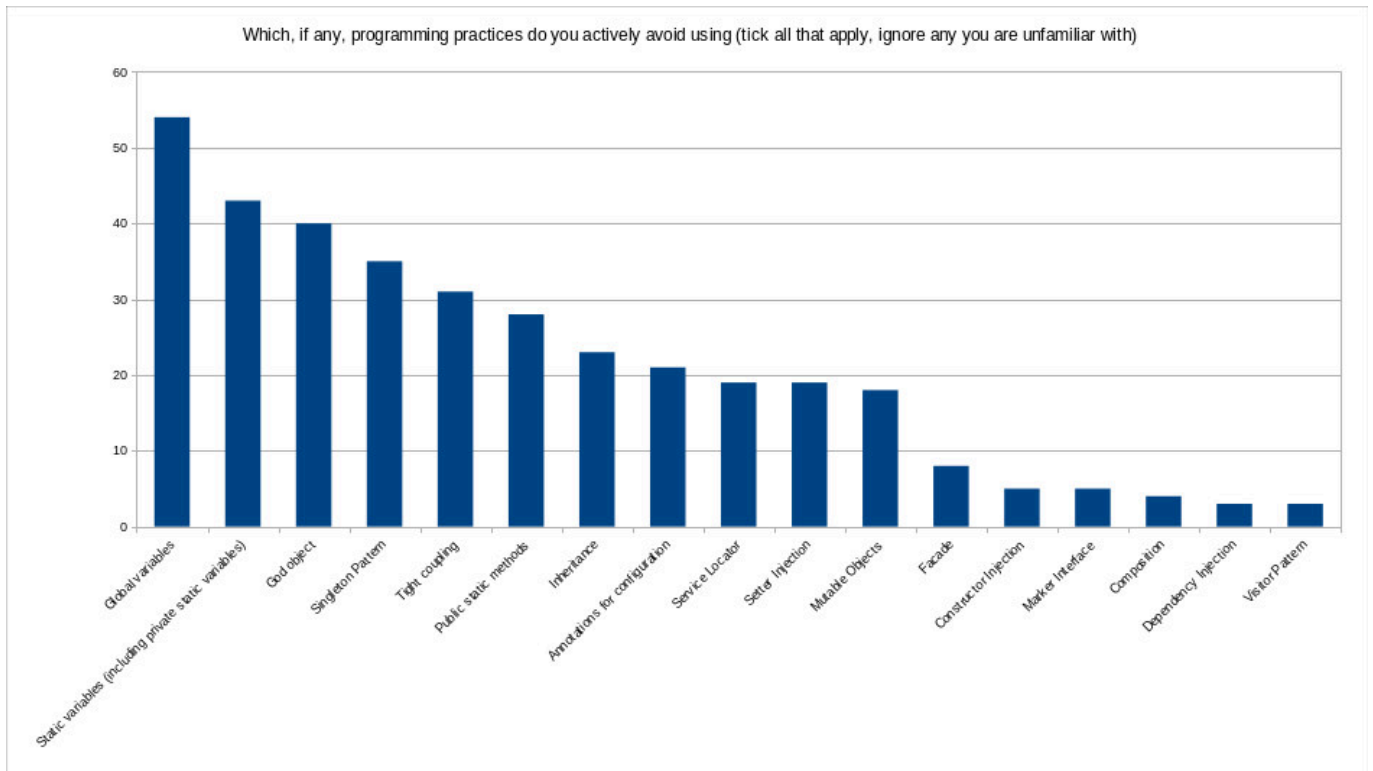


**Figure 6.11:** Question 8. Do you actively try to avoid programming practices which go against best practice principles? For example, do you actively avoid global variables and singletons

This question was asked to see whether programmers try to follow best practices or not. 35 of 75 respondents answered answered "always" indicating that nearly half try to follow best practices all the time.

Among the 31 senior developers, all of them chose either "Always" (25) or "Often" (6) indicating that senior developers keep best practices in mind during development moreso that novices and students, which is unsurprising.

Question 9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)



**Figure 6.12:** Which, if any, programming practices do you actively avoid using tick all that apply, ignore any you are unfamiliar with

This question was asked to see which practices developers avoid, to infer which they consider "bad practice".

This list includes known best practices composition, dependency injection and constructor injection.

Of the 3 people who said they avoid dependency injection, one classed themselves as a senior developer, one as a student and one as an open source developer. This goes against common best practices (as shown in the meta-analysis in chapter 3)

In addition to saying they avoid Dependency Injection, the senior developer and open source also said that they avoid global variables indicating that they did not misunderstand the question and assume it was asking which practices they use.

The student did not say they avoid global variables but they do avoid static variables. This is inconsistent, but as they are a student it is not expected they will be as nuanced as the other two who rate themselves as more experienced.

This shows there is some genuine disagreement from a senior developer. This is to be expected as not everyone will agree on everything. In chapter 3 it was shown that once academic rigour was

taken into account, these disagreements are filtered out.

30 of the 31 (96%) of senior developers surveyed did not say that they avoid dependency injection implying that they do not consider it a bad practice.

Question 10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)

*There are a few (very few) valid uses for the Singleton, but most of the time it is an anti-pattern.*

---

*Inheritance is has even more valid uses, and judging when to use it or not use it is probably extremely difficult to judge in a tool like this.*

---

*You should make the raw data available so we can see the "good" and "bad" articles*

---

*Found the background research really interesting as like you mentioned best practice can be very much open to interpretation! Was a really insightful read*

---

*I believe it is a nice tool which helps improve best practices*

---

*I feel the background research was very interesting to read. It is always nice to come up with tools that are not available out there. I believe this tool is great and I would like to use it in the future.*

---

*Reviewing code and finding weakness is a good idea. We do this all the time however a large body of professional programmers really don't take the sort of care you may think over code. There are a lot of "paycheque programmers" just churning out mediocrity, what is also an issue is that people blindly follow tools that tell them things are a bad idea. Granted a lot of the things discussed here are bad news, ServiceLocator etc I particularly detest setter injection. The reality is often software engineers are following the JFDI process their boss has laid out and are really thinking about their next craft beer than software process.*

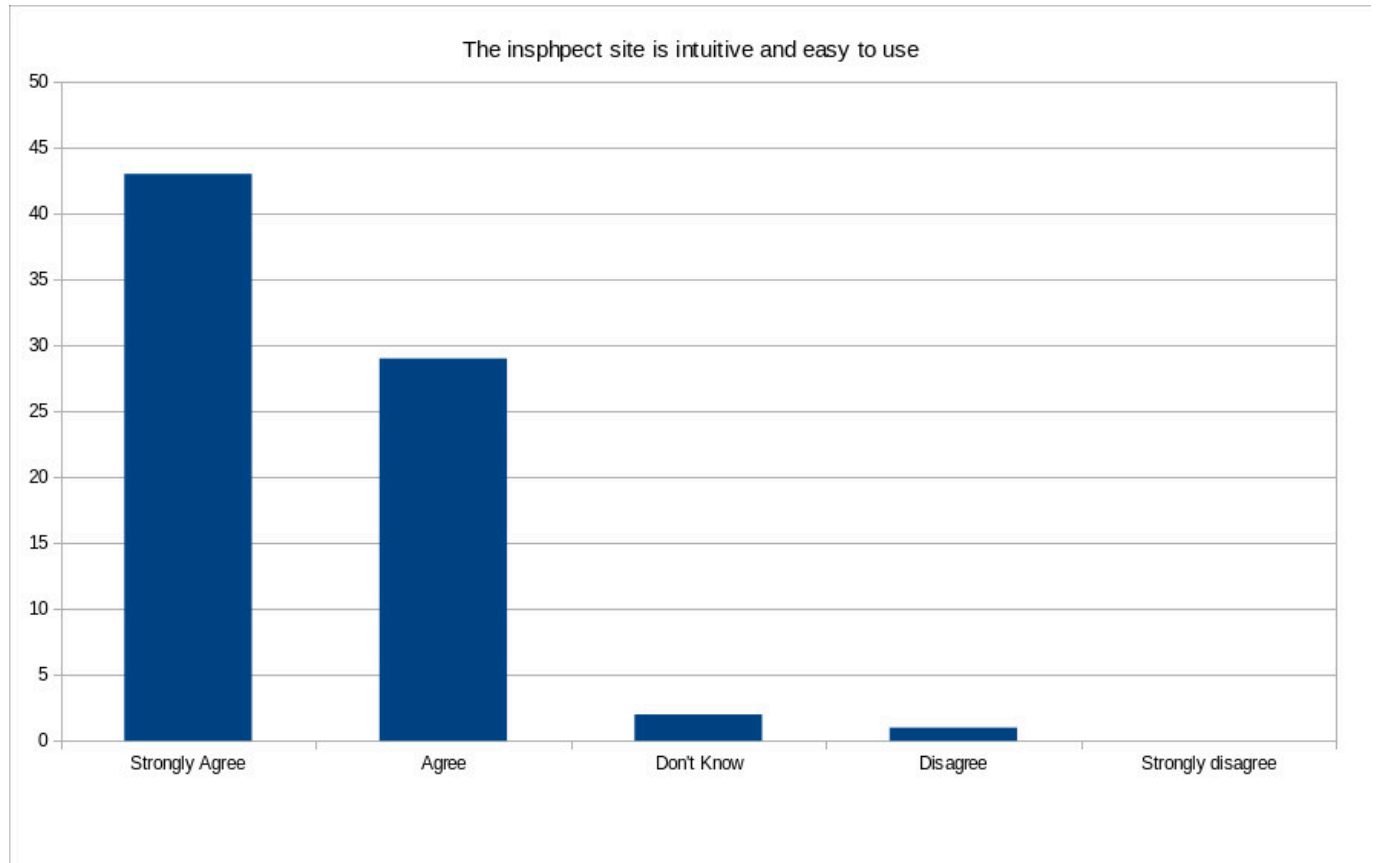
---

*The feedback is very informative but I feel like having a huge piece of text and examples opening when clicking a line is a bit jarring. Maybe these should be short examples and link off to external articles that are more detailed. Also it would be great for CI purposes if this was a Composer package that people could use to maybe automatically add GitHub comments or something along those lines? Just some ideas. Great project*

---

This question was free-text and asked to get opinions on the background research. Several answers were about the tool in general but overall the opinion of the background research was positive.

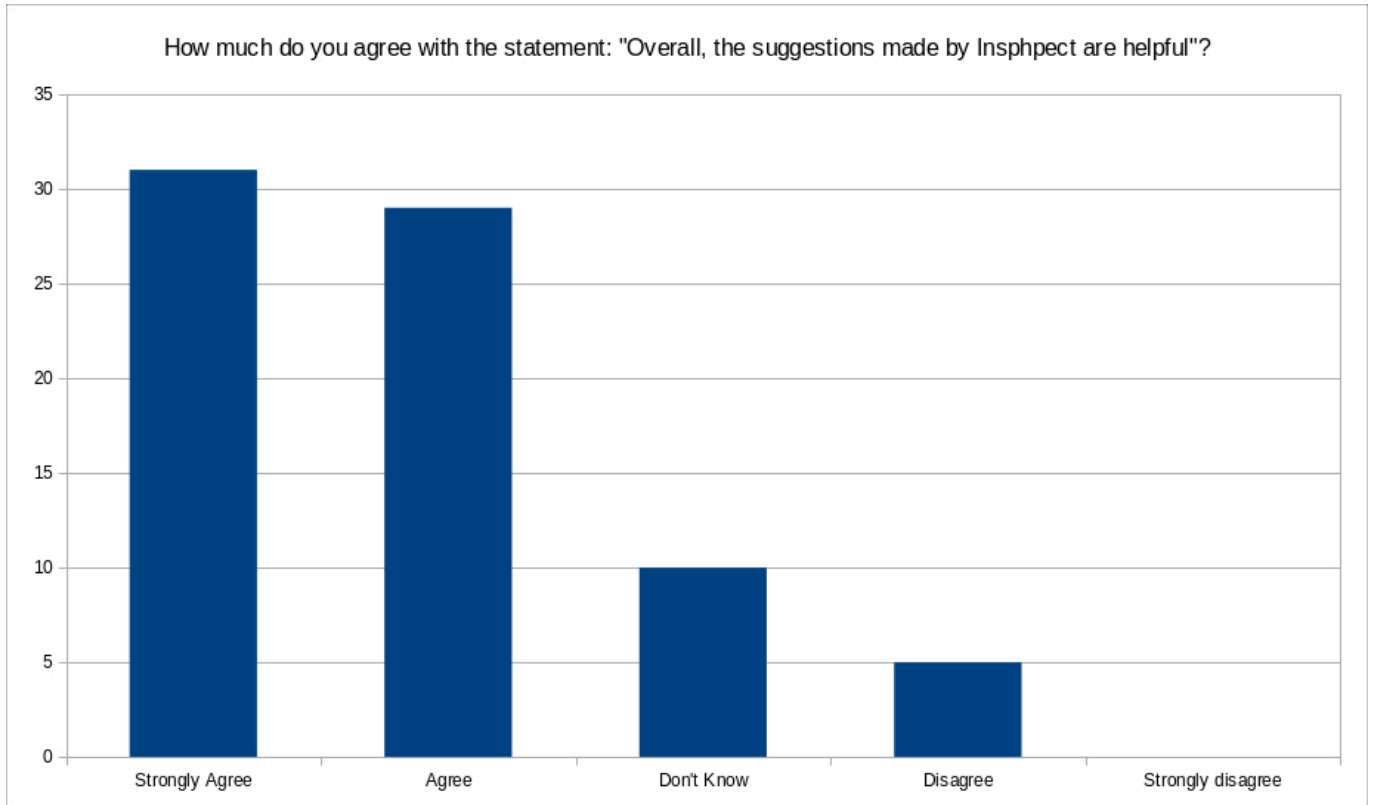
Question 11. The inspheight site is intuitive and easy to use



**Figure 6.13:** Question 11. The inspheight site is intuitive and easy to use

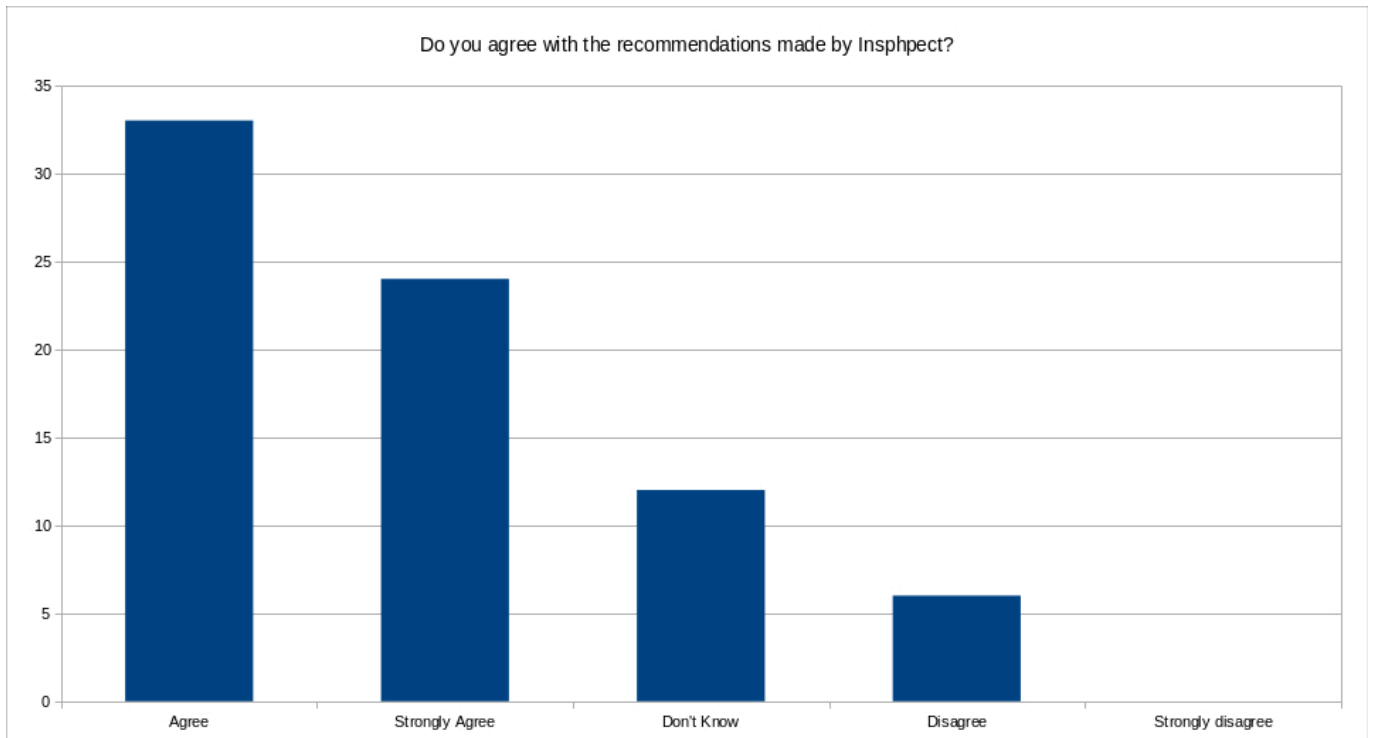
This question was asked to inquire about the usability of the website. Nobody responded "Strongly disagree", one respondent selected "Disagree" and the majority, 43 (57%) selected "Strongly Agree".

Question 12. How much do you agree with the statement: "Overall, the suggestions made by Inspheight are helpful"?



**Figure 6.14:** Question 12. How much do you agree with the statement:"Overall, the suggestions made by Insphpect are helpful"?

Question 13. Do you agree with the recommendations made by Insphpect?

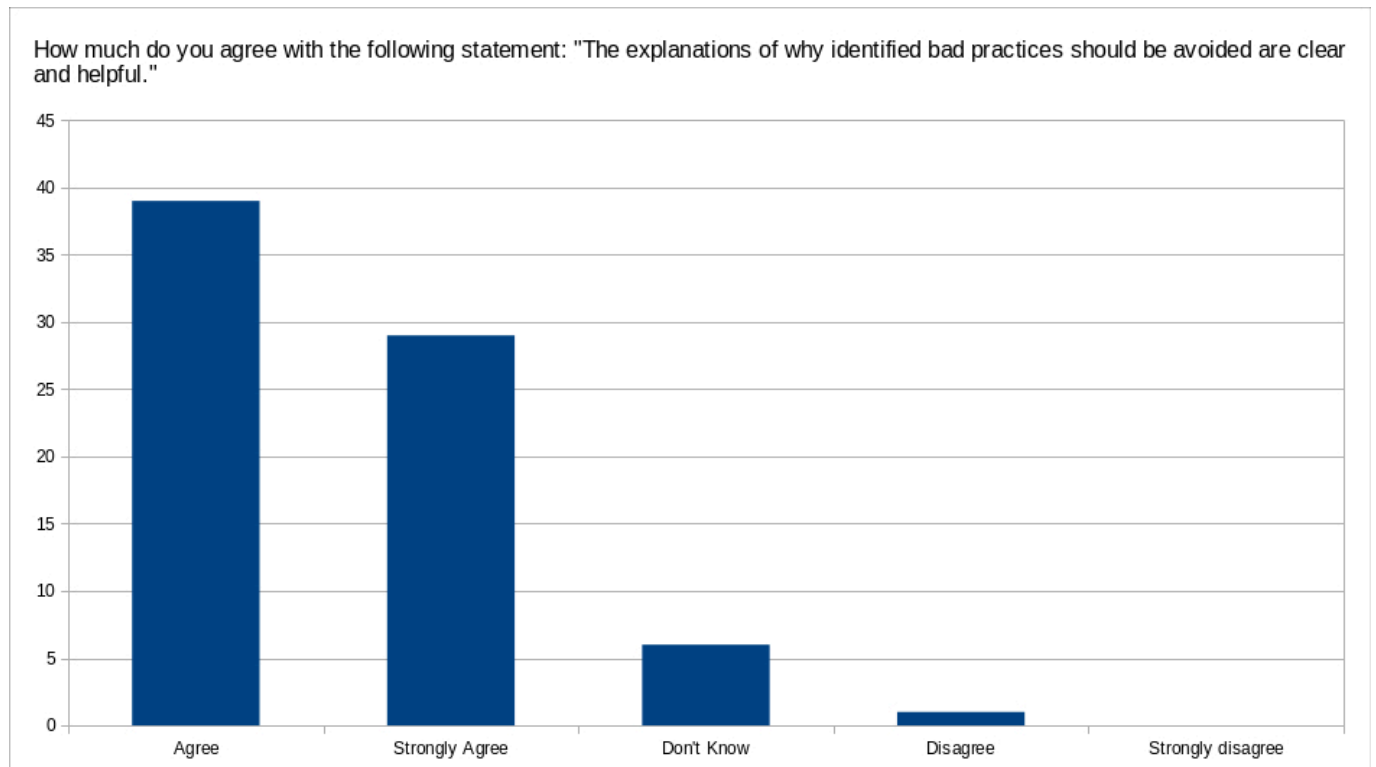


**Figure 6.15:** Question 13. Do you agree with the recommendations made by Insphpect?

This is one of the most important questions asked in the survey. This is to determine respondent's opinions of the metric and identification of bad practices.

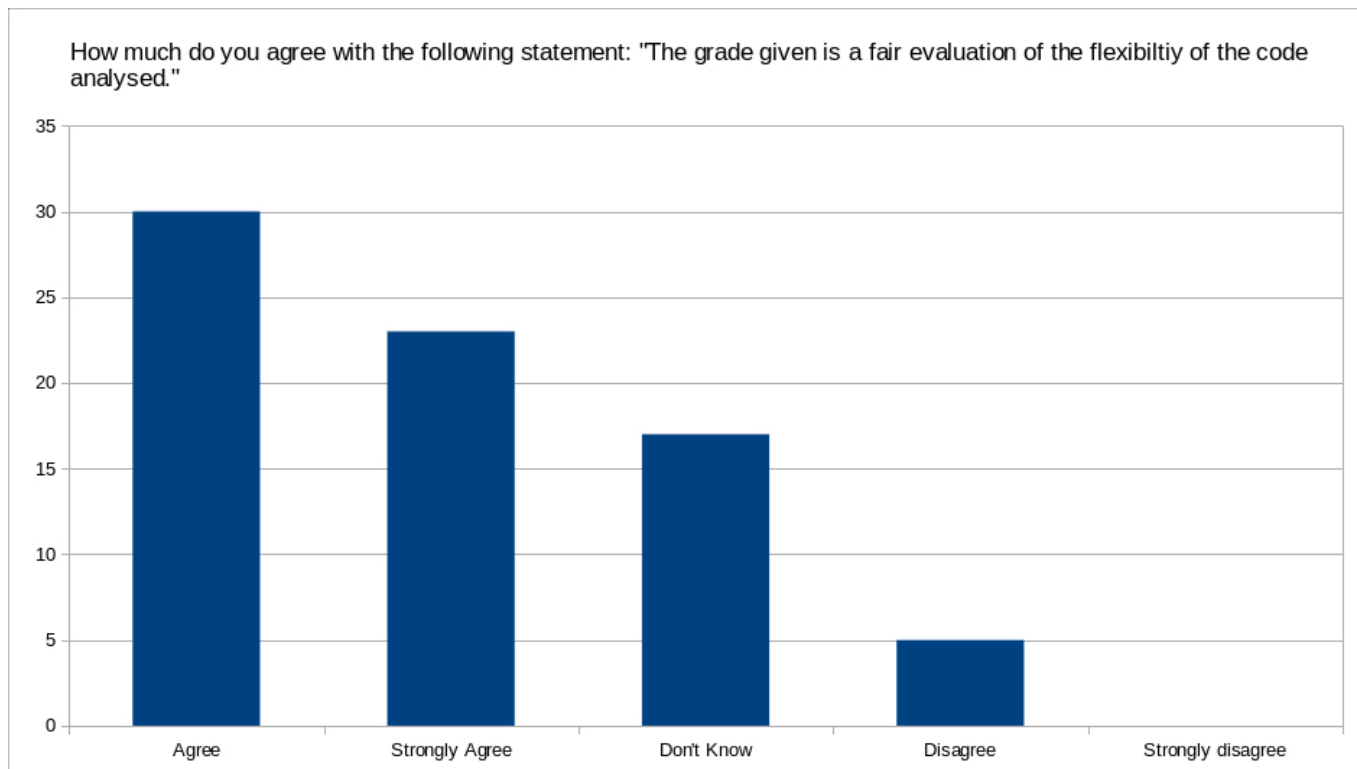
The majority agree or strongly agree, this question will be broken down by programming ability in the following section.

Question 14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."



**Figure 6.16:** Question 14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."

Question 15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."



**Figure 6.17:** Question 15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."

This question was asked to determine people's opinion of the grade. As this grade is a calculated attribute with the scale chosen to provide a wide range of grades it is important to know if people thought the grade given was fair.

68 (90%) of agreed or strongly agreed with the grade given. 0 strongly disagreed and 5 (6.6%) disagreed with the grade.

Of the 5 who disagreed, 1 was a student and 4 were senior developers. Only three of them left comments on why they disagree:

---

*wouldn't consider it as something that is missing, but possibly a cut-down version of the explanation. A summary of the issue without too going in-depth with the explanation*

---

*The feedback is very informative but I feel like having a huge piece of text and examples opening when clicking a line is a bit jarring. Maybe these should be short examples and link off to external articles that are more detailed. Also it would be great for CI purposes if this was a Composer package that people could use to maybe automatically add GitHub comments or something along those lines? Just some ideas. Great project!*

---

*As a tool designed specifically for inspecting PHP code, I would recommend disregarding*

*Inheritance issues when the class in question extends one of the built-in Exception classes, as there is no other way in the language to throw custom exceptions.*

*And for a more opinionated suggestion, you may want to add an option to disregard Inheritance issues when the class in question extends an abstract class. (An option for advanced users, as opposed to by default, because it should still be discouraged in general.)*

*Those two changes alone should take care of the overwhelming majority of false positives in the inspection I ran.*

---

While two also offered positive comments:

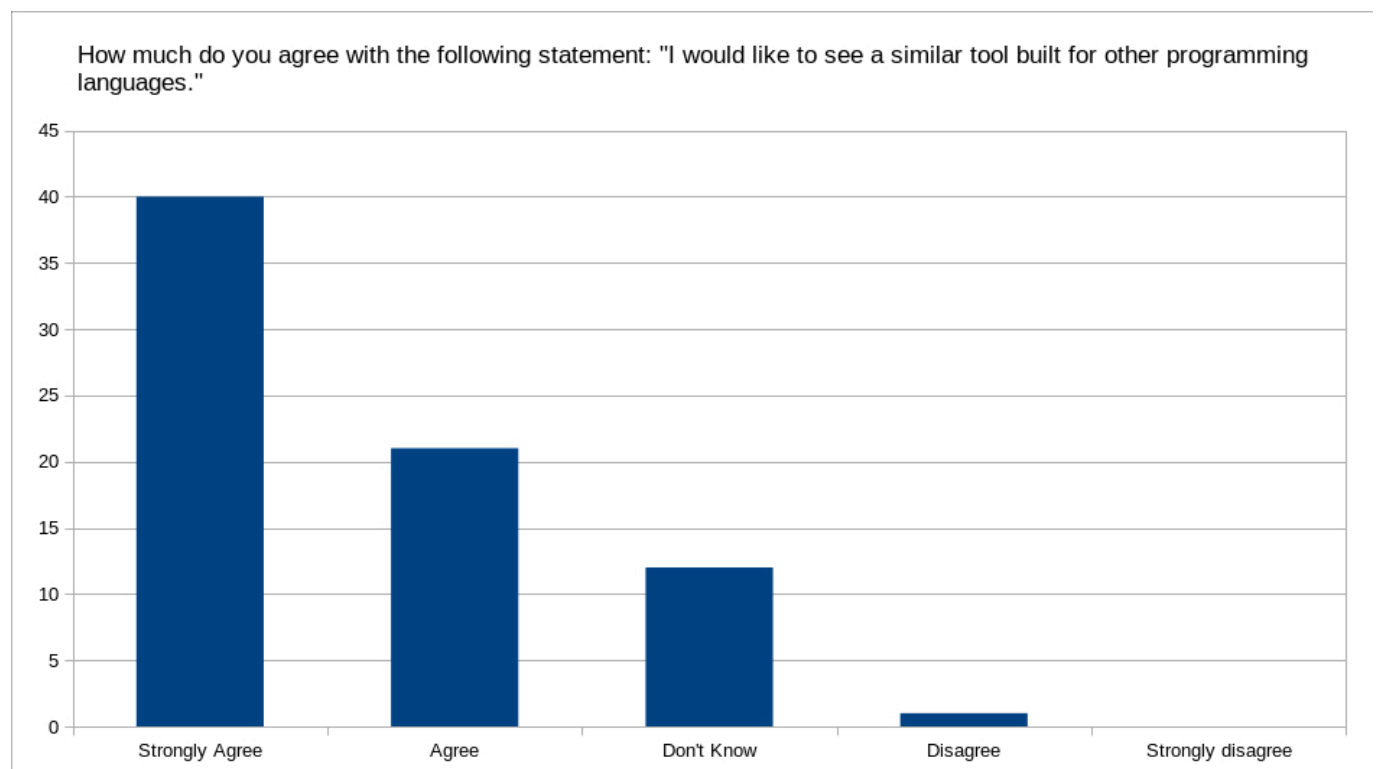
*Good tool, keep up the good work*

---

*I think it's a great idea and I would love to see a similar tool for other languages that would apply more to what I use*

---

Question 16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."



**Figure 6.18:** Question 16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."



Question 17. Is there anything you think is missing from Insphect which should be included in a future update?

This was a free-text question to allow respondents to provide more detailed opinions. A selection of answers are included below. The raw data is available in *appendix X*.

*As a tool designed specifically for inspecting PHP code, I would recommend disregarding Inheritance issues when the class in question extends one of the built-in Exception classes, as there is no other way in the language to throw custom exceptions.*

*And for a more opinionated suggestion, you may want to add an option to disregard Inheritance issues when the class in question extends an abstract class. (An option for advanced users, as opposed to by default, because it should still be discouraged in general.)*

*Those two changes alone should take care of the overwhelming majority of false positives in the inspection I ran.*

---

*I think the information given about an error is too much. 1 easier back and forth navigation when navigating between analyzed files*

---

*I think, as a testing tool, that I think is valuable to good code quality - it should be possible to create an adapter that promotes Test-Driven Development and Behaviour-Driven Design. This would really provide for a cool feature to base a Code Review Service and so on and so forth.*

---

*The responsive mode is not good.*

---

*Flexibility is so important, (Composition over Inheritance) I add it in my best practices list! Thanks a lot for you work, future PhD!*

---

*I'd like to be able to run it locally rather than trusting a site with my private code*

---

*It would be nice to customise the rules, so certain practices which the tool may consider bad could be ignored for a project that needs to use that practice for a valid reason. A tool I use called Sonar for C# has a concept of "Ways" which would be similar.*

---

*Having less information when first clicking on the issue with an option to read-more? e.g. just display the summary, then have another button to expand the data.*

---

*I'd like to see it handle inheritance differently. There are some cases where PHP forces you to use inheritance (e.g. extending `InvalidArgumentException`) and it's flagged up as bad as extending one of your own classes. The explanation popup should at least mention this or grade the code differently when extending an inbuilt class. Another example is PDO. Sometimes it's useful to add functionality to it and the only way that can currently be achieved is with inheritance, if you wrap it you can't then pass it into code that expects a `\PDO` instance so any code which does this is going to lose points for doing something unavoidable even if it does technically introduce tight coupling.*

---

*I wouldn't consider it as something that is missing, but possibly a cut-down version of the explanation. A summary of the issue without too going in-depth with the explanation*

---

*For tooling like this to be valuable, it should be able to be run in an automated fashion from the CLI, for inclusion in build processes.*

*Can't run locally. It's fine for testing some toy project or open source library, but I can't possibly use it at work like this.*

---

*I'd like to be able to analyse a private repository and keep the reports behind a login system. It looks like all code uploaded is available on public URIs which prevents me from uploading code I write for work.*

---

Overall, the responses are positive. There are some common complaints:

- Being able to run the tool locally, rather than on a website.
- Responsive mode (viewing the website on a mobile phone)
- Cutting down the text into a summary/full text

These have been noted and are important features to add going forward.

One respondent said "Tools for other languages that do this do exist." but did not elaborate on what those tools are and they were not discovered during literature review. It's unclear if the respondent meant specifically grading code by identifying bad practices, or tools that grade code more generally.

Question 18. Do you have any general comments about Insphpect? This question is free-text and asked to get general feedback about the tool and metric. A selection of answers are included below, the complete raw data with all responses is available in *appendix X*.

*I think it should be available in an offline version. I'm reluctant in uploading private code on the web*

---

*I am hoping the tool is easy to integrate with given it has things like GuzzleHttp port and that of Symfony Routing*

---

*The site doesn't work well on mobile*

---

*An interesting and cool tool and proof of concept. The website design is perhaps not really what I would have gone with but it's certainly quirky and the animation give it an interactive and fun feel. The background on the project is really interesting as is the methods used to assess programming best/worst practices. Great name too!*

---

*Would love to see this tool developed into a plugin for common IDE's and editors and become available for other languages. 1 It looks fabulous and it is very simple and easy to use. I can quickly see where there are issues in what files and how to improve it*

---

*PHP requires thrown objects to be a subclass of Exception so perhaps it should not complain about that.*

---

*A very useful tool. One possible concern with making code as generalised as possible is it could potentially lead to new developers to a project taking a bit longer to get up to speed.*

---

*A proactive tool, helpful for every user, independently of their experience in programming. Well done.*

---

*I've been following your work for years and learnt a lot. This is a clever progression from what you've been doing previously. Great job!*

---

*I use Scrutinizer regularly and find it to be helpful but rather dumb. As you point out, there are thresholds for grades which seem to have been chosen at random. I like Insphect's approach of identifying antipatterns a lot more as it has a more solid foundation for the grades given.*

---

*Good tool, keep up the good work*

---

*Would love to have a command line version to add to my tooling, along with phpstan, phpmd, phpunit, phpmd, etc. :-)*

---

*The "What is tight coupling" page is the best concise explanation I've seen, will be adding that to the list of reading materials sent to all new developers who join the company.*

---

*Totally not agree with annotations and red warning about inheritance. Modern programming will always depend on some other library; it is not enough just to implement interface.*

---

*Since you are unable to run it locally (or on servers you control) it is not possible to run it on any client work, due to restrictions in the contract about who you can give access of the code to. In other words, as it is, it can really only be utilized for open source projects.*

---

*Nice tool, I'll definitely be using this on my projects moving forward.*

---

*The concept is really interesting. I'd be really interested to see how this impacts the quality of code in the future if it is something budding coders are introduced to early on in their learning.*

---

*On the couple of repositories I looked at there were definitely a few debatable red lines. For example, private static, sure it does introduce global but it's unlikely to cause any real world issues as the scope for damage is limited to the class it's used in and you'd hope that static was used for a good reason*

*Even if you still identify it as introducing global state, public static and private static should not be treated the same way.*

---

The responses are generally positive, with most of the complaints being the same as above (wanting to run the tool locally and responsive mode improvements). There were several disagreements with some of the issues picked up with some edge-cases that may warrant more investigation. However, comments such as:

*And yes, it introduces tight coupling but some things are just primitive enough or specific enough it is not worth decoupling them.*

---

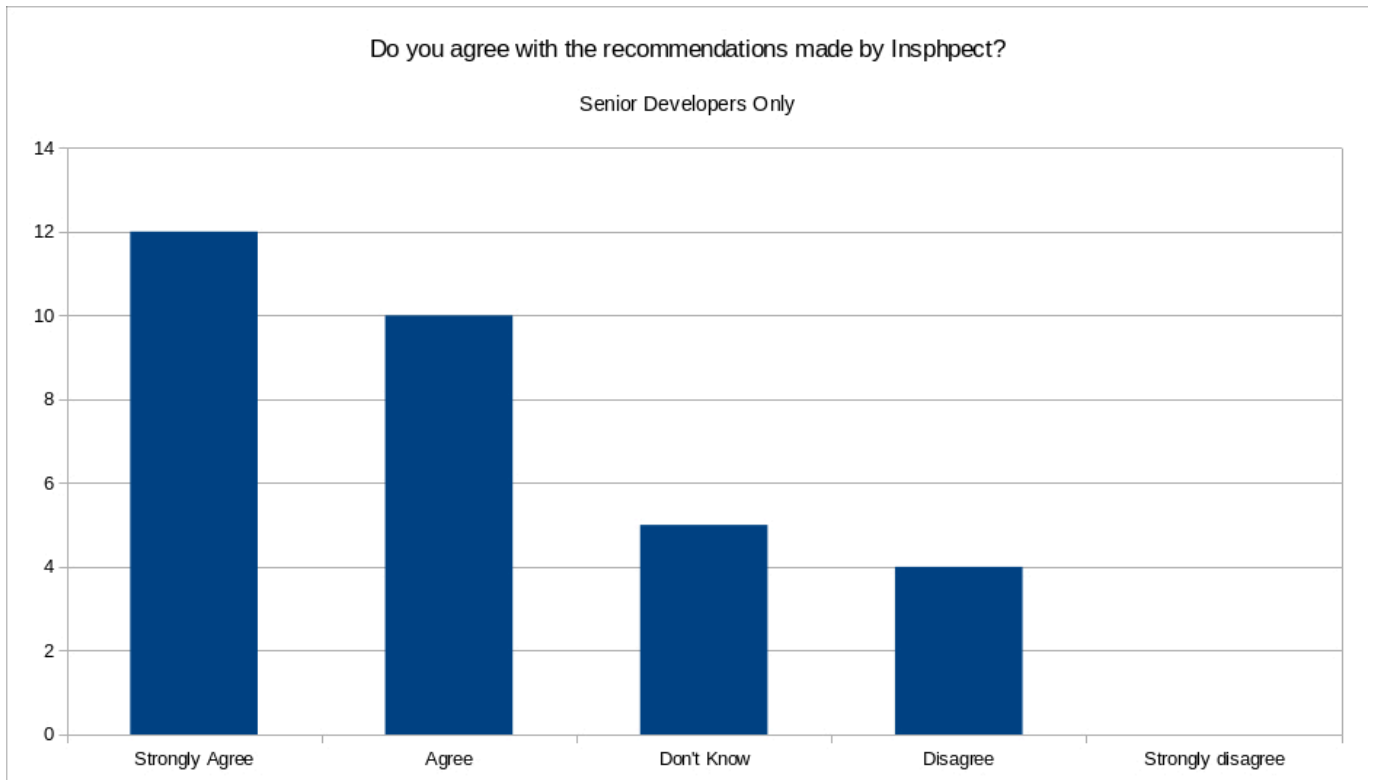
Such comments miss the purpose of the tool and metric. The metric detects inflexible code such as tight coupling, it does not attempt make a decision whether the additional flexibility is "worth it" as doing so would be difficult or even impossible to quantify.

## **6.7.2 Conclusions**

Overall the results are positive with 75% of respondents agreeing or strongly agreeing with the

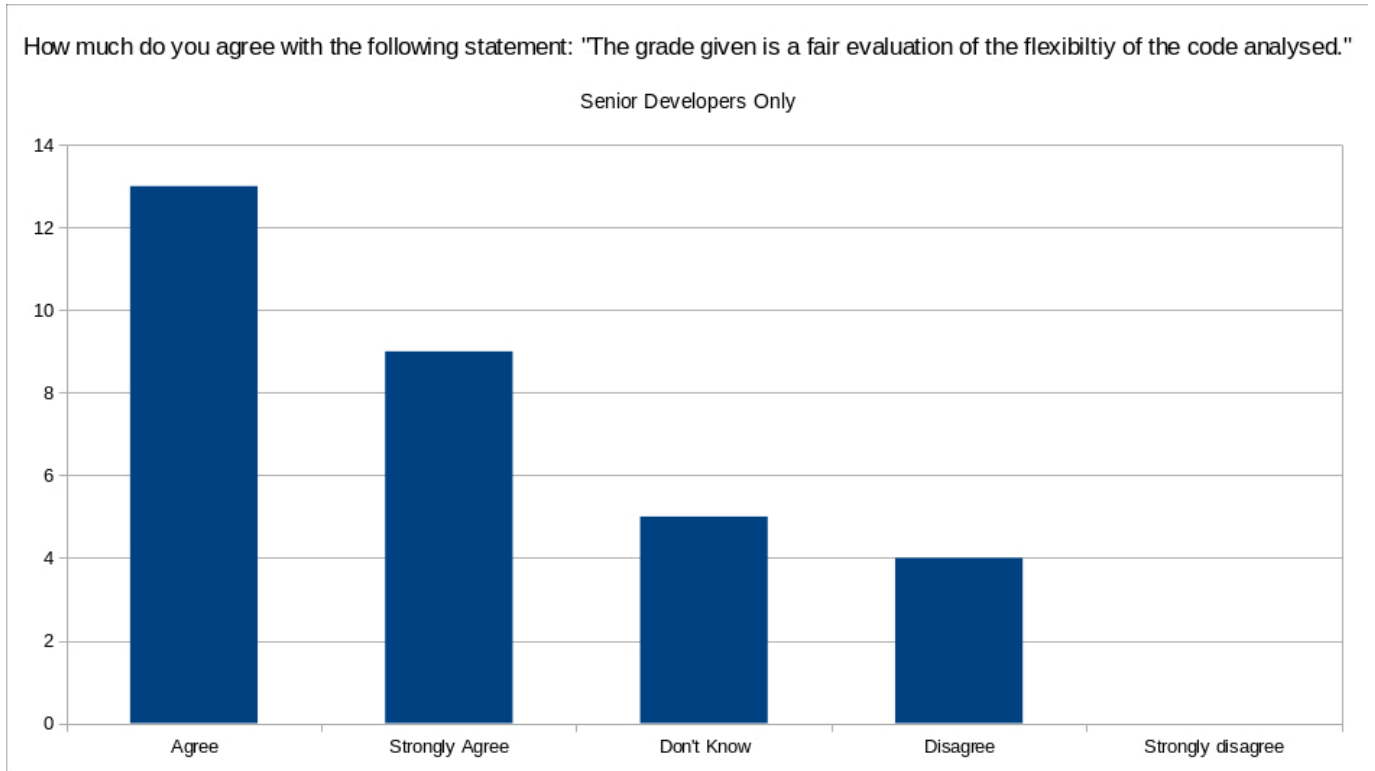
statement "Do you agree with the recommendations made by Insphpect" with just 8% disagreeing and 0% strongly disagreeing.

Breaking down the results by programming ability (figure 6.19), with users who described themselves as "Professional: Senior Developer" the results are very similar to the overall result set though 12% disagreed with the recommendations given and none strongly disagreed



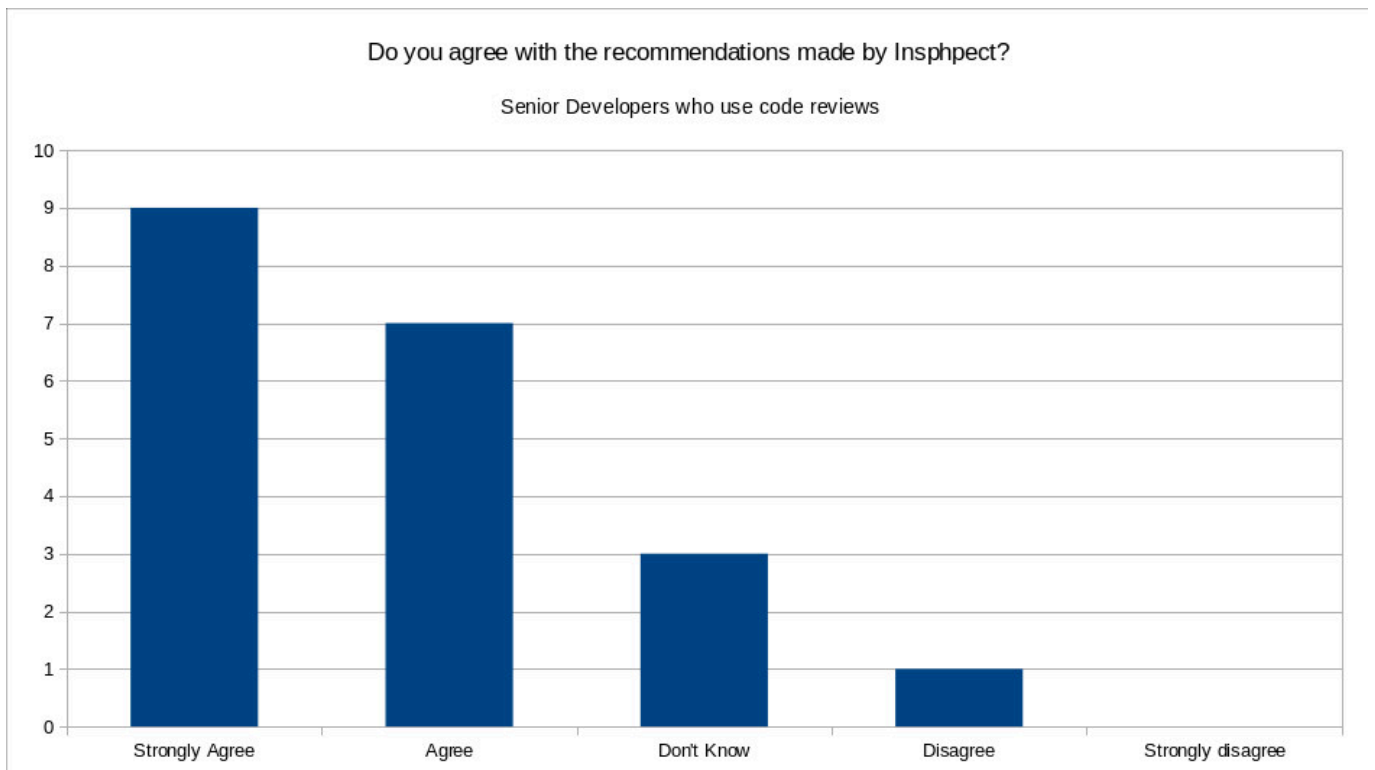
**Figure 6.19:** Question 13. Do you agree with the recommendations made by Insphpect? Senior Developers Only

Senior Developers also generally agreed with the grade given as shown in figure 6.19:



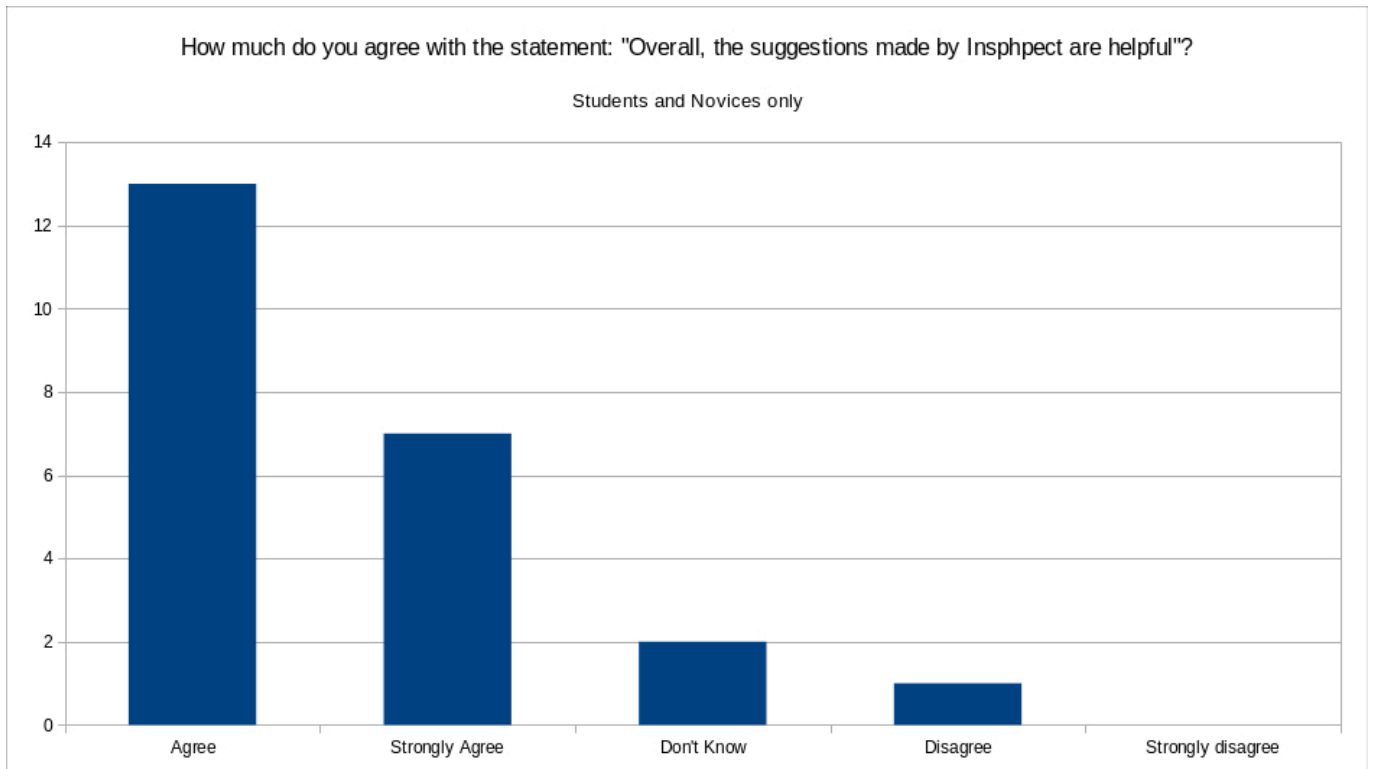
**Figure 6.20:** How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed." Senior Developers Only

Breaking the Senior Developers down further to those who answered "Yes" to "Do you use code reviews as part of your workflow?" gives a similar result, however a higher percentage chose "Strongly Agree" rather than "Agree".



**Figure 6.21:** Question 13. Do you agree with the recommendations made by Insphpect? Senior Developers who use code reviews

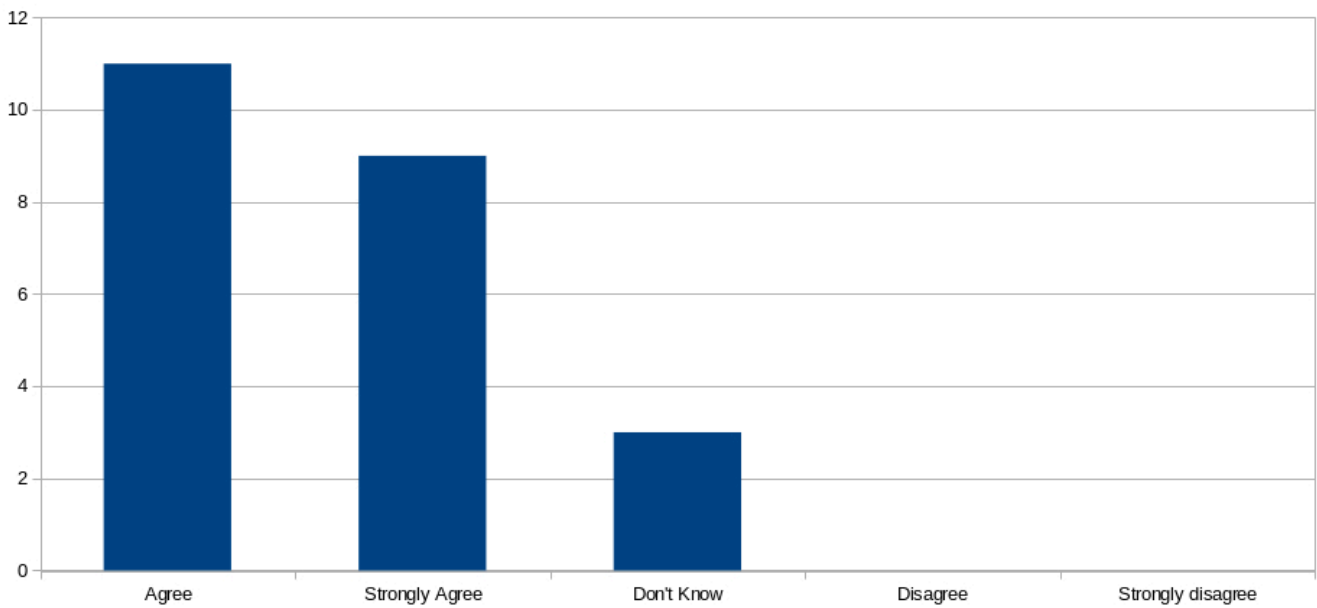
At the other end of the spectrum those who described themselves as students or novices, overwhelmingly found the recommendations made by Insphpect helpful.



**Figure 6.22:** Question 12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"? Students and Novices only

How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."

Students and Novices only



**Figure 6.23:** Question 14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful." Students and Novices only

Shown in figure 6.23, novices and students overwhelmingly found the explanations of bad practices helpful.

### 6.7.3 User comments

Comments from users were mostly positive with a few common complaints.

The primary complaint was that the tool could not be run locally and had to be used as a website with 6 respondents (8%) asking for this feature. The second biggest complaint was that the site is not optimised well for mobile phones (5% of respondents).

An offline version could be provided in future and site redesigned to be more mobile friendly. These are valuable feature requests, however these they do not impact the evaluation of the metric or research project.

On the metric itself, several users had concerns about specific recommendations made by the tool and identified edge cases where it might need tweaking. For example:

*On the couple of repositories I looked at there were definitely a few debatable red lines. For example, private static, sure it does introduce global but it's unlikely to cause any real world*



*issues as the scope for damage is limited to the class it's used in and you'd hope that static was used for a good reason.*

*Even if you still identify it as introducing global state, public static and private static should not be treated the same way.*

---

*PHP requires thrown objects to be a subclass of Exception so perhaps it should not complain about that.*

---

These are valid concerns and could be handled differently in future versions of the metric/tool by treating these differently to other cases of these bad practices. However, there is a trade-off between trying to provide a general purpose metric which produces consistent results and catering to language-specific exceptions.

By enabling users to turn on/off or tweak specific identification rules the metric is less meaningful because the overall score would not always be based on the same rule-set for each project. The only way to have a meaningful score would be to scan multiple projects with the same rule-set.

## **Conclusion**

There are some areas for improvement but overall the project feedback has been positive. The negative feedback is generally about the interface and usability of the tool rather than reservations about the grade or the general issues identified.

## **6.8 Project outputs**

The following outputs have been produced as a result of this project

1. An aggregation of bad practices which impact code flexibility. This was published in a paper presented at the China-Europe International Symposium on Software Engineering Education conference in 2017 (Butler, 2017) (Appendix V)
2. A metric was produced for grading the academic rigor of an article discussing a programming practice, based on a methodology used for grading clinical trials in medicine.
3. A methodology for conducting a meta-analysis was created using the academic rigor grade. This methodology was presented in a paper at the Proceedings of the 2019 Computing Conference (Butler, 2019) (Appendix VII).

4. Meta-analyses were produced for each of the identified bad practices to show what developers thought of the practice.
5. A metric was created for measuring the flexibility of Object-Oriented code by identifying bad practices.
6. A tool was created to analyse PHP projects and grade their flexibility using the metric.
7. An article was published on industry website sitepoint.com introducing the tool and the metric (Appendix IX).
8. The tool was extended to offer automated corrections in some cases.

## 6.9 Project strengths

This project achieves what it set out to do, a metric has been created which can be used to grade the flexibility of source code by identifying programming practices which negatively impact flexibility.

The objectives set out at the start of the project were all met and a tool has been published which allows users to test the metric on their own code.

In addition, during the project a methodology was created for performing a meta-analysis on programming articles.

The additional objective has been shown to be possible on one of the bad practices and the tool is able to provide automated corrections in some cases. It is even able to generate a patch which can be applied to the code.

The partial completion of this additional objective shows that automated fixes are possible and future work will be carried out performing the same task on the remaining bad practices. This is a feature which has not been implemented on this scale before in other metrics. Scrutnizer and similar tools can automatically remove unused variables and fix typos, but they cannot re-structure the architecture of the application to automatically replace tight coupling with loose coupling.

## 6.10 Project weaknesses

The primary issue with the evaluation is the sample size. The results would have a lot more weight if more people had completed the survey. However, as the survey was left open for another two months and advertised on a very popular website, it is not clear how this could have been achieved.

A different programming language may have yielded more results. Python and Java tools might have garnered more interest. However, as the PHP version of the tool took over 6 months of development time, additional tools for other languages would not have been feasible, though could have been chosen up front as the primary language to use.

Secondly, it would have been useful to log where survey respondents came from. Although this data is logged and it is possible to see how many visitors the site got from various sources (with the standard caveat that this information is not 100% accurate), it would have been beneficial to have logged this along with the survey response. For example, to see if people coming from Google gave different answers than people coming from Reddit.

Though with the relatively small sample size acquired, breaking data down this way may not have been meaningful as the sample size from each website would be very small.

## 6.11 Research relevance and use-cases

It is envisaged that the tool and metric could be used for the following cases:

1. As a learning tool. Novices and junior developers can upload their code and see explanations of where improvements can be made and learn why the practice has been flagged up and what they should be doing instead.
2. As a part of a business quotation. The initial idea for this research came from real-world experience. Programmers often inherit projects or are asked to work on code that was previously written by someone else or build using third party libraries. Often they are expected to give quotes for how much a new feature will cost. Doing so requires a lot of up-front work looking through the code to see what is needed to make the required updates. Depending on how the software was written will determine how much the changes will cost or even whether the developer wishes to take on the job. The tool created can be used to give an overview of how flexible inherited code is or is not.
3. As a code-review tool in companies. It is common that code written by junior developers is

reviewed by senior developers before being committed to the project. This can be a long back and forth with several revisions before the code is finally committed. The metric and tool could be used as initial step, saving the code reviewer time filtering out bad practices and enable them to spend more time reviewing the logic.

4. As a continuous-integration tool. Although the tool is not currently capable of automatically scanning source code when new commits are added, if this was introduced in a later version, the metric could be employed as part of a continuous-integration work-flow, checking that no inflexibility has been added to the code each time someone commits a change.

## 6.12 Future improvements and limitations

Each chapter in the research has scope for further research and contains its own limitations.

### 6.11.1 Future improvements/Limitations - Chapter 2 - Aggregation

Chapter 2 collated information about each bad practice and described it. Currently the data gathered is used for this research only. The JSON file format developed and the data stored using it could be made public so that others could describe other programming bad practices in this manner or embed the data in their own research/tools/software.

The documentation format could also be extended to documenting programming best practices by noting *positive traits* instead of *negative traits*.

Additional bad practices could be documented. One such bad practice which was identified later in the research is mutability. A (fairly) recent programming trend in object-oriented programming is making objects *immutable*. This has numerous advantages when it comes to flexibility and would be useful to be added to the list and the metric.

### 6.12.2 Future improvements/Limitations - Chapter 3 - Meta-analysis

Chapter 3 introduced a meta-analysis methodology for comparing articles about any given bad practice.

In addition to performing the meta-analysis on additional bad practices, this methodology could be utilised for performing meta-analyses of programming practices outside the scope of *flexibility*, for example, security.

For extended scope meta-analyses it would be possible to extend the methodology by defining a custom *Jadad-style* score and *recommendation* score. For example, performing a similar meta-

analysis for software *performance* the recommendation score could be based on benchmarks in the article while the *Jadad-style* score could be adjusted to measure the rigour of the study by asking questions such as *Did the article provide the hardware being benchmarked on?*, *Did the article provide the software environment variables (e.g. php.ini settings)?*, etc.

A very useful piece of future research would be to run the same meta-analyses at intervals such as each year over a 5-10 year period. This would track developer opinions over time and it would be possible to see if developers attitudes change as time goes on. A similar study could be done with the existing data by breaking articles down by publish date.

### **6.12.3 Future improvements/Limitations - Chapter 4 - Metric**

Chapter 4 introduced the metric for grading software based on the frequency of bad practices it contained.

The metric itself is already extensible, it would be easy to add additional bad practices and incorporate them into the grade generated. However, doing so would invalidate all previous grades. As such, a versioning system could be introduced to identify which revision of the metric was used to calculate a particular grade.

### **6.12.4 Future improvements/Limitations - Chapter 5 - Tool**

In chapter 5, a tool was developed which enabled other people to test the metric for themselves by uploading some code to be analysed.

User feedback provided several areas where improvements are needed. Most of these are cosmetic or usability issues:

1. *Better usability on mobile phones.* This is an area which requires a significant amount of work to improve as code is generally written with the intention of being displayed on desktop computers. The pages which do not display source code could be re-designed to work better on mobile phones, however.
2. *Summarise the issues.* This was noted by several respondents of the questionnaire. The large block of text detailing the issue is not as user friendly as a summary with an option to expand.
3. *Allow the tool to be run locally.* This was the number one suggestion and the easiest to implement in future. As the entire site runs in docker, it can easily be packaged and

distributed. In addition, as the front-end and back-end are entirely decoupled and communicate only via a very simple JSON API, it would be simple to make a command line application. Doing so would give users access to the code, potentially causing security issues on the live website.

Additional suggestions by respondents such as omitting language specific bad practices which are forced by the language would require a potential redesign of the metric. An example given by a survey respondent is in PHP programmers are forced to extend the built-in `Extension` class. As this is forced by the language, should the project lose marks for doing so if there is no other way to achieve this functionality?

This would require further research and analysis of what should/should not be included in the results for each language. Currently, the tool is designed so that it can be built the same way in other languages and there are pros/cons which would need to be researched by becoming opinionated on a per-language basis.

Other potential future work involves improving the performance of the tool. Although no users complained about the speed in the survey, work could be done to improve the performance of the code which scans the source code uploaded. During development one issue that caused some projects to be scanned twice was noticed and fixed (thereby doubling the time it took to analyse them) but no work has actively been done profiling the code or looking for other bottlenecks. There are no doubt many ways performance of the tool could be improved.

The tool currently makes no effort to utilise multi-threading when analysing a project and is therefore limited by the CPU speed of a single core of the server on which it is running.

Currently most projects are analysed in under a minute, which is deemed reasonable by the developer, however there is room to improve the performance of the tool so that users are not waiting as long for results. For comparison, Scrutinizer takes 10-15 minutes to analyse a small project.

In addition, the tool is running on a medium specification VPS with 8 cores. If the tool became more popular, a queuing system would need to be introduced as only 8 projects can reasonably be analysed at once before performance suffers.

Finally, the tool currently produces a patch to automate fixes of two of the bad practices. This is proof-of-concept and in some cases the patch may not be complete. This is something which requires further work and automated fixes for the remaining bad practices need to be introduced.

## 6.12.5 Future improvements/Limitations - Chapter 6 - Evaluation

Section 6.5 which compares the metric produced as part of this research to other metrics could be extended in several ways:

1. A larger sample size of software for the comparison to other metrics. Rather than 20 projects, 100 or more could be analysed. SonarQube graded all 20 projects as A, with a larger sample size, projects may be graded in a larger range. The primary reason for this moderately sized sample is the 15-20 minutes each project took to scan using Scrutinizer. Increasing the sample size to 100 would require around 24 hours of time spend waiting for Scrutinizer to generate results.
2. Using different tools for other languages. The reason this is currently not possible is that the tool currently only supports PHP. Before this kind of evaluation can be completed, the metric will need to be implemented for additional languages.
3. The user evaluation in section 6.7 could be expanded and altered for different target audiences. It is noted, however, that gathering respondents was difficult and finding a meaningful number of respondents in each target audience may be challenging. Doing so would enable assessing the metric and tool as a learning tool in comparison to a business tool and determine which of the use-cases it is most useful.
4. Further results could be gathered by performing an additional user evaluation where users compare Insphpect's results to other tools. Ideally this could be done blindly by showing them, for example, Scrutinizer-CI's report and Insphpect's report and asking them which they prefer, which they believe more accurately reflects the quality and flexibility of the code that has been analysed. Additional feedback could be gathered on which aspects of each tool users find useful. This feedback could then be used to refine Insphpect and make it even more useful to users.

This type of comparative evaluation would provide useful feedback for further development of the tool and metric

## 6.13 Chapter review

This chapter outlined potential evaluation techniques and explained why the chosen methods were used. It includes data obtained from 75 real users and their feedback.

Overall user feedback was positive and the tool and metric were both useful to users of the website.





## 7. References

Abreu, F. (1995) The MOOD Metrics Set. *ECOOP'95 Workshop on Metrics* .

Adobe, A. (2013a) *OOP 101 Advice (please)* [online]. Available from: <https://forums.adobe.com/thread/1304387?start=0&tstart=0> [Accessed 7th January 2016]

Ahuja, K. (2015) *Are Annotations Bad?* [online]. Available from: <https://dzone.com/articles/are-annotations-bad> [Accessed 6th July 2016]

Aldrich, J. (2004) Selective Open Recursion: A Solution to the Fragile Base Class Problem. *Carnegie Mellon University* .

Alipour, G., Sangar, A., Mogaddam, M. (2016) ASPECT ORIENTED IMPLEMENTATION OF DESIGN PATTERNS USING METADATA. *Journal of Fundamental and Applied Sciences* 57, pp.66-75.

Analysis Tools, A. (n.d.) *Analysis Tools* [online]. Available from: <https://analysis-tools.dev/> [Accessed 20th May 2021]

Arendsen, A. (2007) *Setter injection versus constructor injection and the use of @Required* [online]. Available from: <http://spring.io/blog/2007/07/11/setter-injection-versus-constructor-injection-and-the-use-of-required/> [Accessed 28th December 2015]

Albert, A. (2013) *Why should we use dependency injection?* [online]. Available from: <http://www.javacreed.com/why-should-we-use-dependency-injection/> [Accessed 1st May 2018]

Atwood, J. (2007) *Rethinking Design Patterns* [online]. Available from: <https://blog.codinghorror.com/rethinking-design-patterns/> [Accessed 22nd August 2016]

Badu, K. (2008) *What's so evil about Singleton?* [online]. Available from: <http://www.sitepoint.com/forums/showthread.php?530917-What-s-so-evil-about-Singleton> [Accessed 7th January 2016]

Bansiya, J., Davis, C. (1997) Automated metrics and object-oriented development. *Dr. Dobb's Journal; San Mateo* 22, pp.42-48.

Bansiya, J., Davis, C. (2002) A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering* 28, pp.4-17.

Bell, J. (2013) *PHP Annotations Are a Horrible Idea* [online]. Available from: <http://theunraveler.com/blog/2012/php-annotations-are-a-horrible-idea> [Accessed 6th July 2016]

- Benharosh, J. (2015) *The singleton pattern in PHP* [online]. Available from: <http://phpenthusiast.com/blog/the-singleton-design-pattern-in-php> [Accessed 2nd August 2016]
- Bergmann, S. (2013) *de-legacy-fy* [online]. Available from: <https://github.com/sebastianbergmann/de-legacy-fy> [Accessed 2nd August 2016]
- Biberstein, M., Sreedhar, V., Zaks, A. (2002a) A Case For Sealing Classes In Java. *IBM Research* .
- Bloch, J. (2008) *Effective Java: Second Edition* ISBN: 978-0321356680. Addison-Wesley.
- Bracha, G. (2007) *Constructors Considered Harmful* [online]. Available from: <https://gbracha.blogspot.co.uk/2007/06/constructors-considered-harmful.html> [Accessed 2nd August 2016]
- Brandsma, C. (2009) *Observations on the 'if' statement* [online]. Available from: <http://elegantcode.com/2009/08/14/observations-on-the-if-statement/> [Accessed 5th September 2013]
- Brown, W., Malveau, R., McCormick, H., Mowbray, T. (1998) *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* ISBN: 0471197130. JOHN WILEY & SONS.
- Brown, W. (2013) *Why Singletons are "Bad Patterns"* [online]. Available from: <http://brollace.blogspot.co.uk/2013/04/why-singletons-are-bad-patterns.html> [Accessed 5th September 2013]
- Bryson, B. (2010) *A short history of nearly everything* ISBN: 9780552997041. London : Black Swan.
- Bryton, S., Brito e Abreu, F., Monteiro, M. (2002) Reducing Subjectivity in Code Smells Detection: Experimenting with the Long Method. *Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference. IEEE* .
- Bugayenko, Y. (2015) *Temporal Coupling Between Method Calls* [online]. Available from: <https://www.yegor256.com/2015/12/08/temporal-coupling-between-method-calls.html> [Accessed 6th July 2016]
- Bugayenko, Y. (2016) *Java Annotations Are a Big Mistake* [online]. Available from: <http://www.yegor256.com/2016/04/12/java-annotations-are-evil.html> [Accessed 6th July 2016]
- Buss, M. (2016) *Interfaces vs Inheritance in Swift* [online]. Available from: <https://mikebuss.com/2016/01/10/interfaces-vs-inheritance/> [Accessed 22nd September 2018]

- Butler, T. (2013) *Are Static Methods/Variables bad practice?* [online]. Available from: <https://r.je/static-methods-bad-practice.html> [Accessed 4th October 2015]
- Butler, T. (2013) *Constructor Injection vs Setter Injection* [online]. Available from: <https://r.je/constructor-injection-vs-setter-injection.html> [Accessed 4th October 2015]
- Butler, T. (2013) *PHP: Annotations are an Abomination* [online]. Available from: <https://r.je/php-annotations-are-an-abomination.html> [Accessed 6th July 2016]
- Butler, T. (2015) *Slutty Software is good software: Tight and loose coupling in OOP* [online]. Available from: <https://r.je/slutty-software-tight-and-loose-coupling.html> [Accessed 4th October 2015]
- Butler, T. (2017a) Seven deadly sins of software flexibility. *13th China Europe International Symposium Of Software Engineering Education* . University of Derby.
- Butler, T. (2019a) A Methodology for Performing Meta-analyses of Developers Attitudes Towards Programming Practices. *Intelligent Computing: Proceedings of the 2019 Computing Conference 2*, pp.948-954. Springer.
- van Dongen, J. (2014a) *The Fragile Base Class Problem* [online]. Available from: <http://www.web-brainz.co.uk/fragile> [Accessed 8th August 2016]
- Carneiro, F., Silva, G., Mara, L., Figueiredo, E., Sant'Anna, C., Garcia, A., Mendonça, M. (2010) Identifying code smells with multiple concern views. *Brazilian Symposium on Software Engineering (SBES)* 57, pp.128-137. IEEE.
- Caromel, D. (1993) Toward a method of object-oriented concurrent programming. *Communications of the ACM* , pp.90-102.
- Cherniavsky, J., Smith, C. (1991) On Weyuker's Axioms for Software Complexity Measures. *IEEE Transactions on software engineering* 17, pp.636-638.
- Chidamber, S., Kemerer, C. (2007) A metrics suite for object oriented design. *IEEE Transactions on software engineering* 20, pp.476-493.
- Mel, O., Nixon, P. (1998) Program Restructuring to Introduce Design Patterns. *ECOOP Workshop* , pp.79-80.
- Mathie, R., Frye, J., Fisher, P. (2015) Homeopathic Oscillocochinum® for preventing and treating influenza and influenza-like illness. *Cochrane Database System Rev* 12 .
- Cochrane, C. (n.d.) *Cochrane* [online]. Available from: <http://www.cochrane.org/> [Accessed 8th

August 2016]

Cosentino, N. (2013) *Singletons: Why Are They "Bad"?* [online]. Available from: <https://www.codeproject.com/articles/634723/singletons-why-are-they-bad> [Accessed 4th October 2015]

Crockford, D. (2006) *Global Domination* [online]. Available from: <http://www.yuiblog.com/blog/2006/06/01/global-domination/> [Accessed 5th September 2013]

Davis, P. (2007) *Annotations, the Good the Bad and the Ugly* [online]. Available from: <http://willcode4beer.blogspot.co.uk/2007/12/annotations-good-bad-and-ugly.html> [Accessed 6th July 2016]

Densmore, S. (2004) *Why Singletons Are Evil* [online]. Available from: <http://blogs.msdn.com/b/scottdensmore/archive/2004/05/25/140827.aspx> [Accessed 5th September 2013]

Deshapriya, R. (2011) *A Singleton Java class for MySQL DB connection* [online]. Available from: <http://rdeshapriya.com/a-singleton-java-class-for-mysql-db-connection/> [Accessed 2nd August 2016]

Transphorm, T. (n.d.a) *Dice Dependency Injection Container* [online]. Available from: <https://github.com/Level-2/Dice> [Accessed 12th May 2020]

Dohms, R. (2013) *Annotations in PHP: They Exist* [online]. Available from: <http://www.slideshare.net/rdohms/annotations-in-php-they-exist> [Accessed 2nd May 2016]

van Dongen, J. (2014) *Why composition is often better than inheritance* [online]. Available from: <http://joostdevblog.blogspot.co.uk/2014/07/why-composition-is-often-better-than.html> [Accessed 8th August 2016]

Durand, W. (2013) *From STUPID to SOLID Code!* [online]. Available from: <http://williamdurand.fr/2013/07/30/from-stupid-to-solid-code/> [Accessed 2nd August 2016]

Eberlei, B. (2013) *Traits are Static Access* [online]. Available from: [http://www.whitewashing.de/2013/04/12/traits\\_are\\_static\\_access.html](http://www.whitewashing.de/2013/04/12/traits_are_static_access.html) [Accessed 2nd August 2016]

ECMA International, E. (2017) *The JSON Data Interchange Syntax* [online]. Available from: <http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf> [Accessed 22nd March 2018]

Eden, A., Tom, M. (2006) Measuring software flexibility.. *IEE Software* 153, pp.133-126.

- Ericson, B. (1995) *Association vs Inheritance* [online]. Available from: <http://ice-web.cc.gatech.edu/ce21/1/static/JavaReview-RU/OOBasics/ooAssocVsInherit.html> [Accessed 3rd January 2016]
- Faulkner, L. (2003) Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers* volume 35, pp.379-383.
- Fernández, D. (2011) *Some Java "dependency injection" bad practices* [online]. Available from: <http://diegacho.blogspot.co.uk/2011/09/some-java-dependency-injection-bad.html> [Accessed 6th July 2016]
- Ferreira, G. (2013) *Best C Coding Practices – Global variables* [online]. Available from: <http://guilhermemacielferreira.com/2013/06/01/best-c-coding-practices-global-variables/> [Accessed 5th September 2013]
- Ferris, J. (2012) *Refactoring: Replace Conditional with Polymorphism* [online]. Available from: <http://robots.thoughtbot.com/post/31728620503/refactoring-replace-conditional-with-polymorphism> [Accessed 5th September 2013]
- Fontana, F., Mariani, E., Morniroli, A., Sormani, R., Tonello, A. (2011) An experience report on using code smells detection tools. *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)* , pp.450-457.
- Fontana, F., Ferme, V., Zanoni, M., Yamashita, A. (2015) Automatic metric thresholds derivation for code smell detection. *Proceedings of the Sixth international workshop on emerging trends in software metrics. IEEE Press* .
- Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D. (1999) *Refactoring: Improving the Design of Existing Code (Object Technology Series)* ISBN: 0201485672. Addison Wesley Longman, Inc..
- Fowler, M. (2002) *Patterns of Enterprise Application Architecture* ISBN: 0321127420. Addison Wesley.
- Fowler, M. (2004) *Inversion of Control Containers and the Dependency Injection pattern* [online]. Available from: <http://martinfowler.com/articles/injection.html> [Accessed 5th July 2016]
- Fowler, M. (2013a) *TellDontAsk* [online]. Available from: <http://martinfowler.com/bliki/TellDontAsk.html> [Accessed 2nd August 2016]
- Fowler, M. (2015) *Yagni* [online]. Available from: <http://martinfowler.com/bliki/Yagni.html> [Accessed 22nd August 2016]
- Funaro, M. (2009) *How OO Almost Destroyed My Business* [online]. Available from:

<https://www.advantexllc.com/blog/post.cfm/how-oo-almost-destroyed-my-business> [Accessed 22nd August 2016]

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. ISBN: 0201633612. Addison Wesley.

Geary, D. (2003) *Simply Singleton* [online]. Available from: <http://www.javaworld.com/article/2073352/core-java/simply-singleton.html> [Accessed 7th January 2016]

Gierke, O. (2013) *Why field injection is evil* [online]. Available from: <http://olivergierke.de/2013/11/why-field-injection-is-evil/> [Accessed 28th December 2015]

Gilstrap, J. (2010) *Java Annotations have Become Pixie Dust*. [online]. Available from: <http://viewfromthefringe.blogspot.co.uk/2010/02/java-annotations-have-become-pixie-dust.html> [Accessed 6th July 2016]

Goldacre, B. (2010) *Bad Science* ISBN: 978-0-00-724019-7. Fourth Estate.

Ivo, G., Mogado, P., T, G., R, M. (2009) An overview on the static code analysis approach in software development. *Faculdade de Engenharia da Universidade do Porto, Portugal*. .

Google Scholar, G. (n.d.) *Refactoring: Improving the Design of Existing Code* [online]. Available from: [https://scholar.google.com/scholar?cites=3793765135686184301&as\\_sdt=2005&scioldt=0,5&hl=en](https://scholar.google.com/scholar?cites=3793765135686184301&as_sdt=2005&scioldt=0,5&hl=en) [Accessed 5th September 2019]

Grady, R. (1994) Successfully applying software metrics.. *Computer* 27, pp.18-25.

Green, C. (2015) *Singleton Database Connection Class in PHP* [online]. Available from: <http://sundayepidemic.com/singleton-database-connection-class/> [Accessed 2nd August 2016]

Grimm, A. (2014a) *Demeter: It's not just a good idea. It's the law* [online]. Available from: <http://devblog.avdi.org/2011/07/05/demeter-its-not-just-a-good-idea-its-the-law/> [Accessed 2nd August 2016]

van Gorp, J., Bosch, J. (2001) Design, implementation and evolution of object oriented frameworks: concepts and guidelines. *Software- Practice and Experience* 31, pp.277-300.

Haack, P. (2009) *The Law of Demeter is not a dot counting exercise* [online]. Available from: <http://haacked.com/archive/2009/07/13/law-of-demeter-dot-counting.aspx> [Accessed 2nd August 2016]

Harrison, R., Counsell, S., Nithi, R. (1998) An Evaluation of the MOOD Set of Object-Oriented Software Metrics. *IEEE Transactions on Software Engineering* 24, pp.491-496.

Hart, S. (2011) *Why helper, singletons and utility classes are mostly bad* [online]. Available from: <http://smart421.wordpress.com/2011/08/31/why-helper-singletons-and-utility-classes-are-mostly-bad-2/> [Accessed 6th July 2016]

Hevery, M. (2008) *Flaw: Constructor Does Real Work* [online]. Available from: <http://misko.hevery.com/code-reviewers-guide/flaw-constructor-does-real-work/> [Accessed 4th October 2015]

Hevery, M. (2008) *Static Methods are Death to Testability* [online]. Available from: <http://misko.hevery.com/2008/12/15/static-methods-are-death-to-testability/> [Accessed 4th October 2015]

Hevery, M. (2008) *Brittle Global State & Singletons* [online]. Available from: <http://misko.hevery.com/code-reviewers-guide/flaw-brittle-global-state-singletons/> [Accessed 4th October 2015]

Hevery, M. (2008q) *Flaw: Digging into Collaborators* [online]. Available from: <http://misko.hevery.com/code-reviewers-guide/flaw-digging-into-collaborators/> [Accessed 4th October 2015]

Hevery, M. (2008) *Guide: Writing Testable Code* [online]. Available from: <http://misko.hevery.com/code-reviewers-guide/> [Accessed 4th October 2015]

Hevery, M. (2008) *Top 10 things which make your code hard to test* [online]. Available from: <http://misko.hevery.com/2008/07/30/top-10-things-which-make-your-code-hard-to-test/> [Accessed 4th October 2015]

Hevery, M. (2008) *Breaking the Law of Demeter is Like Looking for a Needle in the Haystack* [online]. Available from: <http://misko.hevery.com/2008/07/18/breaking-the-law-of-demeter-is-like-looking-for-a-needle-in-the-haystack/> [Accessed 4th October 2015]

Hevery, M. (2008) *Singletons are Pathological Liars* [online]. Available from: <http://misko.hevery.com/2008/08/17/singletons-are-pathological-liars/> [Accessed 4th October 2015]

Hevery, M. (2008?) *How to Think About the "new" Operator with Respect to Unit Testing* [online]. Available from: <http://misko.hevery.com/2008/07/08/how-to-think-about-the-new-operator/>



[Accessed 4th October 2015]

Hevery, M. (2008) *Code Reviewers Guide* [online]. Available from:  
<http://misko.hevery.com/code-reviewers-guide/> [Accessed 4th October 2015]

Hevery, M. (2009) *Constructor Injection vs. Setter Injection* [online]. Available from:  
<http://misko.hevery.com/2009/02/19/constructor-injection-vs-setter-injection/> [Accessed 4th October 2015]

Holub, A. (2010) *Why extends is evil* [online]. Available from:  
<http://www.javaworld.com/article/2073649/core-java/why-extends-is-evil.html> [Accessed 8th August 2016]

Hurn, S. (2014) *Favor Composition Over Inheritance* [online]. Available from:  
<https://codingdelight.com/2014/01/16/favor-composition-over-inheritance-part-1/> [Accessed 8th August 2016]

IBM, I. (2012) *Avoid modification of global and static variables* [online]. Available from:  
[http://pic.dhe.ibm.com/infocenter/idshelp/v1117/index.jsp?topic=%2Fcom.ibm.dapip.doc%2Fids\\_da pip\\_0673.htm](http://pic.dhe.ibm.com/infocenter/idshelp/v1117/index.jsp?topic=%2Fcom.ibm.dapip.doc%2Fids_da pip_0673.htm) [Accessed 5th September 2013]

Butler, T. (2020a) *Insphpect – Code review tool that scans PHP code for poor OO design* [online]. Available from: <https://news.ycombinator.com/item?id=22550991> [Accessed 20th July 2020]

Butler, T. (2020b) *Insphpect – Code review tool that scans PHP code for poor OO design* [online]. Available from: <https://www.phptoday.org/news/insphpect-new-code-analysis-tool> [Accessed 20th July 2020]

Butler, T. (2020) *Insphpect - New code analysis tool* [online]. Available from:  
[https://www.reddit.com/r/PHP/comments/fhoday/insphpect\\_new\\_code\\_analyis\\_tool\\_scans\\_your\\_code/](https://www.reddit.com/r/PHP/comments/fhoday/insphpect_new_code_analyis_tool_scans_your_code/) [Accessed 20th July 2020]

Butler, T. (2020c) <https://www.sitepoint.com/how-to-ensure-flexible-reusable-php-code-with-insphpect> [online]. Available from:  
<https://www.sitepoint.com/how-to-ensure-flexible-reusable-php-code-with-insphpect> [Accessed 20th July 2020]

Twitter, T. (2020) *Search results for Insphpect* [online]. Available from:  
[https://twitter.com/search?q=insphpect&src=typed\\_query&f=live](https://twitter.com/search?q=insphpect&src=typed_query&f=live) [Accessed 1st August 2020]

Jadad, A., Moore, A., Carroll, D., Jenkinson, C. (1996) Assessing the quality of reports of randomized clinical trials: Is blinding necessary?. *Controlled Clinical Trials* 17(1), pp.1-12. ELSEVIER.

James, G. (1987) *The Tao of Programming* ISBN: 978-0931137075. Info Books.

Johansson, M. (2015) *Composition over Inheritance* [online]. Available from: <https://medium.com/humans-create-software/composition-over-inheritance-cb6f88070205> [Accessed 8th August 2016]

Johnson, R., Foote, B. (1988) Desinging Reusable Classes. *Journal of Object Oriented Programming* , pp.22-35.

Gutha, S. (2015) *The JSON Data Interchange Standard* [online]. Available from: <https://www.json.org/> [Accessed 22nd March 2018]

Judis, S. (2017a) *The global object in JavaScript: a matter of platforms, unreadable code and not breaking the internet* [online]. Available from: <https://www.contentful.com/blog/2017/01/17/the-global-object-in-javascript/> [Accessed 22nd August 2018]

Kainulainen, P. (2013) *Why I Changed My Mind About Field Injection?* [online]. Available from: <http://www.petrikainulainen.net/software-development/design/why-i-changed-my-mind-about-field-injection/> [Accessed 28th December 2015]

Kernighan, B. (1999) *The Practice of Programming* ISBN: 978-0201615869. Addison Wesley.

Kitcchenham, B., Pfleeger, S., Fenton, N. (1995) Towards a Framework for Software Measurement Validation. *IEEE Transactions on Software Engineering* 21, pp.929-944.

Kofler, P. (2012) *Why Singletons Are Evil* [online]. Available from: <http://blog.code-cop.org/2012/01/why-singletons-are-evil.html> [Accessed 5th September 2013]

Koopman, P. (2010a) *Better Embedded System Software* ISBN: 978-0-9844490-0-2. Drumnadrochit Education LLC.

Kaur, A., Singh, S. (2018) Detecting Software Bad Smells from Software Design Patterns using Machine Learning Algorithms. *International Journal of Applied Engineering Research* , pp.10005-10010.

Lewis, M. (2013) *PHP Annotations are a Bad Idea* [online]. Available from: [https://www.marclewis.com/2013/10/25/php\\_annotations\\_are\\_a\\_bad\\_idea/](https://www.marclewis.com/2013/10/25/php_annotations_are_a_bad_idea/) [Accessed 6th July 2016]

Liu, H., Cai, C., Zu, C. (2011) An object-oriented serial implementation of a DSMC simulation package. *Journal of Fundamental and Applied Sciences* 8, pp.816-825.

Liu, H., Ma, Z., Shao, W., Niu, Z. (2012) Schedule of bad smell detection and resolution: A new way to save effort. *IEEE Transactions on Software Engineering* 38, pp.220-235.

Macefield, R. (2009) How to specify the participant group size for usability studies: a practitioner's guide. *Journal of Usability Studies* 5, pp.34-35.

Martin, R. (2000) *Design Principles and Design Patterns* [online]. Available from: [http://www.objectmentor.com/resources/articles/Principles\\_and\\_Patterns.pdf](http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf) [Accessed 7th January 2016]

Martin, R. (2003) *SRP: The Single Responsibility Principle* [online]. Available from: [https://drive.google.com/file/d/0ByOwmqah\\_nuGNHEtcU5OekdDMkk/view](https://drive.google.com/file/d/0ByOwmqah_nuGNHEtcU5OekdDMkk/view) [Accessed 8th August 2016]

Martin, R. (2008a) *Clean Code: A Handbook of Agile Software Craftsmanship* ISBN: 978-0132350884. Prentice Hall.

Martin, R. (2011) *Agile Software Development, Principles, Patterns, and Practices* ISBN: 978-0132760584. Pearson.

Martin, R. (2014) *SingletonVsJustCreateOne* [online]. Available from: <http://butunclebob.com/ArticleS.UncleBob.SingletonVsJustCreateOne> [Accessed 2nd August 2016]

Martin, R. (2014) *The Single Responsibility Principle* [online]. Available from: <https://8thlight.com/blog/uncle-bob/2014/05/08/SingleResponsibilityPrinciple.html> [Accessed 2nd August 2016]

McCabe, T. (1976) A metrics suite for object oriented design. *IEEE Transactions on software engineering* 4, pp.308-320.

McConnell, S. (2004) *Code Complete: A Practical Handbook of Software Construction* ISBN: 0735619670. Microsoft Press.

Meyer, B. (1988a) Bidding farewell to globals. *JOOP(Journal of Object-Oriented Programming)* , pp.73-77.

Mindra, D. (2014) *Dependency Injection and Abstractions* [online]. Available from: <http://blogs.unity3d.com/2014/05/07/dependency-injection-and-abstractions/> [Accessed 2nd August 2016]

MSDN, M. (2013) *Design Warnings* [online]. Available from: [https://msdn.microsoft.com/en-us/library/bb429476\(v=vs.80\).aspx](https://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx) [Accessed 5th September 2013]

- Neelamegam, C., Punithavalli, M. (2009) A survey-object oriented quality metrics. *Journal of Object Oriented Programming* , pp.183-186.
- Neeraj, S., Sinha, J., Jackson, D. (2005) Using Dependency Models to Manage Complex Software Architecture.. *OOPSLA '05 Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages and applications* .
- Nguyen, V., Deeds-Rubin, S., Tan, T., Boehm, B. (2007) A SLOC Counting Standard. *Cocoma ii forum* 2007, pp.1-16.
- Knack-Nielsen, T. (2008) *What's so bad about the Singleton?* [online]. Available from: <http://www.sitepoint.com/whats-so-bad-about-the-singleton/> [Accessed 7th January 2016]
- Noback, M. (2013) *Dependency Injection Smells* [online]. Available from: <http://php-and-symfony.matthiasnoback.nl/2013/01/dependency-injection-smells/> [Accessed 25th July 2016]
- Nordmann, K. (2011) *static considered harmful* [online]. Available from: [https://kore-nordmann.de/blog/0103\\_static\\_considered\\_harmful.html](https://kore-nordmann.de/blog/0103_static_considered_harmful.html) [Accessed 6th July 2016]
- Olivo, S., Macedo, L., Caroline, I., Fuentes, J., Magee, D. (2008) Scales to assess the quality of randomized controlled trials: a systematic review.(Research Report). *Physical Therapy* 88(2), pp.156.
- Oracle, O. (2014) *JSR 175: A Metadata Facility for the Java™ Programming Language* [online]. Available from: <https://www.jcp.org/en/jsr/detail?id=175#2> [Accessed 22nd November 2018]
- Oracle, O. (2010) *The @Path Annotation and URI Path Templates* [online]. Available from: <https://docs.oracle.com/cd/E19798-01/821-1841/gjnpw/> [Accessed 6th July 2016]
- Oracle, O. (2011) *Annotation Type Inject* [online]. Available from: <http://docs.oracle.com/javaee/6/api/javax/inject/Inject.html> [Accessed 6th July 2016]
- Otander, J. (2015) *Mixins over Inheritance* [online]. Available from: <http://alisoftware.github.io/swift/protocol/2015/11/08/mixins-over-inheritance/> [Accessed 8th August 2016]
- Packagist, P. (n.d.a) *Packagist the PHP Package Repository* [online]. Available from: <https://packagist.org/> [Accessed 22nd June 2020]
- Paul, J. (2012) *Difference between Setter vs Constructor Injection in Spring* [online]. Available from: <http://javarevisited.blogspot.co.uk/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html> [Accessed 28th December 2015]

Paul, J. (2013) *5 Reasons to Use Composition over Inheritance in Java and OOP* [online]. Available from: <http://javarevisited.blogspot.com/2013/06/why-favor-composition-over-inheritance-java-oops-design.html> [Accessed 8th August 2016]

Peterson, J. (2008) *An example of how not to use Java annotations* [online]. Available from: <http://jeffvssoftware.blogspot.co.uk/2009/01/example-of-how-not-to-use-java.html> [Accessed 6th July 2016]

PHP Mess Detector, P. (n.d.a) *PMD Java Mess Detector* [online]. Available from: <https://pmd.github.io/> [Accessed 3rd March 2016]

PHPUnit, P. (n.d.a) *Code Coverage Analysis* [online]. Available from: <https://phpunit.readthedocs.io/en/9.2/code-coverage-analysis.html> [Accessed 12th May 2020]

PHP Mess Detector, P. (n.d.) *PMD Java Mess Detector* [online]. Available from: <https://phpmd.org/> [Accessed 3rd March 2016]

Popov, N. (2014) *Don't be STUPID: GRASP SOLID!* [online]. Available from: <https://nikic.github.io/2011/12/27/Dont-be-STUPID-GRASP-SOLID.html> [Accessed 2nd August 2016]

Qwant, Q. (n.d.) *Qwant search engine* [online]. Available from: <https://www.qwant.com/> [Accessed 28th May 2018]

Radford, M. (2003) Singleton - the anti-pattern. *Overload* 57. ACCU.

RadialBar, R. (n.d.) *Code Coverage Analysis* [online]. Available from: <https://github.com/AZbang/RadialBar> [Accessed 12th May 2020]

J., R. (2001) *Use your singletons wisely* [online]. Available from: <https://www.ibm.com/developerworks/library/co-single/> [Accessed 5th September 2013]

Raymond, E. (2003) *The Art of Unix Programming* ISBN: 978-0131429017. Addison Wesley.

Reddit, R. (2013) *[General] Singletons* [online]. Available from: [https://www.reddit.com/r/csELI5/comments/1qwey4/eli5general\\_singletons/](https://www.reddit.com/r/csELI5/comments/1qwey4/eli5general_singletons/) [Accessed 7th January 2016]

Reigler, G. (2014) *An Annotation Nightmare* [online]. Available from: <http://www.javacodegeeks.com/2014/01/an-annotation-nightmare.html> [Accessed 6th July 2016]

Riley, R., Snell, K., Harrell, F., Martin, G., Reitsma, J. (2020) Calculating the sample size required for developing a clinical prediction model. *BMJ* 368.

Rogers, P. (2001) *Encapsulation is not information hiding* [online]. Available from: <http://www.javaworld.com/article/2075271/core-java/encapsulation-is-not-information-hiding.html> [Accessed 9th July 2016]

Ronacher, A. (2009) *Singletons and their problems in Python* [online]. Available from: <http://lucumr.pocoo.org/2009/7/24/singletons-and-their-problems-in-python/> [Accessed 5th September 2013]

Rybak, M. (2013) *Why Static Code is Bad* [online]. Available from: <https://objcsharp.wordpress.com/2013/07/08/why-static-code-is-bad/> [Accessed 2nd August 2016]

Sabir, F., Palma, F., Rasool, G., Guéhéneuc, Y., Moha, N. (2018) A systematic literature review on the detection of smells and their evolution in object oriented and service oriented systems. *Software: Practice and Experience* 79(1), pp.3-39.

Sanders, B. (2013) *No Time For OOP or Design Patterns* [online]. Available from: <http://www.php5dp.com/no-time-for-oop-or-design-patterns/> [Accessed 22nd August 2016]

Sayfan, M. (n.d.) *Avoid Global Variables, Environment Variables, and Singletons* [online]. Available from: <https://sites.google.com/site/michaelsafyan/software-engineering/avoid-global-variables-environment-variables-and-singletons> [Accessed 5th September 2013]

Schindler, R. (2012) *PHP Constructor Best Practices And The Prototype Pattern* [online]. Available from: <http://ralphschindler.com/2012/03/09/php-constructor-best-practices-and-the-prototype-pattern> [Accessed 4th October 2015]

Schwarz, N., Lungu, M., Nierstraz, O. (2011) Sues: Better Class Responsibilities through Language-Based Dependency Injection. *Proceedings of the 49th international conference on Objects, models, components, patterns* .

Scrutinizer-CI, S. (n.d.) *Scrutinizer* [online]. Available from: <https://scrutinizer-ci.com> [Accessed 5th September 2016]

Seeman, M. (2010) *Service Locator is an Anti-Pattern* [online]. Available from: <http://blog.ploeh.dk/2010/02/03/ServiceLocatorisanAnti-Pattern/> [Accessed 5th September 2013]

Seeman, M. (2011) *Design Smell: Temporal Coupling* [online]. Available from: <https://blog.ploeh.dk/2011/05/24/DesignSmellTemporalCoupling/> [Accessed 5th September 2013]

Seeman, M. (2015) *Service Locator violates Encapsulation* [online]. Available from: <http://blog.ploeh.dk/2015/10/26/service-locator-violates-encapsulation/> [Accessed 7th January

2016]

Seguy, D. (2014) *Static analysis tools for PHP* [online]. Available from:  
<https://github.com/exakat/php-static-analysis-tools> [Accessed 4th October 2018]

SensioLabs Insight, S. (n.d.) *SensioLabs Insight* [online]. Available from:  
<https://insight.symfony.com/> [Accessed 3rd March 2016]

Smith, S. (2012) *Explicit Dependencies Principle* [online]. Available from:  
<http://deviq.com/explicit-dependencies-principle/> [Accessed 2nd August 2016]

SonarQube, S. (n.d.) *Maintainability = Productivity* [online]. Available from:  
<https://www.sonarqube.org/features/maintainability/> [Accessed 1st August 2020]

Sonmez, J. (2010) *Static Methods Will Shock You* [online]. Available from:  
<http://simpleprogrammer.com/2010/01/29/static-methods-will-shock-you/> [Accessed 6th July 2016]

Sosnoski, D. (2005) *Classworking toolkit: Annotations vs. configuration files* [online]. Available from:  
<http://www.ibm.com/developerworks/library/j-cwt08025/> [Accessed 6th July 2016]

Spring, S. (2018) *Spring Framework Annotations* [online]. Available from:  
<https://springframework.guru/spring-framework-annotations/> [Accessed 22nd November 2018]

Kegel, H., Steimann, F. (2008) Systematically Refactoring Inheritance to Delegation in JAVA.  
*Proceedings of the 30th international conference on Software engineering* . ACM.

Stripe, S. (2018) *The Developer Coefficient* [online]. Available from:  
<https://stripe.com/files/reports/the-developer-coefficient.pdf> [Accessed 28th September 2018]

Summit, S. (1997) *Visibility and Lifetime (Global Variables, etc.)* [online]. Available from:  
<https://www.eskimo.com/~scs/cclass/notes/sx4b.html> [Accessed 25th July 2016]

Sumpton, B. (2010) *Inheritance is evil, and must be destroyed* [online]. Available from:  
<http://blog.berniesumption.com/software/inheritance-is-evil-and-must-be-destroyed/> [Accessed 8th August 2016]

Svennerberg, G. (2012) *Global Variables Are Evil* [online]. Available from:  
<http://codecraftingblueprints.com/global-variables-are-evil/> [Accessed 5th September 2013]

Symfony Framework, S. (nd) *@Route and @Method* [online]. Available from:  
<http://symfony.com/doc/2.0/bundles/SensioFrameworkExtraBundle/annotations/routing.html>  
[Accessed 6th July 2016]

Torchiano, M. (2014) *Are Suppress Warning Annotations Bad?* [online]. Available from: <https://mtorchiano.wordpress.com/2014/03/31/are-suppress-warning-annotations-bad/> [Accessed 6th July 2016]

Transphorm, T. (n.d.) *Transphorm Style Sheets* [online]. Available from: <https://github.com/Level-2/Transphorm> [Accessed 12th May 2020]

TutsPlus, T. (n.d.) *What Is Composer for PHP and How to Install It* [online]. Available from: <https://code.tutsplus.com/tutorials/what-is-composer-for-php-and-how-to-install-it--cms-35160> [Accessed 18th June 2020]

Vogt, B. (2008) *Alternatives for the singleton pattern?* [online]. Available from: <https://stackoverflow.com/questions/3171291/alternatives-for-the-singleton-pattern> [Accessed 2nd December 2017]

Waddicor, A. (2014) *Understanding dependency injection* [online]. Available from: <https://blogs.endjin.com/2014/04/understanding-dependency-injection/> [Accessed 28th December 2015]

Walls, C. (2008) *When Good Annotations Go Bad* [online]. Available from: <http://java.dzone.com/articles/when-good-annotations-go-bad> [Accessed 6th July 2016]

Weaver, R. (2010) *Static methods vs singletons: choose neither* [online]. Available from: <http://www.phparch.com/2010/03/static-methods-vs-singletons-choose-neither/> [Accessed 5th September 2013]

Weiskotten, J. (2006) *Dependency Injection & Testable Objects* [online]. Available from: <http://www.drdoobs.com/tools/dependency-injection-testable-objects/185300375> [Accessed 3rd January 2016]

Wenger, P. (1995) Interaction as a basis for empirical computer science. *ACM Computing Surveys (CSUR)* 27(1), pp.45-48.

Weyuker, E. (1988) Evaluating software complexity measures,. *IEEE Transactions on software engineering* 14, pp.1357-1365.

Wulf, W., Shaw, M. (1973) Global variables considered harmful. *ACM SIGPLAN Notices* , pp.28-34.

Yaiser, M. (2011) *Object-oriented programming concepts: Objects and classes* [online]. Available from: <http://www.adobe.com/devnet/actionscript/learning/oop-concepts/objects-and-classes.html> [Accessed 25th July 2016]



Yamashita, A., Moonen, L. (2013) Exploring the impact of inter-smell relations on software maintainability: An empirical study.. *Proceedings of the 2013 International Conference on Software Engineering. IEEE Press* .

Yegge, S. (2004) *Singleton Considered Stupid* [online]. Available from: <https://sites.google.com/site/steveyegge2/singleton-considered-stupid> [Accessed 5th September 2013]

Zakas, N. (2006) *Global Variables Are Evil* [online]. Available from: <http://www.nczonline.net/blog/2006/06/05/global-variables-are-evil/> [Accessed 5th September 2013]

# 8. Appendices

## Appendix I: Markdown extensions

Describing the bad practices in text required several extensions to the Markdown format being used. These are outlined below. These additions were made to allow dynamic sections to appear in the document and the document to be rendered in different ways.

### 8.1.1 References

References are generated on the fly so that they can be embedded in any format necessary. The Markdown format was extended to allow the code:

```
[ref:reference-id]
```

#### Quotation style references

```
[ref:butler-2017]
```

In the rendered document this can be replaced with a reference like [12] or (Butler, 2017).

In addition for quotes and other attributions, it also supports the following:

```
[ref-name:butler-2012]
```

This displays the reference in a quotation style. For example Butler (2017)

#### Including titles

Reference titles can also be included:

```
[ref-title:butler-2012]
```

Will generate a reference in the format: Seven Deadly Sins of Software Flexibility (Butler, 2017).

The exact reference formats can be tweaked for targeting conferences and journals with different referencing styles.

The details about the references, author names, journal names, dates, etc are stored in an external file in JSON format.

## references.json format

A file called references.json was used to store all the references used by the project. Each reference has a unique ID which is the key in the root level of the JSON object.

A sample reference looks like this:

```
"butler-2013": { // unique identifier
  "author": ["Tom", "Butler"], //Author names as an array
  "year": "2013", //Publication year
  "title": "PHP: Annotations are an Abomination", //Publication title
  "online": { // "online", "book", "journal"
    "url":
"https://r.je/php-annotations-are-an-abomination.html",
    "accessed": "2016-07-06"
  }
}
```

Different types of work can also be referenced by changing the online key to book, or journal. For example:

```
{
  "raymond-2003": {
    "author": ["Eric", "Raymond"],
    "year": "2003",
    "title": "The Art of Unix Programming",
    "book": {
      "isbn": "978-0131429017",
      "publisher": "Addison Wesley",
```

```

        "page": "20"
    }
},
"wulf-1973": {
    "author": [["William", "Wulf"], ["Mary", "Shaw"]],
    "year": "1973",
    "title": "Global variables considered harmful",
    "journal": {
        "month": "02",
        "year": "1973",
        "title": "ACM SIGPLAN Notices",
        "Volume": "8",
        "Number": "2",
        "pages": "28-34"
    }
}
}

```

The final document uses the reference ID to generate the complete the in-text reference and places the full references in the references section at the end of the document.

### Advantages of this reference approach

This approach has several advantages over manually embedding the references in the document:

1. The reference style can be changed easily
2. The reference list at the end of the document can be generated for a subset of the document, e.g. a single chapter. The references used by that single chapter can be included in the reference list without manually needing to cut and paste the references around.
3. Missing references can be detected during the compilation process.

### 8.1.2 Example code

For the bad practice sections and additional Markdown extension was added to allow dynamically embedding code examples in the text.

Examples are stored in each bad practice and the compiler has a language set before the Markdown files are loaded. When the Markdown compiler encounters code such as:

```
[example:1]
```

If the code format is set to *Java* it loads the contents of `examples/java/1.java` from the current bad practice's directory. If the code format is set to *PHP* it loads `examples/php/1.php`. This method allows generating the documentation for different programming languages and new programming languages can be easily added by creating the relevant directory.

**Advantages of including sample code in this format.**

1. The languages supported can be easily extended.
2. The same document can be generated with code examples from different languages without needing to re-write the document.

## Appendix II. Raw JSON files describing bad practices

Below are the raw json files used to describe each practice as developed in chapter 1.

### 8.2.3 Service Locator

```
{
  "name": "Service Locator",
  "severity": "4",
  "categories": ["Unnecessary Coupling", "Broken Single Responsibility Principle", "Broken Encapsulation", "Broken Law of Demeter"],
  "references": [
    "hevery-2008a", "hevery-2009", "bohlin-2010", "butler-2015",
    "johnson-1988", "waddicor-2014", "seeman-2015"
  ]
}
```

### 8.2.4 Singleton

```
{
  "name": "Singleton",
  "severity": "5",
  "categories": ["Broken Encapsulation", "Action at a Distance", "Global State", "Tight Coupling", "Broken Single Responsibility Principle"],
  "references": [
    "rainsberger-2001", "densmore-2004", "radford-2003",
    "yegge-2004", "ronacher-2009", "brown-2013", "kofler-2012", "weaver-2010",
    "reddit-2013", "badu-2008", "nielsen-2008", "geary-2003", "hart-2011",
    "nordmann-2011", "sonmez-2010", "benharosh-2015", "deshapriya-2011",
    "durand-2013", "martin-2014", "hevery-2008c", "hevery-2008f",
    "hevery-2008h", "sayfan-nd"
  ]
}
```

### 8.2.7 Object not initliased after constructor finishes (`initialize` and `set` methods)

```
{
  "name": "Object not initliased after constructor finishes"
```

```
(`initialize` and `set` methods)`",
  "severity": "3",
  "categories": ["Broken Encapsulation", "Action at a Distance",
"Temporal Coupling"],
  "references": [
    "hevery-2009", "butler-2013", "schindler-2012",
"muhammad-2013", "gierke-2013", "arendsen-2007", "kainulainen-2013",
"paul-2012", "fowler-2004"
  ]
}
```

### 8.2.10 Annotations for configuration

```
{
  "name": "Annotations for configuration",
  "severity": "4",
  "categories": ["Broken Encapsulation","Broken Single Responsibility
Principle",
    "Action At A Distance", "Unnecessary Coupling"],
  "references": [
    "butler-2013a", "bugayenko-2016", "ahuja-2015",
"uhrig-2015", "davis-2007", "lewis-2013", "sosnoski-2005", "walls-2008",
"peterson-2009", "gilstrap-2010", "fernandez-2011", "bell-2013",
"riegler-2014", "torchiano-2014"
  ]
}
```

### 8.2.11 Use of static methods

```
{
  "name": "Use of static methods",
  "severity": "4",
  "categories": ["Tight Coupling", "Broken Encapsulation",
  "Unclear Dependencies", "Single Responsibility Principle"],
  "extra": "they break SRP because the class which contains them must
have multiple responsibilities.",
  "references": [
```

```

        "cinneide-1999", "neeraj-2005", "bracha-2007",
"hevery-2008b", "sonmez-2010", "nordmann-2011", "schwarz-2011",
        "butler-2012", "smith-2012", "rybak-2013", "eberlei-2013",
"bergmann-2014", "mindra-2014"
    ]
}

```

### 8.2.12 Using `new` in constructor

```

{
    "name": "Using `new` in constructor",
    "severity": "3",
    "categories": ["Tight Coupling", "Broken Encapsulation", "Broken
Single Responsibility Principle"],
    "references": [
        "hevery-2008a", "hevery-2009", "bohlin-2010", "butler-2015",
"johnson-1988", "waddicor-2014"
    ]
}

```

### 8.2.13 Inheritance

```

{
    "name": "Inheritance",
    "severity": "3",
    "categories": ["Tight Coupling", "Broken Encapsulation", "Broken
Single Responsibility Principle"],
    "references": [
        "ericson-2014", "supton-2010", "hurn-2014", "dongen-2014",
"otander-2015", "paul-2013", "johansson-2015", "srinivasan-2008", "buss-2016"
    ]
}

```

### 8.2.14 Global/Static variables

```

{
    "name": "Global/Static variables",
    "severity": "5",
    "categories": ["Tight Coupling", "Broken Encapsulation", "Broken
Single Responsibility Principle",

```



```
    "Action at a Distance", "Global State"],  
    "references": [  
        "radford-2003", "densmore-2004", "yegge-2004",  
"crockford-2006", "zakas-2006", "hevery-2008g",  
        "hevery-2008c", "ronacher-2009", "weaver-2010", "hart-2011",  
        "nordmann-2011", "butler-2012", "ibm-2012", "kofler-2012",  
"svennerberg-2012", "ferreira-2013", "sayfan-nd"  
    ]  
}
```

## **Appendix III. Full explanations of bad practices**

Below are the full explanations of why each practice is bad. These explanations can be referenced in future work or embedded in future tools using this research.

---

### 8.3.1 Service Locator

A service locator is a specific common example of a Law Of Demeter violation (Hevery, 2008). This causes limited flexibility and code that is difficult to test. The Service Locator pattern looks like this:

```
class UserModel {  
  
    private serviceLocator;  
  
    public Collaborator(ServiceLocator serviceLocator) {  
        this.serviceLocator = serviceLocator;  
    }  
  
    public void save(User user) {  
        if (user.isValid()) {  
            this.serviceLocator.resolve("UserRepository").save(user);  
        }  
    }  
}
```

**Figure 8.1:** Service locator example 1

Here the `UserModel` class requires a `ServiceLocator` instance as a dependency. The author of the test must construct a valid `ServiceLocator` instance and ensure it is populated with any dependency the code being tested requires. This causes a brittle test, if the `save` method changes and requires extra dependencies from the service locator then the test will break unless the service locator is tested as part of the test.

The `ServiceLocator` itself cannot be mocked easily, as the service locator will need to be set up to return a mock `UserRepository` instance. This will cause broken encapsulation: The author of the test must be aware of how to configure the `serviceLocator` and know which services to include, the only way to do this is to look at the code for the class being tested and set up the mock service locator accordingly. By contrast, when injecting the specific instances that are needed, a complete list of required services should be visible in the constructor's method header (and can be picked up by IDE code completion tools).

*there's no reason to ever use a Service Locator. There's always a better alternative that*

*involves proper inversion of control.*

---

Seeman (2015)

### **Other problems**

Beyond the extra coupling introduced by the Law Of Demeter violation, the service locator by its nature exposes its dependencies to the outside world, breaking encapsulation (Seeman, 2015).

Summary of issues:

- Ambiguous portability (Service locator and everything provided to it must be present in a second project using the class)
  - Breaks law of Demeter
  - Unclear dependencies
  - Action at a distance (changes in the service locator affect any class that depends on it)
-

## 8.3.2 Singleton

*for it is true that global variables are often demonised and more recently the Singleton has befallen the same fate.*

---

Knack-Nielsen (2008)

The singleton has become regarded as an anti-pattern by most developers. This is for several reasons that don't apply to many other bad practices:

- The singleton was one of the patterns mentioned in two very popular and highly referenced books: *Design Patterns: Elements of Reusable Object-Oriented Software*. (Gamma et al, 1994) and *Patterns of Enterprise Application Architecture* (Fowler, 2002). This caused widespread knowledge of the pattern and it took some time until the issues it causes were documented.
- The pattern was given a formal name and is easy to identify
- The pattern has been labeled bad practice for over a decade (Densmore, 2004)
- The problems it causes are severe compared to other more subtle patterns so developers much more quickly identify one of the many issues it introduces.

Like *Global Variables*, because of abundant use, the problems associated with the pattern are very well documented.

Densmore (2004), Hevery (2008), Durand (2013) and Martin (2014) all touch on the same root problems with singletons.

### Hidden dependencies and Tight coupling

Hevery (2008) calls singleton's "Pathological Liars" because someone looking at the class cannot see its true dependencies. There is no way for a developer looking at the code to see what object is actually being used. Worse still, it's not possible for them to change it. Like all static methods, singletons introduce tight coupling (Densmore, 2004; Durand, 2013) as there is no way to substitute the class being used at runtime. Because the call is static, e.g.

`singleton::getInstance()` the object returned by the `getInstance()` call will always be used, making it very difficult to mock, and therefore very difficult to test in isolation.

Any class who uses the singleton (contains the line `singleton::getInstance()`) has an

ambiguous API. When using dependency injection, it's clear from the class API what, if any, collaborators it has. A developer can look at the arguments for the methods (usually the constructor) and quickly identify the dependencies of a class. When using a singleton, this isn't the case. When using singletons the only reference to the class is in one or more of the methods, there is no way to tell that there is a dependency by looking at the API alone. This doesn't directly affect flexibility, however it does make portability difficult, you cannot just move a class between projects because it's difficult to tell whether you also need to move the singleton class it relies on.

## Single responsibility principle

Singletons break the *Single Responsibility Principle*. Robert C. Martin, who coined the term *Single Responsibility Principle* (Martin, 2011) defines it as:

*a class should have one and only one responsibility.*

---

Singletons will always break this, because they have at least two responsibilities:

1. Whatever the class' job is (for example interacting with a database)
2. Ensuring only one instance can be created

Although the *Single Responsibility Principle* doesn't have a direct impact on flexibility in this instance it does introduce an obvious limitation. If only one instance of a class can exist, this is a direct limitation on flexibility. For example a common usage for singletons is a database access class (Benharosh, 2015; Deshapriya, 2011; Green, 2015). This ensures that only one database connection is possible throughout the application. There are practical reasons for this, performance and easy access to the relevant class. However, there is an obvious problem: What if a new requirement is introduced and now you need to connect to a second database for backup purposes or to retrieve some data? If a singleton was used, you must write another class to do this, or make changes to the class to handle multiple connections. Instead, if the singleton pattern was not used in the first place, multiple database connections could be created without an issue.

## Global state

A singleton is global state, it is available everywhere and has the same set of problems that affect global variables (Hevery, 2008).

## Law of Demeter

Singletons break the Law of Demeter (Hevery, 2008) because you have to go via the static method to access the true dependency.



### 8.3.3 Object not initialised after constructor finishes (`initialize` and `set` methods)

Once the constructor has run, the object should be 100% configured. Several common bad practices have emerged which go against this.

#### 1. initialize methods

These are often included to do further construction after the constructor has run. An example is included in figure 8.2:

```
public class House {
    private int number;
    private String streetName;
    private Kitchen kitchen;
    private Bedroom;

    public House(int number, String streetName) {
        this.number = number;
        this.streetName = streetName;
    }

    public void initialize() {
        this.kitchen = new Kitchen();
        this.bedroom = new Bedroom();
    }

    public String displayHouse() {
        return "the house has a kitchen with an area of " + kitchen.getWi
        dth() + " by " + kitchen.getLength();
    }
}
```

**Figure 8.2:** Initialize methods

This is sometimes so that the object can be initialized in different ways or alternatively so that performance heavy tasks (e.g. connecting to a database) can be avoided until they're needed.



## 2. Setter injection

Setter injection is also used for this purpose. Either to keep the constructor clean or to reduce the number of constructor arguments.

```
public class House {
    private int number;
    private String streetName;
    private Kitchen kitchen;
    private Bedroom;

    public House(int number, String streetName) {
        this.number = number;
        this.streetName = streetName;
    }

    public void setKitchen(Kitchen kitchen) {
        this.kitchen = kitchen;
    }

    public String displayHouse() {
        return "the house has a kitchen with an area of " + kitchen.getWi
        dth() + " by " + kitchen.getLength();
    }
}
```

**Figure 8.3:** Setter injection

In figure 8.3, for both `initialize` methods and setter injection, these minor advantages come at a huge cost in program stability and code flexibility. When using setter injection, methods on the object must be called in a specific order for it to work causing ambiguity and the internal state of the object creates action-at-a-distance. The code shown in figure 8.4 will work:

```
House house = new House();

house.setKitchen(new Kitchen());
```

```
System.out.println(house.display());
```

**Figure 8.4:** Setter injection problem demonstration (a)

However, reversing the order of arguments will break the code as demonstrated in figure 8.5:

```
House house = new House();  
  
//If the display method depends on the kitchen being set this will crash  
System.out.println(house.display());  
  
house.setKitchen(new Kitchen);
```

**Figure 8.5:** Setter injection problem demonstration (b)

This is because the `displayHouse` method requires a valid `Kitchen` instance and it may or may not be set. In this example it's easy to see whether or not the `setKitchen` method has been called or not. However, given the following code:

```
public void processHouse(House house) {  
    System.out.println(house.display());  
}
```

**Figure 8.6:** Setter injection problem demonstration (c)

In figure 8.6, the `processHouse` method may be given an incomplete house instance. It will crash if the setter was never called before the house instance was passed in to the `processHouse` method. The author of the `processHouse` method needs to be aware of the state of the house instance and potentially add code to prevent crashes, breaking encapsulation.

By allowing incomplete objects to exist in the system it opens up the program to potential bugs and forces methods to be called in a specific order. If that order is not followed, the program will

crash, indirectly exposing the implementation of the class to developers using the class. This is an example of [Brittle Code](#brittle-code).

Fowler (2004) sums up this problem:

*My long running default with objects is as much as possible, to create valid objects at construction time. This advice goes back to Kent Beck's Smalltalk Best Practice Patterns: Constructor Method and Constructor Parameter Method. Constructors with parameters give you a clear statement of what it means to create a valid object in an obvious place. If there's more than one way to do it, create multiple constructors that show the different combinations.*

---

## Other problems caused by setter injection

### Action at a distance

Setter injection is also a root cause of action-at-a-distance. Any code in the application can change the dependency of any object it has a reference to at any time during the program's execution. For example, consider a *DataMapper* which stores a database connection instance. Any class which is passed the *DataMapper* instance can change the database connection and everywhere else in the application that the *DataMapper* is being used will now be reading from/writing to a different database. By using constructor injection, to do this, a new *Data Mapper* instance must be created which will not have side effects throughout the application.

### Broken encapsulation

By exposing *set* methods, the implementation of the class is exposed to the client code. If the client code is calling a *set* method on an instance, the instance is not polymorphic, the code cannot be reused with an implementation that does not have the same dependency.

To avoid these problems, code from `initailize` methods and setter injection methods should be moved into the constructor as shown in figure 8.7.

```
public class House {
    private int number;
    private String streetName;
    private Kitchen kitchen;
    private Bedroom;

    public House(int number, String streetName, Kitchen kitchen, Bedroom b
```

```
edroom) {
    this.number = number;
    this.streetName = streetName;
    this.kitchen = kitchen;
    this.bedroom = bedroom;
}

public String display() {
    return "the house has a kitchen with an area of " + kitchen.getWi
dth() + " by " + kitchen.getLength();
}
}
```

**Figure 8.7:** Setter injection solution

Summary of issues:

1. Calling methods in the wrong order can cause crashes or bugs
2. Dependencies can be overwritten during the lifetime of the application and introduce bugs.  
E.g. changing the database connection an object depends on while that object is being used in multiple locations

### 8.3.4 Annotations for configuration

Annotations are widely adopted in Java (Reigler, 2014) and are slowly being adopted into other languages (Dohms, 2013), however, in most cases they break the fundamental Object-Oriented principal of *encapsulation* and limit flexibility in the process.

When used for configuration as opposed to documentation annotations cause a large set of problems. For example, a common use-case for annotations is `@Inject` to tell an external Dependency Injection Container to set a specific private property to the given dependency (Oracle, 2011):

```
public class Product {
    @Inject
    private Database db;
}
```

**Figure 8.8:** Annotations example

In figure 8.8, `@Inject` signals to the Dependency Injection Container that a Database instance must be written to the private property. In a lot of common usage, no constructor is even supplied (Bugayenko, 2016). This means it's impossible to use the class without a dependency injection container.

The author of the class has written the code in the expectation that it will be created by a Dependency Injection Container. This breaks encapsulation as there is implied knowledge of other parts of the application within this class, knowledge that it will be used in an environment where a Dependency Injection Container looks for the annotation and supplies the dependency. This severely limits flexibility because there is no easy way to construct the class without using a Dependency Injection Container that understands `@Inject` and knows to look for it.

Encapsulation is broken because the class is no longer in control of its own state, it assumes that it will be running in a very specific environment.

If the class is moved to a project using a different container, or even no dependency injection container, it's usefulness is severely limited because there is no way to set the database dependency.

Instead, the code above should be written as outlined in Figure 8.8.

```

public class Product {
    private Database db;

    Product(Database db) {
        this.db = db;
    }
}

```

**Figure 8.9:** Annotations solution

Compared with figure 8.8, flexibility has been greatly enhanced because the class has no knowledge of the environment it is being used in. There may be a dependency injection container, there may not.

Encapsulation is now maintained, there is no way to instantiate the class without the constructor being run and dependencies supplied.

By using annotations, a dependency is needlessly introduced on the component which reads the annotations and the author of the class has indirect control over how external components can use the class. The class now has two responsibilities: exposing its own interface and telling the outside rules how it should be used. In the example above it is telling the container which dependencies it needs. These are two different responsibilities and this breaks the [Single responsibility principle](#srp) and [Encapsulation](#encapsulation).

Although `Inject` is a common annotation, the problems exist anywhere annotations are used for application configuration.

They store metadata about the class and how the class should be used. This becomes a problem when different projects need different configurations for the class, for example annotations are commonly used for URL routing (Oracle, 2010; Symfony Framework, nd):

```

@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String userName) {

```

```
    ...  
  }  
}
```

**Figure 8.10:** Annotations example 2

In figure 8.10, annotations are used to tell a web-server the URI the class will handle. The annotation sets the route to `users`. This tight coupling of the metadata to the class causes flexibility issues. It's not unreasonable to want to use a class that deals with users on more than one website. However, because the class uses internal metadata using annotations, it's impossible to use the class on the URI `/users` on one website and `/members/` on another without changing the class. This is a violation of the Single Responsibility Principle as the class has more than one reason to change.

If the class is changed then significant issues are introduced with version control: Once a bug is fixed on one website, it's then difficult to copy it over to the other website as the sites are using independent branches.

The solution, shown in figure 8.10, is to separate out the metadata from the class using any format, for example JSON to map the URI path to the class name

```
{  
  "/users": "UserResource",  
  "/Products": "Products"  
}
```

**Figure 8.11:** Annotations example

Figure 8.11 shows that using external metadata, the metadata can differ per website and the class can remain identical, allowing for easier bugfixes and sharing of resources between projects.

As Ahuja (2015) writes:

*My advice is not to use annotations to configure your applications. Configurations were never meant to be part of code—that's why they're configurations and not source code. So*

*let configurations stay in the configuration files. A small productivity gain in the short term won't go amount to much when a client asks for a change in a table or a value and you tell them it's going to take five days of development, testing and deployment.*

---

There is an ongoing debate among developers about if and when annotations should be used. However, the sole benefit of annotations is being able to edit metadata and code in the same file, the argument in favour of their use is always at the expense of flexibility. This is a debate of convenience vs flexibility. From a flexibility point of view, annotations are considerably worse than available alternatives.

It should also be noted that this only applies to annotations which adjust the program's outcome. If removing the annotations does not affect how the program works, the annotations are there for documentation and do not cause any problems with flexibility.

As noted by Walls (2008), annotations also break the law of demeter by adding extra dependencies to classes:

*But, as luck would have it, SearchController also transitively depends on @Searchable and any other Compass annotations I use in Document. That means that I can't compile the web module without it having Compass in its build classpath (that is, as a dependency in its POM. Even though SearchController doesn't know about or need to know about Compass!*

---

*Doesn't this violate the law of demeter?*

---

Although this problem is specific to Java, in other languages annotations may not introduce a hard dependency.

However, there is a remaining issue of comprehension. You could move a class with `@Inject` or similar annotation to a project where the annotations are just comments. Anyone looking at the class in this project will assume that the annotations are used and will be surprised when they change the annotations and it doesn't affect the configuration. This is not directly an issue of flexibility but it breaks the *Principle of Least Surprise* (Raymond, 2003; James, 1987) and makes the class more difficult to reuse because it's not clear to anyone reading the code.

## **Summary of problems:**

### **Practical issues**

- Cannot be debugged easily as you can't print the contents of the configuration
- Finding application configuration requires looking across every class in the application
- Breaks the Single Responsibility Principle



- Breaks Encapsulation
- Breaks Separation of Concerns
- Introduces ambiguity/bugs in polymorphic code
- Introduces coupling between unrelated components
- Makes it more difficult to instantiate an object with different configurations
- Makes version control more difficult

**Negative traits:**

- Broken encapsulation
  - Action at a distance
  - Hardcoded values
  - Implicitly breaks Law of Demeter by setting properties outside the class scope
-

### 8.3.5 Use of static methods

Use of static methods always reduces flexibility by introducing tight coupling (Popov, 2014). A static method tightly couples the calling code to the specific class the method exists in.

```
public double totalAbs(double value, double value2) {  
    return Math.abs(value) + Math.abs(value2);  
}
```

**Figure 8.12:** Static methods example

In figure 8.12, the method `totalAbs` has a dependency on the `Math` class and the `.abs()` method will always be called. Although for testing purposes this may not be a problem, the coupling reduces flexibility because the `total` method can only work with doubles/integers, as that's all the `Math.abs()` function can use. Although type coercion will allow the use of any primitive numeric type, these types have limitations. It's impossible to use another class such as `BigInteger` or a class for dealing with greater precision decimals or even alternative numbering systems such as Roman numerals.

The `totalAbs` function takes two doubles and converts them to their absolute values before adding them. This is inflexible because it only works with doubles. It's tied to doubles because that's what the `Math.abs()` static method requires. If, instead, using OOP an interface was created to handle any number, such as described in figure 8.12:

```
interface Numeric {  
    public Numeric abs();  
}
```

**Figure 8.13:** Static methods solution

Using code from figure 8.13, would then be possible to rewrite the `totalAbs` method to work with any kind of number:

```
public Numeric totalAbs(Numeric value, Numeric value) {
```

```
    return value.abs() + value2.abs();
}
```

**Figure 8.14:** Static methods solution

In figure 8.14, By removing the static method and using an instance method in its place the `totalAbs` method is now agnostic about the type of number it is dealing with. It could be called with any of the following (assuming they implement the `Numeric` interface).

```
<?php totalAbs(new Integer(4), new Integer(-53));

totalAbs(new Double(34.4), new Integer(-2));

totalAbs(new BigInteger("123445454564765739878989343225778"), new Integer(
2343));

totalAbs(new RomanNumeral('VII'), new RomanNumeral('CXV'));
```

**Figure 8.15:** Removal of static methods offers more flexibility

Figure 8.15 demonstrates how the method can be used in different context by changing the static methods to instance methods and that flexibility has been enhanced as the method can be used with any numeric type, not just numeric types that are supported by the `Math.abs()` method.

Static methods also break encapsulation. Encapsulation is defined by Rogers (2001) as:

*the bundling of data with the methods that operate on that data*

---

By passing the numeric value into the `abs` method, the data being operated on is being separated from the methods that operate on it, breaking encapsulation. Instead using `num.abs()` the data is encapsulated in the `num` instance and its type is not visible or relevant to the outside world. `abs()` will work on the data and work regardless of `num`'s type, providing it implements the `abs` method.

This is a simple example, but applies to all static methods. Use of polymorphic instance methods that work on encapsulated data will always be more flexible than static method calls which can only ever deal with specific pre-defined types.

## Exceptions

The only exception to this rule is when a static method is used for object creation in place of the `new` keyword (Sonmez, 2010). This is because the `new` keyword is already a static call. However, even here a non-static factory is often preferable for testing purposes (Hevery, 2008; Butler, 2013).

---

### 8.3.6 Using `new` in constructor

If a dependency is constructed inside the object that requires it rather than passed in as a reference then flexibility is lost (Hevery, 2008; Hevery, 2008)

```
public class Car {  
  
    private Engine engine;  
  
    public Car() {  
        this.engine = new Engine();  
    }  
}
```

**Figure 8.16:** New in constructor example

In figure 8.16, the Car constructor creates the Engine instance. This is inflexible as it forces all Car objects to use the exact same Engine type. Instead, it would encourage reuse if the program supported different engine types (e.g. DieselEngine, PetrolEngine or HybridEngine).

The same is true when an instance variable is created when the class is defined:

```
public class Car {  
    private Engine engine = new Engine();  
}
```

**Figure 8.17:** New in constructor example 2

In figure 8.17, by using the new keyword to instantiate a dependency, the specific implementation of that dependency is hardcoded and cannot be substituted.

Instead, the dependency should be constructed outside the class and injected in:

```
public class Car {
```

```

private Engine engine;

public Car(Engine engine) {
    this.engine = engine;
}
}

```

**Figure 8.18:** New in constructor solution

By using using dependency injection, as demonstrated in figure 8.18, it is possible to pass in any engine type:

```

//Instead of
Car myCar = new Car();

//It's now possible to construct different types of car:
Car petrolCar = new Car(new PetrolEngine);
Car electricCar = new Car(new ElectricEngine);

```

**Figure 8.19:** Dependency Injection

Figure 8.19 demonstrates how dependency injection can be used to inject any engine type.

A secondary advantage to Dependency Injection with regards to flexibility and encapsulation is that the class which has the dependency (Car, in this example) it not aware of the dependencies of the Engine class.

For example, if the Engine class required a Gearbox instance as a constructor argument, the Car class would need to instantiate and pass in the relevant Gearbox instance. And provide any dependencies of the Gearbox class when instantiating it.

If the constructor arguments of any of the classes which need to be instantiated are modified during development, any class which creates an instance of the class must also be modified. A change to the constructor for Engine would require modifying the Car class. Instead, if the fully constructed Engine instance

By loosely coupling the Engine class to the Car class, the author of the Car class does not need to know anything about the implementation of Engine class or have knowledge of what dependencies it has.

## Exceptions

Using the new keyword is not a problem for the follow cases:

1. Factory methods - The single concern of the method is constructing and returning a new instance, it should not be able to be substituted.
2. An immutable object returning a new instance of itself. A class is already tightly coupled to itself, encapsulation is not broken and creating a new representation of itself is the method's task.

## Summary of issues

- Tight Coupling
  - Breaks single responsibility principle
  - Breaks encapsulation (the class with the new keyword knows implementation details e.g. dependencies of the class being instantiated)
-

### 8.3.7 Inheritance

Inheritance has been pointed at as a source of programming problems since at least 1999. The popular book *Design Patterns: Elements of Resusable Object-Oriented Software* by Gamma *et al* (1994) has a long section on replacing inheritance with composition. Holub (2010) quotes a speech by James Gosling, the inventor of Java, who said he would leave out inheritance if he were to design Java again.

The biggest problem with inheritance is the tight coupling it creates:

*One of the problems with implementing an abstract class with inheritance is that the derived class is so tightly coupled to the base class*

---

Martin (2000)

Inheritance creates tight coupling because there is no way to substitute the parent class at runtime and the subclass is stuck with the same parent class forever.

For example, consider a Car class which has the following subclasses: PetrolCar, DieselCar, ElectricCar. All 3 subclasses are tightly coupled to the same base-class. It's likely a lot of the functionality in the PetrolCar class is also applicable to the PetrolBoat class, however, due to the tight coupling, the only way to share the functionality between the PetrolBoat and PetrolCar classes is copying/pasting the relevant properties and methods.

Instead, *is-a* relationships are better expressed as *has-a* relationships. A boat *has-a* engine, instead of a petrol powered boat *is-a* boat, an employee *has-a* job instead of an employee *is-a* person, a cat *has* warm blood, fur and paws instead of a cat *is-a* animal.

For example, consider the classic inheritance Employee-Person example:

```
class Person {
    public String name;
    public String address;
    public String gender;
    public Date birthdate;
}

class Employee extends Person {
    public String jobTitle;
}
```



```
public int salary;  
public Date startDate;  
}
```

**Figure 8.20:** Inheritance example

The code demonstrating inheritance in figure 8.20 is inflexible as:

- A person cannot ever have more than one job
- Only people can have jobs, it is impossible to represent robots in factories or animals who work (e.g. police dogs)

Instead, by representing the class structure using *has-a* and composition, these problems can be avoided as demonstrated in figure 8.21:

```
Person tom = new Person('Tom', new Job('Software Developer', 30000));  
  
Animal rex = new Animal('Rex', new Job('Police Dog'));  
  
Robot curiosityRover = new Robot('Curiosity', new Job('Exploration of Mars  
'));
```

**Figure 8.21:** Inheritance solution

and the classes could be written to allow multiple jobs as demonstrated in figure 8.22:

```
<?php Person tom = new Person('Tom');  
tom.addJob(new Job('Software Developer'));  
tom.addJob(new Job('University Lecturer'));
```

**Figure 8.22:** Benefits of removing inheritance

When developing software it's easy to say *The system doesn't need to support an animal with a job*, however requirements change frequently during a project and this reasoning prevents the reuse of the class in the next project where you may well need this flexibility. By replacing inheritance with composition, flexibility is always improved by removing tight coupling.

Holub (2010) sums up the problem as:

*Why should you avoid implementation inheritance? The first problem is that explicit use of concrete class names locks you into specific implementations, making down-the-line changes unnecessarily difficult.*

---

The major problem with inheritance is that is essentially *static*, there is no way to dynamically change the base class like you can with composition. When a method is invoked such as `this.drive()`, the `drive` method is a very tight coupling. It is either the method in the same class or a parent class and there is no way to override it without making a new subclass. However, when using composition `this.engine.drive()`, the `Engine` object can be substituted at runtime and the `drive()` method could be any one of an infinite possible number of implementations.

Other problems with inheritance include

### **The fragile base class problem, inheritance breaks encapsulation**

This is a problem that occurs because the author of the base class does not know how it is being used by subclasses. A simplified version of an example given by Aldrich (2004) is as follows:

Consider two classes written by different authors. The classes act as counters, the first one counts in steps of 1 or 2 and the second counter, counts in double the speed as demonstrated in figure 8.23:

```
class Counter {
    private int counter 0;

    public void add1() {
        counter = counter + 1;
    }

    public void add2() {
        counter = counter + 2;
    }
}
```

```

}

class CountInTwos extends Counter {
    public void add1() {
        add2();
    }

    public void add2() {
        add2();
        add2();
    }
}

```

**Figure 8.23:** Fragile base class example

These classes work perfectly. However, there is no way for the author of the parent class to know how or where their base class is being used. It's not unreasonable for them to re-write their class at some point, as shown in figure 8.24:

```

class Counter {
    private int counter 0;

    public void add1() {
        counter = counter + 1;
    }

    public void add2() {
        add1();
        add1();
    }
}

```

**Figure 8.24:** Fragile base class problem

Anywhere Counter is used still works perfectly, all unit tests on the class still pass and the API hasn't changed. However, this affects CountInTwos: There is now an infinite loop: The add1() function calls add2(), which calls add1(), creating infinite recursion. The subclass's functionality has been broken by a seemingly harmless change in the base-class.

This is *action at a distance* as a change in one place inadvertently breaks code elsewhere.

Aldrich (2004) suggests several solutions to this problem, however they involve careful class design and requiring the author of the base class to design their class around the problem in order to prevent it. A simpler solution is avoiding inheritance in the first place (Holub, 2010; Biberstein *et al*, 2002; Bloch, 2008; Gamma *et al* p.20, 1994; van Dongen, 2014) giving more flexibility to class design.

---

*Because inheritance exposes a subclass to details of its parent's implementation, it's often said that 'inheritance breaks encapsulation'*

---

Gamma *et al* (1994)

---

*Though both Composition and Inheritance allows you to reuse code, one of the disadvantage of Inheritance is that it breaks encapsulation. If sub class is depending on super class behavior for its operation, it suddenly becomes fragile. When behavior of super class changes, functionality in sub class may get broken, without any change on its part.*

---

Paul (2013)

The fragile base class problem exists because inheritance subtly breaks encapsulation (Biberstein *et al*, 2002; Gamma *et al* p.20, 1994). In the example above, the CountInTwos class is subtly exposed to the implementation of the parent class. Changes in the parent class are reflected in child class meaning the two classes are not self-contained. It is expected API changes will always have an affect elsewhere no matter the relation type, with inheritance *implementation* changes in one class can affect the outcome of another class, even if the arguments and return values of the method do not change.

Protected and public properties also break encapsulation in a much more obvious way. Protected properties share data between two classes, breaking encapsulation in a much more obvious way (Martin p.80, 2008).

### **The diamond problem**

A major problem with inheritance is that it becomes difficult to share code in large inheritance

trees. The diamond problem is demonstrated in figure 8.25:

```
class Bird extends FlyingAnimal {  
  
}  
  
class Fish extends SwimmingAnimal {  
  
}
```

**Figure 8.25:** The diamond problem

How would a developer model a penguin in this inheritance tree. Hurn (2014) and van Dongen (2014) also provide detailed examples of this problem. Whenever you get an inheritance hierarchy you run into this problem when the project becomes large enough, you will find you need to share some properties of two classes at opposite ends of the hierarchy. The only fix becomes a large refactoring of the class hierarchy or duplicating code. As duplicating code is a large maintainability problem[martin-2008 p.173,mcconnel-2004 p.565,hunt-1999 p.48], the first is more favourable, however the problem can be avoided all together by favouring composition over inheritance at the start of a project (Otander, 2015).

Some languages do allow for *Multiple Inheritance* which solves the diamond problem, however it doesn't fix the static method calls or the situations above e.g. multiple jobs or different things that can have jobs.

### **Separation of concerns, single responsibility principle**

The final problem with inheritance is that it breaks the single-responsibility-principle. The single responsibility principle was coined by Martin (2003) who defines it as:

*A class should have only one reason to change*

---

And later reiterated it as

*Gather together the things that change for the same reasons. Separate those things that change for different reasons.*

---

Martin (2014)

If a subclass and a parent class are sharing an API but are otherwise different, they will change at different times for different reasons. By definition changing the base class will change the subclass, giving the subclass at least two reasons to change.

In the case of inheritance this has little practical effect and breaking the single responsibility principle in this way doesn't cause the same issues as it normally would because there is some separation between classes. However it's worth noting that a subclass will always export two APIs.

As with most programming practices, there are trade-offs when choosing which approach to use. Inheritance can be easy to use and understand, it can also simplify designs (Gamma *et al* p.25, 1994), but composition is always more flexible (Gamma *et al* p.25, 1994) and should be favoured for flexibility (Paul, 2013; Johansson, 2015; van Gurp *et al*, 2001)

Summary of problems of Inheritance:

- Tightly couples the child to the parent
  - Breaks Single Responsibility Principle
  - Diamond Problem
  - Fragile Base Class Problem
-

### 8.3.8 Global/Static variables

The identification of global variables as a bad practice dates as far back at least as far as Wulf *et al* (1973) and are one of the most widespread and well known *bad practices* related to flexibility. This is likely due to being available in almost every programming language, ease of use and speed to learn. They also cause severe problems in code and it's very easy to get caught out by using them, even in a small application.

Global variables are widely labelled "bad practice" and have been for some time, for example back in 1999 Kernighan wrote:

---

*Avoid global variables; wherever possible it is better to pass references to all data through function arguments*

---

Kernighan (1999)

And Hevery (2008) states:

---

*I hope that by now most developers agree that global state should be treated like GOTO.*

---

This attitude is widespread and Sayfan (n.d.) sums up the problem:

---

*Whenever shared mutable state is involved, it is easy for components to step on each other's toes.*

---

Although "global variables are bad" is a common thing to hear, for novice developers it's not immediately obvious why this is. However, the reasons have been covered frequently by developers of varying prominence. While writing about designing the Eiffel programming language, (Meyer, 1988) stated several problems with global variables:

*Since global variables are shared by different modules, they make each of these modules more difficult to understand separately, diminishing readability and hence hampering maintenance.*

*As global variables constitute a form of undercover dependency between modules, they are a major obstacle to software evolution, since they make it harder to modify a module without impacting others.*

*They are a major source of nasty errors. Through a global variable, an error in a module may propagate to many others. As a result, the manifestation of the error may be quite remote from its cause in the software architecture, making it very hard to trace down errors and correct them. This problem is particularly serious in environments where incorrect array references may pollute other data.*

---

This is a good overview but it misses out on some of the more practical problems that developers face when using global variables. The first issue most people will face is name clashes:

*everywhere in the program, you would have to keep track of the names of all the variables declared anywhere else in the program, so that you didn't accidentally re-use one.*

---

Summit (1997)

The problem of name clashes is magnified by the size of a team. If two people are working on a piece of software and both use global variables, it's possible they'll write some code using the same variable names. During execution this might cause the two peices of code to interfere with each other.

This problem is commonly referred to as *action at a distance* and described by Hevery (2008) as:

*Spooky Action at a Distance is when we run one thing that we believe is isolated (since we did not pass any references in) but unexpected interactions and state changes happen in distant locations of the system which we did not tell the object about. This can only happen via global state as demonstrated in figure 8.26:*

---

```
class FileReadWrite {
    private static String fileName;

    public FileReadWrite(String file) {
        fileName = file;
    }

    public void read() {
        return new FileReader(fileName);
    }

    public void write(String data) {
        FileWriter writer = new FileWriter(new File(fileName));
        writer.write(data);
    }
}
```



```
//Works as expected:

FileReadWrite file = new FileReadWrite('./one.txt');
file.write('data');

//cause a problem

FileReadWrite file1 = new FileReadWrite('./one.txt');
FileReadWrite file2 = new FileReadWrite('./two.txt');

file1.write('data1');
file2.write('data2');
```

**Figure 8.26:** Global variables example

The code in figure 8.26 causes a problem because there is a global variable storing the file name. Assuming a class requires only one value of a variable across the whole application always limits flexibility. There are occasionally practical reasons for this such as keeping track of and limiting the number of open files/connections but flexibility is always reduced. Even in these practical exceptions, it introduces a new issue of separation of concerns: Should the class be concerned with the number of open connections throughout the application or should that be managed at an application level rather than a class level?

Global variables also break encapsulation. Encapsulation is defined by Rogers (2001) as:

*Encapsulation refers to the bundling of data with the methods that operate on that data*

---

By making the data globally accessible, encapsulation has been lost. Any part of the program has access to the data and can modify it. Even when using private static variables, each instance no longer has control of its own state.

Finally, global variables introduce coupling. In Object Oriented Programming an object should be *self-contained* (Yaiser, 2011; Caromel, 1993). If a class depends on a global variable, then moving the class to a different project requires defining the required global variables in the new project. However, instance and static class variables do not have this problem because they are defined as part of the class.

## Summary of problems introduced by global variables

- Coupling of data between every component
- One component can accidentally overwrite data required by another component
- Adding code requires knowing exactly what variables are already in use
- When working in teams, name clashes can be easily introduced
- Global state makes it difficult to reuse the code. E.g. having two files open at the same time would require writing the code twice, three times for three files, etc.

## Appendix IV. Full explanations of negative traits

Below are the full explanations of negative traits which are introduced by bad practices. These explanations can be referenced in future work or embedded in future tools using this research.

### 8.4.1 Broken encapsulation

Encapsulation is a fundamental feature of Object-Oriented design. It is generally defined as:

*Bulding the data with the methods that act on that data*

---

Encapsulation be broken in many ways. The most obvious is public properties:

```
class User {
    public Database database;
    public int id;
    public String name;
    public String emai;
public void save() {
    Query query = this.database.query('UPDATE user SET name = ?,
email =? password = ? WHERE id = ?');
    query.bind(this.name);
    query.bind(this.email);
    query.bind(this.id);
    query.execute();
}
}
```

**Figure 8.27:** Broken Encapsulation example

In figure 8.27, the database property is public, anything with access to a User instance can change the database connection, for example, the code in figures 8.28 or 8.29.

```
public function processUser(User user) {
    user.database = null;
```

```
//...  
}
```

**Figure 8.28:** Demonstration of Broken Encapsulation issue (a)

```
public function processUser(User user) {  
    user.database.disconnect();  
    //...  
}
```

**Figure 8.29:** Demonstration of Broken Encapsulation issue (b)

By exposing properties, an object is no longer in charge of its own state. Any external code can change the dependency at any point in the execution of the program.

This causes maintainability as properties cannot be relied upon and the code cannot be rewritten to no longer use them. For example, a `total` property cannot be refactored to a `getTotal()` method that performs a calculation.

Example 2:

A second example, shown in figure 8.30, is unbundling the behavior and the data being worked on:

```
// Take two numbers and add the absolute values together  
public int addAbs(int value1, int value2) {  
    return Math.abs(value1) + Math.abs(value2);  
}
```

**Figure 8.30:** Operating on data outside of the scope it is defined in

In figure 8.30, the `Math.abs()` method is operating on data it does not own and therefore encapsulation has been broken.

The data being worked on (`value1` and `value2`) is exposed to the `addAbs` method rather than encapsulated.

This is evident because it's impossible to substitute the behaviour for a different type, only integers will work. The `addAbs` method cannot ever work with different types. For example, `BigInteger`, `BigDecimal` or `RomanNumeral`.

With encapsulation, the code would be expressed as shown in figure 8.31 and 8.32.

```
interface Numeric {
    public Numeric abs();
}
```

**Figure 8.31:** Alternative approach using an interface

```
// Take two numbers and add the absolute values together
public int addAbs(Numeric value1, Numeric value2) {
    return value1.abs() + value2.abs();
}
```

**Figure 8.32:** Using the interface from figure 2.33

In figure 8.32, as the data type is never exposed to the method, it is not limited to the types supported by `Math.abs()`.

In the original version of the code from figure 8.30, the `addAbs` method would not work with an instance of `BigInteger`, `BigDecimal` or `RomanNumeral` because encapsulation has been broken and the only way to add support for it would be to add support in the `Math` class.

The updated version, using the `Numeric` interface is now type agnostic. It will work with any class that implements `Numeric` as each class provides its own implementation of the `abs` method, new numeric types can be easily added.

Static methods like `Math.abs()` always break encapsulation because they operate on data which they do not own.

## 8.4.2 Single Responsibility Principle

The single responsibility principle was created by Martin (2000) who described it as "An class should have a single reason to change".

A class which has two or more responsibilities is difficult to maintain because the behaviour is tightly coupled and difficult to replace.

For example, an object with a `serialize()` method knows that it is going to be serialized (broken encapsulation) but it has two responsibilities:

1. Whatever the class should be doing.
2. Providing the serialization mechanism.

If a different serialization format is added, for example JSON, a second method `serializeJson()` would be required, for XML, `serializeXml()` and so on.

If a class has more than one responsibility, it is impossible to substitute one of the responsibilities (e.g. the serialization method above).

The above example would be better expressed as shown in figure 8.33:

```
obj.serialize();
```

**Figure 8.33:** Broken single responsibility principle

The code in figure 8.34 is preferred as the object is not responsible for how it is serialized:

```
serializer.serialize(obj);
```

**Figure 8.34:** Demonstrating the single responsibility principle

With this approach, where `obj` is not in charge of how it is serialized, different serialization methods can be used on the same object as shown in figure 8.35.

```
XMLSerializer.serialize(obj);  
JSONSerializer.serialize(obj);  
YAMLSerializer.serialize(obj);
```

**Figure 8.35:** Advantage of following the single responsibility principle

### 8.4.3 Unclear dependencies

A class which has dependencies that are not made clear as part of the class API. Someone looking at the class has to very carefully look through the code to identify the coupled classes.

#### Why this is an issue

Moving the class between projects is difficult as it requires locating and also moving all the classes which

#### Example

This is often a result of a Law of Demeter violation or service locator: `a.getB().c()`, it is not clear that the class has a dependency on the type returned by `getB()`.

### 8.4.4 Temporal Coupling

Temporal Coupling is a situation where the order in which methods are called on an object has a negative effect if one method must be called before another for the object to fulfil its contracts. (Seeman, 2011; Bugayenko, 2015).

This happens commonly with `init` and `initialize` methods. For example:

```
class Car {  
    private Engine engine;  
public void initialize(Engine engine) {  
    this.engine = engine;  
}  
public void drive() {  
    this.engine.start();  
}  
}
```

### Figure 8.36: Temporal Coupling example

In figure 8.36, when the Car class is used, if the drive method is called before the initialize method, the object will not work correctly.

If an object can reach such a state, where methods must be called in a 'correct' order and 'incorrect' order, then it suffers from *Temporal Coupling*.

### 8.4.5 Law of Demeter

The law of demter is closely related to [encapsulation](#encapsulation) and [tight coupling](#tight-coupling) and a subset of Unnecessary Coupling. It involves breaking encapsulation by exposing dependencies (Hevery, 2008; Grimm, 2014; Haack, 2009). This is demonstrated in figure 8.37:

```
public void refuel(Car car) {  
    car.getEngine().addFuel(new Diesel);  
}
```

### Figure 8.37: Law of Demeter example

This creates coupling between the calling code (the refuel method) and the engine class. The calling code is now exposed to all the methods in the car class and all the methods in the engine class.

This means that because the car is exposing the engine class to the calling code, the car class cannot have its implementation changed to use a different engine.

For example, a car may have an electric engine that has a charge() method and not a refuel() method. However, the calling code here is coupled to a diesel engine not an electric engine. By properly encapsulating the engine inside the car class and not exposing it to calling code, the client code that is using the car instance is only coupled to the car class. To anyone looking at the code it's easy to see what dependencies the car class has (As they should be listed in the constructor). The engine class can change both its implementation and its API without breaking any classes other than the ones that use it directly and the car class can be rewritten to use a different engine type (or no engine at all) and the calling code will still work. The fact that the car even has an engine should be hidden from the outside world.



This makes testing difficult. To test the `refuel` method above, a `Car` instance needs to be created just so it can return the real `Engine` instance. If there are other classes being used by the car class (`Door`, `Seat`, `Wheel`, etc) these all need to be created (or mocked) just to test the engine's `refuel` method. This leads to very verbose and messy tests. If the test fails, is it due to a problem in the `Car` class or the `Engine` class?

Instead, if the code was rewritten as shown in figure 8.38 to only use the engine object then the test only needs to create an `Engine` instance and not the whole object stack starting from the `Car` instance. This also greatly enhances the flexibility of the `refuel` method because it can work on an engine that isn't stored inside a `Car` instance. For example, inside the application the `Engine` could be part of a `Boat` or `Plane` or `Car`, but there is no reason for the method to know this.

```
public void refuel(Engine engine) {
    engine.addFuel(new Diesel);
}
```

**Figure 8.38:** Avoiding breaking the Law of Demeter

The problems created by breaking the Law of Demeter are exacerbated by digging further into the object graph as shown in figure 8.39.

```
public void refuel(Driver driver) {
    driver.getVehicle().getEngine().addFuel(new Diesel);
}
```

**Figure 8.39:** Digging deeper into the object graph

To test the method shown in figure 8.39, the object graph shown in figure 8.40 must be created.

```
new Driver(new Car(new Engine));
```

**Figure 8.40:** Object graph required to test the code shown in figure 2.41

If the test using the object graph from figure 8.40 fails, it's not clear which class the problem exists in. If the only object that is created is the `Engine` class and the test fails, the developer immediately knows where to look for the bug.

Although this is an example of loose coupling, loose coupling is less flexible than entirely decoupled code.

Because the `refuel` method is coupled to both the `Car` class and the `Engine` class, there are some practical problems:

- It cannot be used to add fuel to an engine that's not in a car, e.g. refueling a boat.
- To move the `Engine` class and `refuel` method to another project, the `Car` class must also be moved, even if only the `Engine` class is required.

The same underlying idea is also expressed by Fowler (2013) as "Tell, Don't ask"

## 8.4.6 Tight Coupling

Coupling, with objects, is much like "Coupling" with people, it describes how a pair of objects interact with one another.

There are two different types of coupling: Tight coupling and Loose coupling. These describe the different types of relationships between objects.

A married couple can be said to be *tightly coupled* because they are only allowed to date one another. A single person can date anyone they like. In php the married couple who are tightly coupled can be demonstrated as:

```
class Dave {
    private Kate partner;
public Dave() {
    this.partner = new Kate();
}
public void date() {
    this.partner.takeOut();
}
```

```
}
```

**Figure 8.41:** Example of Tight Coupling

In figure 8.41, Dave can only ever take Kate out on a date. Whenever `dave.date()` is called, the `takeOut()` method will be called on an instance of Kate. There is no way for Dave to take a different partner on a date.

```
Dave dave = new Dave();  
dave.date();
```

**Figure 8.42:** No dependencies are visible externally

In figure 8.42, no dependencies are visible outside of the class. This is the essence of tight coupling. Dave is tightly coupled to Kate because there is no way for Dave to take anyone else on a date in this code.

Bob, shown in figure 2.45, represents a single person:

```
class Bob {  
    private Partner partner;  
public Bob(Partner partner) {  
    this.partner = partner;  
    }  
public void date() {  
    this.partner.takeOut();  
    }  
}
```

**Figure 8.43:** Example of loose coupling (a)

There's only one difference between Dave and Bob, and that is, Bob can have his partner assigned when the instance is created using Dependency Injection. This gives Bob more flexibility because

he can go on a date with anyone.

```
Bob bob = new Bob(new Amy);  
bob->date();  
Bob bob = new Bob(new Kate);  
bob->date();  
Bob bob = new Bob(new Dave);  
bob->date();  
Bob bob = new Bob(new Bob);  
bob->date();
```

**Figure 8.44:** Example of loose coupling (b)

Figure 8.44 shows how the Bob instance can be configured with any partner. This is the difference between tight and loose coupling in Object-Oriented Programming.

### Real world example

The above example is contrived. In real programs loose coupling is incredibly useful as it allows programmers to build code that can work with different collaborators. Consider the program in figure 8.45 which acts as a newsletter signup form. It takes some data from the user via `formSubmission` and saves it to a file:

```
class Signup {  
    private File file;  
public Signup() {  
    this.file = new File('./signups.txt');  
    }  
public void process(FormSubmission formSubmission) {  
    this.file.writeObject(formSubission);  
    }  
}
```

**Figure 8.45:** Real world example of tight coupling

The file class used in figure 8.45 has a `writeObject` method that takes an object and writes it to a file. How that works is not important for this example but it could be assumed that it writes it in JSON: `{"name": "Bob", "email": "bob@example.org"}`.

One immediately obvious problem with this approach is that it can only ever write to `signups.txt`. To change the file being written to, the class must be edited. This prevents me having two instances of the `Signup` class that write to different files. It's not unreasonable that the system could be extended to have multiple newsletters that the user wants to sign up to, if all the sign up data is stored in the same file it wouldn't be possible to know which user wants to sign up for which newsletter

Figure 8.46 shows how this can be overcome by passing the file name as a constructor argument.

```
class Signup {
    private File file;
public Signup(string fileName) {
    this.file = new File(fileName);
}
public void process(FormSubmission formSubmission) {
    this.file.writeObject(formSubission);
}
}
```

**Figure 8.46:** Enhanced Signup example class

Figure 8.47 demonstrates the added flexibility, as an instance of `Signup` is created, the file name can be set to the file for the particular mailing list being signed up to.

```
new Signup('./news.txt');
new Signup('./specialoffers.txt');
```

**Figure 8.47:** Enhanced Signup example class usage

Figure 8.47 shows an improvement over the previous version, however it is still less flexible than it

could be. In this version, newsletter sign ups can only be written to a file because the constructor instantiates a `File` instance. Instead, as demonstrated in figure 8.48, if the instance was passed in using dependency injection it would be possible to use any storage mechanism. For example, swapping out file storage for database storage:

```
class Signup {
    private Storage storage;
public Signup(Storage storage) {
    this.storage = storage;
}
public void process(FormSubmission formSubmission) {
    this.storage.writeObject(formSubission);
}
}
```

**Figure 8.48:** Signup class using loose coupling

The code is almost exactly the same, the only difference is that the coupling of the storage instance has been changed the coupling from tight to loose. The signup class still requires a storage mechanism but it isn't coupled to the specific implementation that writes to a file. Figure 8.50 demonstrates the flexibility of using the `Signup` class with any storage mechanism:

```
new Signup(new File('./news.txt'));
new Signup(new File('./specialoffers.txt'));
new Signup(new Database('127.0.0.1', 'username', 'password', 'tablename'));
new Signup(new Database('127.0.0.1', 'username', 'password',
'another_table'));
new Signup(new RESTApi('rest.example.org'));
```

**Figure 8.49:** Signup class using loose coupling usage

### Other forms of tight coupling

In addition to using the `new` keyword as above, there are other ways of introducing tight coupling

into Object-Oriented code.

## Inheritance

Inheritance is another method of tightly coupling classes. When using inheritance, it's impossible to substitute the base class at runtime and the relationship is incredibly rigid.

The example from figure 8.41 could be modeled using inheritance as shown in figure 8.50:

```
class Dave extends Kate {
    public void date() {
        super.takeOut();
    }
}
```

**Figure 8.50:** Tight coupling with inheritance

Whenever the `date()` method is called, the `takeOut()` method from the `Kate` class is called and there is no way of substituting this without rewriting the class. Once again, `Dave` can only date `Kate` because they are tightly coupled.

Figure 8.51 demonstrates how the `Signup` class could also be modeled using inheritance:

```
class Signup extends File {
    public void process(FormSubmission formSubmission) {
        super.writeObject(formSubission);
    }
}
```

**Figure 8.51:** The `Signup` class modeled using inheritance

The same problem occurs in figure 8.51 as it does in figure 8.46. The only `writeObject` implementation that can be used without rewriting the class is the implementation in the `File` class because of the tight coupling which has been introduced, this time using inheritance. There is no way to substitute the implementation at runtime.

Although it's possible to override the `writeObject` method in the `Signup` class, the `Signup` class would still be locked to a single implementation (the one in the `Signup` class). There would be no way to replace the implementation without physically altering the code, either by changing the `Signup` class or adding a new subclass which overrides the behaviour.

Inheritance and using `new` in a constructor are identical in the way they limit flexibility and in the way they force class design. They both prevent runtime substitution and require modifying the code in the class to alter the execution of the method.

## Static Methods

Tight coupling is also introduced via static methods. The above examples could also be re-written using static methods as show in in figures 8.53 and 8.54:

```
class Dave {
    public void date() {
        Kate.takeOut();
    }
}
```

**Figure 8.52:** Tight coupling using static methods

```
class Signup {
    public void process() {
        File.writeObject($signup);
    }
}
```

**Figure 8.53:** Using static methods to model the Signup class

Static methods introduce exactly the same problem as inheritance and using the `new` keyword. There's no way to substitute where the data is being written without changing the code.



## Solution

The solution, for all three of these issues is to use *loose coupling* via dependency injection as shown in figures 8.54 and 8.55.

```
class Dave {
    private $partner;
public function __construct($partner) {
    $this->partner = $partner;
    }
public function date() {
    $this->partner->takeOut();
    }
}
//...
Dave dave = new Dave(new Amy);
dave.date();
Dave dave = new Dave(new Kate);
$dave.date();
Dave dave = new Dave(new Dave);
$dave.date();
Dave dave = new Dave(new Bob);
dave.date();
```

**Figure 8.54:** Loosely coupled Dave class

```
class Signup {
    private Storage storage;
public Signup(Storage storage) {
    this.storage = storage;
    }
public void process(FormSubmission formSubmission) {
    this.storage.writeObject(formSubission);
    }
}
```

```
}  
//...  
new Signup(new File('./news.txt'));  
new Signup(new File('./specialoffers.txt'));  
new Signup(new Database('127.0.0.1', 'username', 'password', 'tablename'));  
new Signup(new Database('127.0.0.1', 'username', 'password',  
'another_table'));  
new Signup(new RESTApi('rest.example.org'));
```

**Figure 8.55:** Loosely coupled Signup class

This alternative is much more flexible as:

- The storage mechanism can be replaced at runtime `new Signup(new Database)`
- Multiple instances of `Signup` can be created, each with a different storage mechanism
- The code can be moved between different projects and different projects can use different storage mechanisms (a file on one project, a database on another, all with the same `Signup` class)

### How to identify tight coupling

Tight coupling exists whenever a class named is explicitly used inside another class. Whether that's a static method call, `extends` or the `new` keyword. The result is the same, the relationship between the two classes is rigid and it's impossible to substitute the dependency for a different implementation.

### Conclusion

When defining relationships between classes there are multiple approaches which can be taken: instantiating one object inside another, inheritance, static methods, singletons or dependency injection.

Dependency Injection is by far the most flexible as it doesn't define a hardcoded relationship between the classes and objects can be used with different dependencies.

Loose coupling makes classes far more flexible with little to no extra effort required. Using loose coupling, as the requirements of the project inevitably change it's very easy to implement the updates. With tight coupling it can be incredibly difficult, especially when the program needs to be achieve two methods of doing the same thing (e.g. having the signup write to a file in one place

and a database in another.)

By using Dependency Injection, the object can be instantiated with as many different configurations as required and isn't tied to a specific implementation. With tight coupling, the configuration is explicitly defined inside the class and there is no way to override it.

Considering this very useful benefit and zero extra development time or drawbacks, using loose-coupling is a no-brainer!

### 8.4.7 Global State

Global state exists when a variable is available in global scope and can be altered by any part of the application.

#### Example

A database connection in global scope could be disconnected in one part of the application, inadvertently breaking another part of the application that expects the connection to still be active.

#### Issues

This causes buggy applications because one part of the application can accidentally overwrite a variable being used for a completely different purpose elsewhere.

In addition, it is very difficult to remove global variables from an application once they exist because it's hard for the developer to know where or if the variable is being used.

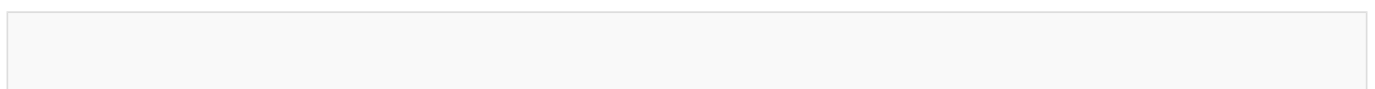
#### Solution

All variables should be private instance variables. It is then clear to someone looking at the code that the variable can only ever be used in the class in which it is defined. If the variable is no longer used in the class, it can be safely removed.

### 8.4.8 Unnecessary Coupling

Unnecessary Coupling exists when one class is coupled to a class but does not need to be. This usually happens when one class is used as a 'kitchen sink' object and passed around to dependencies.

Figure 8.56 shows a Car class which accesses an Engine instance via a CarParts container.



```

class Car {
    private CarParts parts;
public Car(CarParts parts) {
    this.parts = parts;
}
public function drive() {
    Engine engine = this.parts.getEngine();
    engine.start();
}
}

```

**Figure 8.56:** Unnecessary Coupling example

In this example, the Car class doesn't have a dependency on CarParts, it has a dependency on the Engine class and Car and CarParts can be decoupled entirely as shown in figure 8.58.

```

class Car {
    private Engine engine;
public Car(Engine engine) {
    this.engine = engine;
}
public function drive() {
    this.engine.start();
}
}
///
Car car = new Car(parts.getEngine());

```

**Figure 8.57:** Removing Unnecessary Coupling

### 8.4.9 Action at a Distance

Action at a distance is a type of broken encapsulation where one part of the application can accidentally affect the way a different part of the program runs.

This usually happens due to global state as shown in figure 8.58.

```

class ShoppingBasket {
    public static double taxRate = 0.2;
public void getSubtotal() {
    //loop through items and calcualte subtotal
    }
public void getTotal() {
    return this.getSubTotal() * (1 + this.taxRate);
    }
}
///  

public void getBasketTotal(User user, ShoppingBasket shoppingBasket) {
    if (user.country == 'Ireland') {
        ShoppingBasket.taxRate = 0.23;
    }
}
///  

}

```

**Figure 8.58:** Global state and action at a distance

Because of the global taxRate variable, the entire application's taxRate has changed after the first user from Ireland places an order. This may cause bugs if when a user from another country places an order, the tax rate is not reset.

## **Appendix V. Paper: Seven Deadly Sins of Software Flexibility**

The paper below was presented at the China-Europe International Symposium on Software Engineering Education conference in 2017 (Athens). The paper is a cut down version of chapter 2 of this thesis to meet the conference's word limit.

# Seven Deadly Sins of Software Flexibility

Thomas Butler<sup>1</sup> and Mark Johnson<sup>1</sup>

<sup>1</sup>University of Northampton, Northampton, United Kingdom  
thomas.butler@northampton.ac.uk

**Abstract.** As software development techniques evolve, practices emerge which both help and hinder software development. These practices are often identified first by industry experts who work with large codebases in big teams. There are many software development techniques that have been labelled "bad practice" by these industry experts that aren't formally recognised in academia. This paper briefly describes some of these bad practices.

## 1. Introduction

When teaching programming the primary goal is giving students the tools they need to be able to write code. From the very basics of variables and `if` statements through to Objects and Classes, all the while the teaching focus tends to be on "how". "How to write a class", "How to write a static method" with less focus on arguably more important questions beginning with "Why": "Why do those methods belong in that class?", "Why did you use a static method there?", "Why should global variables be avoided?". These are questions which typically come up during industry code reviews[1-2].

Although there is some emphasis on code structure, usually it's only as far as Separation of Concerns (For example, MVC), there is often little to no discussion of best practices and the avoidance of known bad practices. The choices made by the developer to use one language tool over another can heavily impact the maintainability and flexibility of the final product.

Certain practices have been labeled as "bad" for many years. Practices such as *global variables* have been widely identified as impacting the flexibility of code going back at least as far as Wulf *et al*[3]. Others have highlighted problems with their use since[4-6].

Other bad practices have been identified as the complexity of programs and development methodology grows. For example Hevery[7], a programming coach at Google, has identified several bad practices which have emerged due to a shift towards Test-Driven-Development (TDD) among developers. These practices were still problematic before the widespread use of TDD, however because TDD requires an extra level of flexibility, practices which limit flexibility become apparent to the developer considerably faster than when using alternative development methods.

### 1.1 Case Study: Singleton Pattern

One such bad practice is the *Singleton Pattern*, which to most developers is regarded as bad practice[9]. Despite this being identified as a bad practice by developers at

IBM[10], Microsoft[11], Google[12] and by many other industry professionals[12-14]. The singleton is one of the most commonly discussed and derided "anti-patterns".

Despite the widespread derision of the pattern in industry, the singleton can still be found in academic teaching materials[15-17] and academic papers [18-19]. Even papers which specifically discuss the merits of design patterns[20-21] fail to mention that it's a widely discouraged pattern. The singleton is one of the most commonly described *bad practices* yet it still appears in academic materials.

There is a gap between industry and academia which this paper attempts to bridge by formally describing the bad practices in an academic setting.

This paper briefly outlines seven practices which have been identified by industry experts as "bad practice" in regards to software flexibility. Each identified practice is labeled as such because it makes the code difficult to test, adapt or change to meet new requirements. This paper does not attempt to weigh in on the pros/cons of flexibility[23] or possible performance/design trade-offs when building software in a flexible way.

## 2. Sins of Software Engineering



**Fig 1.** Seven Deadly Sins of Software Flexibility

The following are seven of the most commonly discussed bad practices, however this is not a comprehensive list and other developers may prioritise other bad practices.

### 2.1 Global/Static Variables

Global variables are widely labelled "bad practice" and have been for some time, for



example in 1999 Kernighan[23] wrote:

*Avoid global variables; wherever possible it is better to pass references to all data through function arguments*

And Hevery[24] states:

*I hope that by now most developers agree that global state should be treated like GOTO.*

Although "global variables are bad" is a common thing to hear, for novice developers, it's not immediately obvious why this is. The underlying issue is that global state can cause "action at a distance"[26]. By relying on shared state across the application, changes to this state can have knock on effects throughout the entire application. If, for example a global or static variable is used to store a database connection, a disconnection command in one part of the application will cause any other part of the program that needs an active connection to stop working.

## 2.2 Singleton

*for it is true that global variables are often demonised and more recently the Singleton has befallen the same fate.*

Knack-Nielsen[8]

The singleton has become regarded as an anti-pattern in the minds of most developers. This is for several reasons that don't apply to many other bad practices:

- The singleton was one of the patterns mentioned in two very popular and highly referenced books: *Design Patterns: Elements of Reusable Object-Oriented Software*[26] and *Patterns of Enterprise Application Architecture*[27]. This caused widespread knowledge of the pattern and it took some time until the issues it causes were documented.
- The pattern was given a formal name and is easy to identify
- The pattern has been labeled bad practice for over a decade [10]
- The problems it causes are severe compared to other more subtle patterns so developers much more quickly identify one of the many issues it introduces.

Like *Global Variables*, because of abundant use, problems associated with the pattern are very well documented.

## 2.3 Inheritance

Inheritance has been pointed at as a source of programming problems for a long time.

The popular book *Design Patterns: Elements of Reusable Object-Oriented Software* by Gamma *et al*[26] has a section on replacing inheritance with composition.

Holub[28] quotes a speech by James Gosling, the inventor of Java who said he would leave out inheritance if he had to design Java again.

The biggest problem with inheritance is the tight coupling it creates[30] as there is no way to substitute the parent class at runtime. From a TDD perspective, this means that testing the subclass also means testing the parent class. Instead, *is-a* relationships are better expressed as *has-a* relationships. A boat *has-a* engine, instead of a petrol powered boat *is-a* boat, an employee *has-a* job instead of an employee *is-a* person, a cat *has* warm blood, fur and paws instead of a cat *is-a* animal.

Inheritance also creates the the *Fragile Base Class* problem[31] where making a change to a child class can break functionality in the parent class and problems with inheritance in regards to encapsulation[26][32].

## 2.4 Using The New Keyword in Constructors

Tight coupling has been identified as a limitation on flexibility by numerous developers[32-33] over the last decade or so. It causes a problem because it makes it impossible to substitute one part of the system for another. [34][36] has identified a specific technique which introduces tight coupling that makes code hard to test, that is the new keyword.

If, when an object is instantiated, it instantiates 5 other objects it's impossible to isolate that initial object for testing purposes. If a unit test fails, there is no way to know whether the bug is in the object we're testing or one of the objects it instantiated. Instead, the objects should be instantiated once at the top level and passed in as constructor arguments. A practice known as *Dependency Injection*[36-37]. By injecting dependencies into a constructor, those dependencies can be mocked for testing purposes.

## 2.5 Service Locator

A service locator is a specific common example of a Law Of Demeter violation[39]. Flexibility is reduced as the service locator introduces coupling between the service locator and the code which uses it.

*there's no reason to ever use a Service Locator. There's always a better alternative that involves proper inversion of control.*

Seeman[39]

As with **New in constructor**, the solution is to inject dependences to the constructor.

## 2.6 Object Not Initliased After Constructor Finishes (Initialize/Set Methods)

Once the constructor has run, the object should be 100% configured. Several common bad practices have emerged which go against this but all lead to the same issue. The practices are:

*Initialize Methods* Some developers will create `initialize` methods[35] which are intended to be called immediately after the constructor to provide some additional setup.

*Setter Injection* which is used to provide dependencies using individual function calls rather than as part of the constructor[41].

In both instances the same problem is introduced: It's possible to have an object in an unusable state. If an object is passed into a method, calling a function on it may or may not work depending on whether other methods have been called on the object to set up the dependencies or initialize it with default values.

## 2.7 Use of Static Methods

Use of static methods always reduces flexibility by introducing tight coupling[42]. A

static method tightly couples the calling code to the specific class the method exists in.

A static method cannot be mocked for testing purposes[43] leading to difficult to test code. It's also impossible to substitute the method heavily reducing flexibility. `Math.abs(num)` will only work with some types, those which have a method in the `Math` class, to make the `abs()` method work with a new type (e.g. `BigInt`) the `Math` class must be amended with a new method. A better solution is polymorphism. Instead of a static method `num.abs()` would allow extending the code with new numeric types without needing to update the `Math` class.

Any value for `num` that was passed into a method would supply the `abs()` method which dealt with that particular type. Static methods always break encapsulation because they decouple the data from the methods working on the data.

### 3. Conclusion

This list of sins is incomplete and does not represent all the bad practices which are either commonly derided or widely discussed among industry professionals, however these are the most common. Although there is ample discussion among academics about global variables, the same rigor is not taken with other practices which are known to cause similar problems, which has a knock on effect to teaching and learning.

Moving forward the following steps can be taken to bridge the gap between industry professionals and academics:

1. Alongside exercises that ask students to write a program that performs a task, ask them to evaluate different implementations of the code.
2. Set assignments and exercises that focus on testing and maintaining existing code, with an emphasis on asking where improvements can be made and problems with current approaches
3. Shift the focus from exercises that result in dozens of small programs to building a single larger program over a course; the students will encounter these kind of problems themselves
4. Introduce the concept of *code reviews* to students during the academic course
5. Specifically teach how to identify bad practices and alternative approaches.
6. Develop tools that can detect flexibility limitations in source code. This could be done in a similar vein to the W3 HTML Validator[44], online accessibility checkers or code quality metric tools such as Scrutinizer[45] which check for problems such as unused variables, inaccessible code blocks but do not check for known bad practices such as singletons.
7. Finally, develop tools that can provide suggestions or automate fixes for removing bad practices from existing code.

### 4. References

1. Hevery, M. (2008) *Code Reviewers Guide* [online]. Available from: <http://misko.hevery.com/code-reviewers-guide/>
2. Gutha, S. (2015) *Code Review Checklist – To Perform Effective Code Reviews* [online].

Available from:

<http://www.evoketechnologies.com/blog/code-review-checklist-perform-effective-code-reviews/>

3. Wulf, W., Shaw, M. (1973) Global variables considered harmful. *ACM SIGPLAN Notices*, pp.28-34.
4. Ferreira, G. (2013) *Best C Coding Practices – Global variables* [online]. Available from: <http://guilhermemacielferreira.com/2013/06/01/best-c-coding-practices-global-variables/>
5. IBM, I. (2012) *Avoid modification of global and static variables* [online]. Available from: [http://pic.dhe.ibm.com/infocenter/idshelp/v117/index.jsp?topic=%2Fcom.ibm.dapip.doc%2Fids\\_dapip\\_0673.htm](http://pic.dhe.ibm.com/infocenter/idshelp/v117/index.jsp?topic=%2Fcom.ibm.dapip.doc%2Fids_dapip_0673.htm)
6. Crockford, D. (2006) *Global Domination* [online]. Available from: <http://www.yuiblog.com/blog/2006/06/01/global-domination/>
7. Hevery, M. (2008) *Guide: Writing Testable Code* [online]. Available from: <http://misko.hevery.com/code-reviewers-guide/>
8. Knack-Nielsen, T. (2008) *What's so bad about the Singleton?* [online]. Available from: <http://www.sitepoint.com/whats-so-bad-about-the-singleton/>
9. J., R. (2001) *Use your singletons wisely* [online]. Available from: <https://www.ibm.com/developerworks/library/co-single/>
10. Densmore, S. (2004) *Why Singletons Are Evil* [online]. Available from: <http://blogs.msdn.com/b/scottdensmore/archive/2004/05/25/140827.aspx>
11. Hevery, M. (2008) *Singletons are Pathological Liars* [online]. Available from: <http://misko.hevery.com/2008/08/17/singletons-are-pathological-liars/>
12. Geary, D. (2003) *Simply Singleton* [online]. Available from: <http://www.javaworld.com/article/2073352/core-java/simply-singleton.html>
13. Durand, W. (2013) *From STUPID to SOLID Code!* [online]. Available from: <http://williamdurand.fr/2013/07/30/from-stupid-to-solid-code/>
14. Martin, R. (2014) *SingletonVsJustCreateOne* [online]. Available from: <http://butunclebob.com/ArticleS.UncleBob.SingletonVsJustCreateOne>
15. Adamek, M. (2016) *Computer Science III Programming Patterns* [online]. Available from: <http://vega.cs.kent.edu/~mikhail/classes/cs3/>
16. Harle, R. (2016) *Object-Oriented Programming* [online]. Available from: <https://www.cl.cam.ac.uk/teaching/1516/OOProg/>
17. Tarr, B. (2013) *The Singleton Pattern* [online]. Available from: <http://www.dcs.bbk.ac.uk/~oded/OODP13/Sessions/Session6/Singleton.pdf>
18. Hamie, A. (2002) Pattern-based mapping of OCL specifications to JML contracts. *Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference*, pp.193-200. IEEE.
19. Delic, E., Schreiber, M., Hayek, A., Börcsök, J. (2013) Pattern-based mapping of OCL specifications to JML contracts. *Information & Communication Technology Electronics & Microelectronics (MIPRO)*, pp.85-90. IEEE.
20. Gui-lan, H., Wu, S., Yao, J. (2013) Application of design pattern in the JDBC programming. *8th International Conference on Computer Science & Education*, pp.85-90. IEEE.
21. Raja, W., Nirmala, K., Yao, J. (2016) Agile Development Methods for Online Training Courses Web Application Development. *International Journal of Applied Engineering Research* 11, pp.2601-2606. Research India Publications.
22. Eden, A., Tom, M. (2006) Measuring software flexibility. *IEE Software* 153, pp.133-126.
23. Kernighan, B. (1999) *The Practice of Programming* ISBN: 978-0201615869. Addison Wesley.
24. Hevery, M. (2008) *Top 10 things which make your code hard to test* [online]. Available from:



# Appendix VI. Meta-Analysis Raw Data

This appendix contains the raw data used for the meta-analyses in chapter 3 of this thesis.

## Singleton

URL	Describe	Example	Implications	Alternatives	Comparison code	Pros/Cons	Recommendation	Recommendation scale	Jadad
<a href="http://www.fssnip.net/7p/title/Singleton-Pattern">http://www.fssnip.net/7p/title/Singleton-Pattern</a>		1						3	1 -3
<a href="https://www.hackerrank.com/challenges/java-singleton/forum">https://www.hackerrank.com/challenges/java-singleton/forum</a>		1						3	1 -3
<a href="https://gist.github.com/mssola/6138155">https://gist.github.com/mssola/6138155</a>		1						3	1 -3
<a href="http://fortranwiki.org/fortran/show/Singleton+pattern">http://fortranwiki.org/fortran/show/Singleton+pattern</a>	1							3	1 -3
<a href="http://www.adam-bien.com/roller/abien/entry/singleton_pattern_in_es6_and">http://www.adam-bien.com/roller/abien/entry/singleton_pattern_in_es6_and</a>		1						3	1 -3
<a href="https://coderwall.com/p/iemfbg/objective-c-singleton-pattern-with-arc">https://coderwall.com/p/iemfbg/objective-c-singleton-pattern-with-arc</a>		1						3	1 -3
<a href="http://cruise.eecs.uottawa.ca/umple/SingletonPattern.html">http://cruise.eecs.uottawa.ca/umple/SingletonPattern.html</a>		1						3	1 -3
<a href="http://www.netobjectives.com/resources/books/design-patterns-explained/java-code-examples/chapter21">http://www.netobjectives.com/resources/books/design-patterns-explained/java-code-examples/chapter21</a>		1						3	1 -3
<a href="https://www.dotnetperls.com/singleton-static">https://www.dotnetperls.com/singleton-static</a>		1					1	2	2 -2
<a href="https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm">https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm</a>	1	1						3	2 -3
<a href="https://msdn.microsoft.com/en-gb/library/ff650316.aspx">https://msdn.microsoft.com/en-gb/library/ff650316.aspx</a>	1	1						3	2 -3
<a href="https://www.javaworld.com/article/2073352/core-java-simply-singleton.html">https://www.javaworld.com/article/2073352/core-java-simply-singleton.html</a>	1	1						3	2 -3
<a href="http://www.oodeesign.com/Singleton-Pattern.html">http://www.oodeesign.com/Singleton-Pattern.html</a>	1	1						3	2 -3
<a href="https://code.tutsplus.com/tutorials/android-design-patterns-the-singleton-pattern--cms-29153">https://code.tutsplus.com/tutorials/android-design-patterns-the-singleton-pattern--cms-29153</a>	1	1						3	2 -3
<a href="http://www.dofactory.com/javascript/Singleton-Design-Pattern">http://www.dofactory.com/javascript/Singleton-Design-Pattern</a>	1	1						3	2 -3
<a href="https://www.techopedia.com/definition/15830/Singleton">https://www.techopedia.com/definition/15830/Singleton</a>	1	1						3	2 -3
<a href="https://www.geeksforgeeks.org/Singleton-Design-Pattern/">https://www.geeksforgeeks.org/Singleton-Design-Pattern/</a>	1	1						3	2 -3
<a href="https://dzone.com/articles/Singleton-pattern-a-deep-dive">https://dzone.com/articles/Singleton-pattern-a-deep-dive</a>	1	1						3	2 -3
<a href="https://www.javatpoint.com/Singleton-Design-Pattern-in-Java">https://www.javatpoint.com/Singleton-Design-Pattern-in-Java</a>	1	1						3	2 -3
<a href="https://developer.salesforce.com/page/Apex_Design_Patterns_-_Singleton">https://developer.salesforce.com/page/Apex_Design_Patterns_-_Singleton</a>	1	1						3	2 -3
<a href="https://fullstack-developer.academy/Singleton-Pattern-in-TypeScript/">https://fullstack-developer.academy/Singleton-Pattern-in-TypeScript/</a>	1	1						3	2 -3
<a href="http://jargon.js.org/_glossary/SINGLETON_PATTERN.md">http://jargon.js.org/_glossary/SINGLETON_PATTERN.md</a>	1	1						3	2 -3
<a href="https://djalbornasevic.com/posts/9-ruby-singleton-pattern">https://djalbornasevic.com/posts/9-ruby-singleton-pattern</a>	1	1						3	2 -3
<a href="http://marcio.io/2015/07/Singleton-Pattern-in-Go/">http://marcio.io/2015/07/Singleton-Pattern-in-Go/</a>	1	1						3	2 -3
<a href="https://basarat.gitbooks.io/typescript/docs/tips/Singleton.html">https://basarat.gitbooks.io/typescript/docs/tips/Singleton.html</a>	1	1						3	2 -3
<a href="http://www.vogella.com/tutorials/DesignPatternsSingleton/article.html">http://www.vogella.com/tutorials/DesignPatternsSingleton/article.html</a>	1	1						3	2 -3
<a href="https://www.nada.kth.se/kurser/kth/2D1359/01-02/contents/forelasningar/lecture10.ppt">https://www.nada.kth.se/kurser/kth/2D1359/01-02/contents/forelasningar/lecture10.ppt</a>	1	1						3	2 -3
<a href="https://www.linkedin.com/pulse/Singleton-pattern-eager-lazy-enum-ramasamy-kasiviswanathan">https://www.linkedin.com/pulse/Singleton-pattern-eager-lazy-enum-ramasamy-kasiviswanathan</a>	1	1						3	2 -3
<a href="https://www.avajava.com/tutorials/lessons/Singleton-Pattern.html">https://www.avajava.com/tutorials/lessons/Singleton-Pattern.html</a>	1	1						3	2 -3
<a href="http://ftp.mak.com/out/classdocs/rforces4.4.1/classref/rvrv_the_rooted_singleton_pattern.html">http://ftp.mak.com/out/classdocs/rforces4.4.1/classref/rvrv_the_rooted_singleton_pattern.html</a>	1	1						3	2 -3
<a href="http://www.galloway.me.uk/tutorials/Singleton-Classes/">http://www.galloway.me.uk/tutorials/Singleton-Classes/</a>	1	1						3	2 -3
<a href="https://wiki.base22.com/btg/Singleton-Pattern-3459.html">https://wiki.base22.com/btg/Singleton-Pattern-3459.html</a>	1	1						3	2 -3
<a href="https://coffeescript-cookbook.github.io/chapters/design_patterns/Singleton">https://coffeescript-cookbook.github.io/chapters/design_patterns/Singleton</a>	1	1						3	2 -3
<a href="https://blogs.sap.com/2012/04/06/Singleton-Pattern-in-ABAP/">https://blogs.sap.com/2012/04/06/Singleton-Pattern-in-ABAP/</a>	1	1						3	2 -3
<a href="http://docs.ros.org/indigo/api/typelib/html/group__singleton.html">http://docs.ros.org/indigo/api/typelib/html/group__singleton.html</a>	1	1						3	2 -3
<a href="https://learnswithbob.com/course/object-oriented-swift/Singleton-Pattern.html">https://learnswithbob.com/course/object-oriented-swift/Singleton-Pattern.html</a>	1	1						3	2 -3
<a href="https://www.visual-paradigm.com/tutorials/SingletonPattern.jsp">https://www.visual-paradigm.com/tutorials/SingletonPattern.jsp</a>	1	1						3	2 -3
<a href="https://medium.freecodecamp.org/lets-talk-about-you-and-the-singleton-design-pattern-bb2e160fa952">https://medium.freecodecamp.org/lets-talk-about-you-and-the-singleton-design-pattern-bb2e160fa952</a>	1	1						3	2 -3
<a href="https://www.htmlgoodies.com/beyond/javascript/implementing-the-singleton-design-pattern-in-javascript.html">https://www.htmlgoodies.com/beyond/javascript/implementing-the-singleton-design-pattern-in-javascript.html</a>	1	1						3	2 -3
<a href="http://web.science.mq.edu.au/~mattr/courses/object_oriented_development_practices/6/notes.html">http://web.science.mq.edu.au/~mattr/courses/object_oriented_development_practices/6/notes.html</a>	1	1						3	2 -3
<a href="http://www.jot.fm/issues/issue_2007_03/column2/">http://www.jot.fm/issues/issue_2007_03/column2/</a>	1	1						3	2 -3
<a href="https://itexico.com/blog/bid/99247/Software-Development-The-Singleton-Design-Pattern-and-other-Creational-Patterns">https://itexico.com/blog/bid/99247/Software-Development-The-Singleton-Design-Pattern-and-other-Creational-Patterns</a>	1	1						3	2 -3
<a href="http://code.activestate.com/recipes/52558-the-singleton-pattern-implemented-with-python/">http://code.activestate.com/recipes/52558-the-singleton-pattern-implemented-with-python/</a>	1	1						3	2 -3
<a href="https://alvinalexander.com/scala/how-to-implement-singleton-pattern-in-scala-with-object">https://alvinalexander.com/scala/how-to-implement-singleton-pattern-in-scala-with-object</a>	1	1						3	2 -3
<a href="http://wiki.unity3d.com/index.php/Singleton">http://wiki.unity3d.com/index.php/Singleton</a>	1	1						3	2 -3
<a href="https://locklessinc.com/articles/Singleton_Pattern/">https://locklessinc.com/articles/Singleton_Pattern/</a>	1	1						3	2 -3
<a href="http://www.tothenew.com/blog/Singleton-Pattern-with-javascript/">http://www.tothenew.com/blog/Singleton-Pattern-with-javascript/</a>	1	1						3	2 -3
<a href="https://sweetcode.io/Singleton-Design-Pattern-using-Java/">https://sweetcode.io/Singleton-Design-Pattern-using-Java/</a>	1	1						3	2 -3
<a href="http://elbenshira.com/blog/Singleton-Pattern-in-Python/">http://elbenshira.com/blog/Singleton-Pattern-in-Python/</a>	1	1						3	2 -3
<a href="https://php.earth/docs/php/ref/ooop/design_patterns/Singleton">https://php.earth/docs/php/ref/ooop/design_patterns/Singleton</a>	1	1						3	2 -3
<a href="https://www.ibm.com/developerworks/library/fj-dcl/">https://www.ibm.com/developerworks/library/fj-dcl/</a>	1	1						3	2 -3
<a href="https://amir.rachum.com/blog/2012/04/26/implementing-the-singleton-pattern-in-python/">https://amir.rachum.com/blog/2012/04/26/implementing-the-singleton-pattern-in-python/</a>	1	1						3	2 -3
<a href="https://krakendev.io/blog/the-right-way-to-write-a-singleton">https://krakendev.io/blog/the-right-way-to-write-a-singleton</a>	1	1						3	2 -3
<a href="http://moddb.wikia.com/wiki/Singleton_Pattern">http://moddb.wikia.com/wiki/Singleton_Pattern</a>	1	1						3	2 -3
<a href="http://rcardin.github.io/design/programming/2015/07/03/the-good-the-bad-and-the-singleton.html">http://rcardin.github.io/design/programming/2015/07/03/the-good-the-bad-and-the-singleton.html</a>	1	1						3	2 -3
<a href="http://csharpindepth.com/Articles/General/Singleton.aspx">http://csharpindepth.com/Articles/General/Singleton.aspx</a>	1	1	1					3	3 -3
<a href="https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples">https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples</a>	1	1	1					3	3 -3
<a href="http://best-practice-software-engineering.info/tuwienn.ac.at/patterns/Singleton.html">http://best-practice-software-engineering.info/tuwienn.ac.at/patterns/Singleton.html</a>	1	1	1					3	3 -3
<a href="https://refactoring.guru/design-patterns/Singleton">https://refactoring.guru/design-patterns/Singleton</a>	1	1			1			3	3 -3
<a href="https://www.techrepublic.com/blog/software-engineer/using-the-singleton-pattern-in-java/">https://www.techrepublic.com/blog/software-engineer/using-the-singleton-pattern-in-java/</a>	1	1	1					3	3 -3
<a href="https://www.gamasutra.com/blogs/MattChristian/20101013/88205/OOPsie_Patterns_The_Singleton_Pattern.php">https://www.gamasutra.com/blogs/MattChristian/20101013/88205/OOPsie_Patterns_The_Singleton_Pattern.php</a>	1	1	1					3	3 -3
<a href="https://pdfs.semanticscholar.org/presentation/f1dc667b86aac3f5db63c26a67d30d586d2244a6.pdf">https://pdfs.semanticscholar.org/presentation/f1dc667b86aac3f5db63c26a67d30d586d2244a6.pdf</a>	1	1	1					3	3 -3
<a href="https://www.implementingquantlib.com/2017/09/odds-and-ends-singleton.html">https://www.implementingquantlib.com/2017/09/odds-and-ends-singleton.html</a>	1	1	1					3	3 -3
<a href="https://8thlight.com/blog/josh-cheek/2012/10/20/implementing-and-testing-the-singleton-pattern-in-ruby.html">https://8thlight.com/blog/josh-cheek/2012/10/20/implementing-and-testing-the-singleton-pattern-in-ruby.html</a>	1	1	1					4	3 -4
<a href="http://csc.columbusstate.edu/woolbright/java/Singleton.html">http://csc.columbusstate.edu/woolbright/java/Singleton.html</a>	1	1	1		1			3	4 -3
<a href="https://www.codeproject.com/Articles/307233/Singleton-Pattern-Positive-and-Negative-Aspects">https://www.codeproject.com/Articles/307233/Singleton-Pattern-Positive-and-Negative-Aspects</a>	1	1	1			1		3	4 -3
<a href="https://sourcemaking.com/design_patterns/Singleton">https://sourcemaking.com/design_patterns/Singleton</a>	1	1	1				1	4	4 -4
<a href="https://www.gofpatterns.com/design-patterns/module3/consequences-effects-singleton-pattern.php">https://www.gofpatterns.com/design-patterns/module3/consequences-effects-singleton-pattern.php</a>	1	1	1				1	4	4 -4
<a href="http://wiki.c2.com/?SingletonPattern">http://wiki.c2.com/?SingletonPattern</a>	1	1	1				1	4	4 -4
<a href="https://medium.com/if-let-swift-programming/the-swift-singleton-pattern-442124479b19">https://medium.com/if-let-swift-programming/the-swift-singleton-pattern-442124479b19</a>	1	1	1				1	4	4 -4
<a href="http://2ality.com/2011/04/Singleton-pattern-in-javascript-not.html">http://2ality.com/2011/04/Singleton-pattern-in-javascript-not.html</a>	1	1	1				1	4	4 -4
<a href="https://reftimov.com/Singleton-Pattern">https://reftimov.com/Singleton-Pattern</a>	1	1	1				1	4	4 -4
<a href="https://addyosmani.com/resources/essentialjsdesignpatterns/book/">https://addyosmani.com/resources/essentialjsdesignpatterns/book/</a>	1	1				1	1	4	4 -4
<a href="https://phpenthusiast.com/blog/the-singleton-design-pattern-in-php">https://phpenthusiast.com/blog/the-singleton-design-pattern-in-php</a>	1	1	1				1	4	4 -4
<a href="https://www.perl.com/article/52/2013/12/11/Implementing-the-singleton-pattern-in-Perl/">https://www.perl.com/article/52/2013/12/11/Implementing-the-singleton-pattern-in-Perl/</a>	1	1	1				1	4	4 -4
<a href="http://www.bogotobogo.com/DesignPatterns/Singleton.php">http://www.bogotobogo.com/DesignPatterns/Singleton.php</a>	1	1			1			5	4 -5











URL	Describe	Example	Implications	Alternatives	Comparison code	Pros/Cons	Recommendation	Recommendation scale	Jadad
https://code-maven.com/slides/dart-programming/extending-class		1						3	1 -3
https://www.mathworks.com/help/matlab/matlab_oo/subclassing-multiple-classes.html		1						3	1 -3
https://www.integralist.co.uk/posts/object-oriented-design/		1						3	1 -3
https://www.brandonsavage.net/five-tips-to-make-good-object-oriented-code-better/		1						5	1 -5
https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)		1	1					3	2 -3
https://msdn.microsoft.com/en-us/library/ms973803.aspx		1	1					3	2 -3
https://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html		1	1					3	2 -3
https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance		1	1					3	2 -3
https://www.tutorialspoint.com/cppplusplus/cpp_inheritance.htm		1	1					3	2 -3
https://www.tutorialspoint.com/java/java_inheritance.htm		1	1					3	2 -3
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/inheritance		1	1					3	2 -3
https://www.digitalocean.com/community/tutorials/understanding-class-inheritance-in-python-3		1	1					3	2 -3
https://www.adobe.com/devnet/actionsript/learning/ooop-concepts/inheritance.html		1	1					3	2 -3
https://stackoverflow.com/ooop-concept-inheritance/		1	1					3	2 -3
https://www.python-course.eu/python3_inheritance.php		1	1					3	2 -3
http://www.cplusplus.com/doc/tutorial/inheritance/		1	1					3	2 -3
https://www.geeksforgeeks.org/inheritance-in-c/		1	1					3	2 -3
https://jscript.info/class-inheritance		1	1					3	2 -3
https://www.jesshamrick.com/2011/05/18/an-introduction-to-classes-and-inheritance-in-python/		1	1					3	2 -3
https://hackernoon.com/java-for-humans-class-inheritance-d82a357b2659		1	1					3	2 -3
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Inheritance		1	1					3	2 -3
https://beginnersbook.com/2013/03/inheritance-in-java/		1	1					3	2 -3
https://medium.com/@isaacjumba/overview-of-inheritance-interfaces-and-abstract-classes-in-java-3fe22404baf8		1	1					3	2 -3
https://processing.org/examples/inheritance.html		1	1					3	2 -3
https://www.guru99.com/java-class-inheritance.html		1	1					3	2 -3
http://www.learnjavaonline.org/en/Inheritance		1	1					3	2 -3
http://php.net/manual/en/language.oop5.inheritance.php		1	1					3	2 -3
https://martinfowler.com/eaCatalog/classTableInheritance.html		1	1					3	2 -3
https://launchschool.com/books/oo_ruby/read/inheritance		1	1					3	2 -3
https://www.datamentor.io/r-programming/inheritance		1	1					3	2 -3
https://www.programiz.com/kotlin-programming/inheritance		1	1					3	2 -3
http://www.numl.fnl.gov/offline_software/srt_public_context/WebDocs/Companion/cox_crib/inheritance.html		1	1					3	2 -3
http://math.hws.edu/eck/cs124/javanotes5/c5/s5.html		1	1					3	2 -3
https://docs.swift.org/swift-book/LanguageGuide/Inheritance.html		1	1					3	2 -3
https://crystal-lang.org/docs/syntax_and_semantics/inheritance.html		1	1					3	2 -3
http://allaboutscala.com/tutorials/chapter-3-beginner-tutorial-using-classes-scala/scala-extend-class/		1	1					3	2 -3
https://scotch.io/tutorials/demystifying-es6-classes-and-prototypal-inheritance		1	1					3	2 -3
https://www.csie.ntu.edu.tw/~sylee/courses/dlps/class.htm		1	1					3	2 -3
https://en.wikibooks.org/wiki/C%2B%2B_Programming/Classes/Inheritance		1	1					3	2 -3
http://propelorm.org/Propel/documentation/09-inheritance.html		1	1					3	2 -3
https://unity3d.com/learn/tutorials/topics/scripting/inheritance		1	1					3	2 -3
https://orientdb.com/docs/2.1.x/Inheritance.html		1	1					3	2 -3
https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.cbcbx01/inher.htm		1	1					3	2 -3
https://kotlinlang.org/docs/reference/classes.html		1	1					3	2 -3
https://docs.python.org/3/tutorial/classes.html		1	1					3	2 -3
http://people.cs.aau.dk/~normark/ooop-csharp/html/notes/inheritance_themes-inheritance-csharp.html		1	1					3	2 -3
https://doc.winddev.com/76010007&name=class_inheritance		1	1					3	2 -3
https://fsharpforfunandprofit.com/posts/inheritance/		1	1					3	2 -3
https://www.lua.org/pil/16.2.html		1	1					3	2 -3
https://www.mq5.com/en/docs/basis/ooop/inheritance		1	1					3	2 -3
https://www.gnu.org/software/guile/manual/html_node/Inheritance.html		1	1					3	2 -3
https://en.cppreference.com/w/cpp/language/derived_class		1	1					3	2 -3
https://www.sapien.com/blog/2016/03/16/inheritance-in-powershell-classes/		1	1					3	2 -3
https://torquemag.io/2016/06/introduction-class-inheritance-ooop-php/		1	1					3	2 -3
https://www.baeldung.com/java-inheritance		1	1					3	2 -3
https://medium.freecodecamp.org/multiple-inheritance-in-c-and-the-diamond-problem-7c12a9ddbcec		1	1					3	2 -3
https://www.universalclass.com/articles/computers/object-oriented-concepts-inheritance-and-polymorphism-in-c-programming.htm		1	1					3	2 -3
https://www.typescriptlang.org/docs/handbook/classes.html		1	1					3	2 -3
https://usclab.github.io/cereal/inheritance.html		1	1					3	2 -3
https://johnresig.com/blog/simple-javascript-inheritance/		1	1					3	2 -3
https://www.calliope.com/kotlin-inheritance/		1	1					3	2 -3
https://octave.org/doc/v4.0.0/Inheritance_and_Aggregation.html		1	1					3	2 -3
ftp://ftp.desy.de/pub/courses/cc/text/tutorial1/html/chap07.html		1	1					3	2 -3
https://community.bistudio.com/wiki/Class_Inheritance		1	1					3	2 -3
https://community.bistudio.com/wiki/Class_Inheritance		1	1					3	2 -3
https://thepythonguru.com/python-inheritance-and-polymorphism/		1	1					3	2 -3
https://www.alphaoftware.com/documentation/pages/Guides/Xbasic/Subclasses%20and%20Inheritance.xml		1	1					3	2 -3
https://www.hackingwithswift.com/sixty/8/2/class-inheritance		1	1					3	2 -3
https://www.cs.bu.edu/teaching/cpp/inheritance/intro/		1	1					3	2 -3
https://ruby-doc.org/docs/ruby-doc-bundle/UsersGuide/rg/inheritance.html		1	1					3	2 -3
https://www.andrew.cmu.edu/course/15-121/lectures/Inheritance/inheritance.html		1	1					3	2 -3
https://www.eiffel.org/doc/solutions/Inheritance		1	1					3	2 -3
https://linuxconfig.org/python-inheritance		1	1					3	2 -3
https://haxe.org/manual/types-class-inheritance.html		1	1					3	2 -3
https://v1.realworldocaml.org/v1/en/html/classes.html		1	1					3	2 -3
http://esug.org/data/Old/lbm/tutorial/CHAP6.HTML		1	1					3	2 -3
https://www.imo.umontreal.ca/~pift1025/bigjava/Ch13/ch13.html		1	1					3	2 -3
https://en.wikiversity.org/wiki/C%2B%2B/Classes_and_Inheritance		1	1					3	2 -3
https://docs.objectbox.io/advanced/entity-inheritance		1	1					3	2 -3
https://www.hackerrank.com/challenges/30-inheritance/tutorial		1	1					3	2 -3
https://phpenthusiast.com/object-oriented-php-tutorials/inheritance-in-object-oriented-php		1	1					3	2 -3
http://www.peachpit.com/articles/article.aspx?p=2468332&seqNum=6		1	1					3	2 -3
https://cran.r-project.org/web/packages/R6/vignettes/Introduction.html		1	1					3	2 -3
https://www.sitepoint.com/understanding-ecmascript-6-class-inheritance/		1	1					3	2 -3
http://prototyeps.org/learn/class-inheritance		1	1					3	2 -3
https://doc.zeroc.com/ice/3.6/the-slice-language/classes/class-inheritance-semantics		1	1					3	2 -3
https://www.accelerate.com/blog/javascript-es6-classes-and-prototype-inheritance-part-1-of-2/		1	1					3	2 -3
https://www.linuxtopia.org/online_books/programming_books/thinking_in_java/TJ308_006.htm		1	1			1		3	2 -3
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/		1	1					3	2 -3
https://javascriptweblog.wordpress.com/2010/12/22/delegation-vs-inheritance-in-javascript/		1	1					3	2 -3
http://www.karthikscorner.com/sharepoint/design-patterns-composition-vs-inheritance/		1	1					3	2 -3
http://www.web-feats.com/classes/javaprogram/lessons/ooop/ih_vs_comp.htm		1	1					3	2 -3
http://ice-web.cc.gatech.edu/ce211/static/javaReview-RU/OOBasics/ooAssocVsInherit.html		1	1				1	4	2 -4
https://www.mimuw.edu.pl/~sl/teaching/00_01/Delfin_EC/Patterns/InheritanceVsComposition.htm		1	1				1	4	2 -4



Table with columns: URL, Describe, Example, Implications, Alternatives, Comparison code, Pros/Cons, Recommendation, Recommendation scale, Jadad. The table contains numerous rows of links related to service locator patterns and dependency injection.

Static Methods

Table with columns: URL, Describe, Example, Implications, Alternatives, Comparison code, Pros/Cons, Recommendation, Recommendation scale, Jadad. The table contains numerous rows of links related to static methods in programming.



https://knpuiversity.com/screencast/symfony-fundamentals/logger-trait	1	1	1					3	3	-3	
https://javarevisited.blogspot.com/2012/11/difference-between-setter-injection-vs-constructor-injection-spring-framework.html	1		1	1				3	3	-3	
http://www.techbloglife.com/setter-injection-overrides-the-values-injected-by-constructor-injection-in-spring/	1	1	1					3	3	-3	
http://websystique.com/spring/spring-dependency-injection-example-with-constructor-and-property-setter-xml-example/	1	1	1					3	3	-3	
https://antonlogoncalves.org/2011/05/03/injection-with-cdi-part-ii/	1	1		1				3	3	-3	
http://javaeasy.weebly.com/dependency-injections.html	1	1		1				3	3	-3	
https://javabeginnerstutorial.com/spring-framework-tutorial/spring-setter-dependency-injection-using-annotation/	1	1		1	1			3	3	-3	
http://www.vogella.com/tutorials/SpringDependencyInjection/article.html	1	1		1				3	3	-3	
https://java2blog.com/dependency-injection-via-setter-method/	1	1		1				3	3	-3	
https://autofacn.readthedocs.io/en/latest/register/prop-method-injection.html	1	1		1				3	3	-3	
https://www.java4coding.com/contents/spring/05setterInjection.html	1	1		1				3	3	-3	
https://www.borajl.com/spring-dependency-injection-example-with-annotation	1	1		1				3	3	-3	
http://coders-kitchen.com/2015/01/05/dependency-injection-field-vs-method/	1		1	1				3	3	-3	
http://codegeekslab.com/injection-of-set-java-collection-using-setter-and-constructor-dependency-injection/	1	1		1				3	3	-3	
https://gerardnico.com/code/design_pattern/injection	1	1		1				3	3	-3	
https://stackify.com/dependency-injection-c-sharp/	1		1	1				3	3	-3	
http://docs.drush.org/en/master/dependency-injection/	1	1		1				3	3	-3	
https://www.vodori.com/an-introduction-into-springs-dependency-injection/			1	1				3	3	-3	
http://static.javado.io/org/mockito/mockito-core/2.3.4/org/mockito/InjectMocks.html	1	1		1				3	3	-3	
https://javaranch.com/journal/200709/dependency-injection-unit-testing.html	1	1		1				3	3	-3	
http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/dependency_injection.html	1	1		1				3	3	-3	
http://s2container.seasar.org/en/DIContainer.html#SetterInjection	1	1		1				3	3	-3	
https://www.info-world.com/article/2974298/application-architecture-exploring-the-dependency-injection-principle.html	1	1		1				3	3	-3	
http://www.baslv.com/psd/blog/2009/java-based-configuration-of-spring-dependency-injection	1	1		1				3	3	-3	
https://www.devbridge.com/articles/dependency-injection-in-javascript/#	1	1		1				3	3	-3	
https://docs.oracle.com/javae/6/tutorial/doc/bncjk.html	1	1		1				3	3	-3	
https://blog.michalszalowski.com/java/dependency-injection-with-the-spring-framework-setter-injection-vs-constructor-injection/	1	1		1				3	3	-3	
https://symfony.com/doc/current/service_container/calls.html	1	1		1		1		5	3	-5	
http://www.devx.com/dotnet/Article/34066/0/page/2	1		1	1		1		1	4	-1	
https://www.java4s.com/spring/difference-between-setter-injection-and-constructor-injection/	1	1	1	1				3	4	-3	
https://javabeginnerstutorial.com/spring-framework-tutorial/spring-setter-dependency-injection-using-annotation/	1	1		1	1		1	3	4	-3	
https://blog.marcnurl.com/field-injection-is-not-recommended/	1	1	1	1				3	4	-3	
http://xunitpatterns.com/Dependency%20Injection.html	1	1	1	1				3	4	-3	
https://www.lucius.digital/blog/dependency-injection-drupal-8-introduction	1	1	1	1				3	4	-3	
http://jayroman.com/blog/finding-a-real-world-use-case-for-setter-injection-in-symfony2	1	1	1	1				4	4	-4	
https://documentation.help/MS-Enterprise-Library-5.0/EntLib50_135fa151-4553-43bf-87c2-e71935a1075a.html	1		1	1	1		1	4	4	-4	
http://blog.schauerhaft.de/2012/06/05/repeat-after-me-setter-injection-is-a-symptom-of-design-problems/	1		1	1			1	5	4	-5	
http://thesolidphp.com/constructor-vs-setter-dependency-injection/	1		1	1			1	5	4	-5	
https://codeopinion.com/throw-out-your-dependency-injection-container/	1	1		1			1	5	4	-5	
https://www.yegor256.com/2014/10/03/di-containers-are-evil.html	1		1	1			1	5	4	-5	
https://www.masterzendframework.com/zend-framework/easy-setter-injection-in-zend-framework-2/	1	1	1	1				4	5	-4	
http://fabien.potencier.org/what-is-dependency-injection.html	1	1		1	1		1	4	5	-4	
http://picocontainer.com/setter-injection.html	1	1	1	1				5	5	-5	
https://richardmiller.co.uk/2014/03/12/avoiding-setter-injection/	1	1	1	1			1	5	5	-5	
https://symfony.com/doc/current/service_container/injection_types.html	1	1	1	1	1	1		3	6	-3	
https://www.vojtechruzicka.com/field-dependency-injection-considered-harmful/	1	1	1	1	1	1		3	6	-3	
https://www.c-sharpcorner.com/UploadFile/ff2f08/dependency-injection-pattern/	1	1	1	1	1	1		3	6	-3	
http://umeshspring.blogspot.com/	1	1	1	1	1	1		3	6	-3	
https://help.semmie.com/wiki/display/JAVA/Use+setter+injection+instead+of+constructor+injection	1	1	1	1	1	1	1	1	7	-1	
https://brandonhilkert.com/blog/a-ruby-refactor-exploring-dependency-injection-options/	1	1	1	1	1	1	1	1	7	-1	
https://martinfowler.com/articles/injection.html#SetterInjectionWithSpring	1	1	1	1	1	1	1	1	4	7	-4
https://sergeyzhuk.me/2017/04/25/di-constructor-vs-setter/	1	1	1	1	1	1	1	1	4	7	-4
https://fr.je/constructor-injection-vs-setter-injection.html	1	1	1	1	1	1	1	1	4	7	-4
https://steveschols.wordpress.com/2012/06/05/i-was-wrong-constructor-vs-setter-injection/	1	1	1	1	1	1	1	1	4	7	-4
http://codebetter.com/jeremymiller/2008/10/09/setter-injection-in-structuremap-2-5/	1	1	1	1	1	1	1	1	4	7	-4
https://stormpath.com/blog/spring-boot-dependency-injection	1	1	1	1	1	1	1	1	4	7	-4
http://jeffbelback.me/posts/2014/08/12/SetterInjection/	1	1	1	1	1	1	1	1	4	7	-4
https://yobrieca.se/blog/2012/08/20/dependencies-in-javascript-constructor-setter-or-global/	1	1	1	1	1	1	1	1	4	7	-4
https://coderoncode.com/dependency-injection/design-patterns/programming/php/development/2014/01/06/dependency-injection-php.html	1	1	1	1	1	1	1	1	4	7	-4
https://jgassistant.org/context-dependency-injection-are-you-doing-it-right/	1	1	1	1	1	1	1	1	4	7	-4
https://spring.io/blog/2007/07/11/setter-injection-versus-constructor-injection-and-the-use-of-required/	1	1	1	1	1	1	1	1	5	7	-5
https://dzone.com/articles/constructor-injection-vs-0	1	1	1	1	1	1	1	1	5	7	-5
https://github.com/ninject/Ninject/wiki/Injection-Patterns	1	1	1	1	1	1	1	1	5	7	-5
https://kinbiko.com/java/dependency-injection-patterns	1	1	1	1	1	1	1	1	5	7	-5
http://misko.hevery.com/2009/02/19/constructor-injection-vs-setter-injection/	1	1	1	1	1	1	1	1	5	7	-5
https://alankent.me/2015/03/03/why-i-prefer-constructor-injection-over-setter-or-property-injection/	1	1	1	1	1	1	1	1	5	7	-5
http://royvanrijn.com/blog/2010/09/setter-vs-constructor-injection/	1	1	1	1	1	1	1	1	5	7	-5
https://testing.googleblog.com/2009/02/constructor-injection-vs-setter.html	1	1	1	1	1	1	1	1	5	7	-5
https://justin.abrah.ms/misc/an-overview-of-guice-java-dependency-injection.html	1	1	1	1	1	1	1	1	5	7	-5
http://evan.bottch.com/2009/02/03/setter-injection-sucks/	1	1	1	1	1	1	1	1	5	7	-5

## **Appendix VII. Paper: A Methodology for Performing Meta-analyses of Developers Attitudes Towards Programming Practices**

This paper describes the methodology for performing meta-analyses described in chapter 3 and was presented at the Proceedings of the 2019 Computing Conference (London). The paper is a cut down version of the first half of chapter 3 of this thesis to meet the conference's word limit.

*Note: the raw data was also provided as part of this paper as an appendix, as this raw data is already in this document as appenix IV, it has been omitted to avoid duplication.*



# A methodology for performing meta-analyses of developers attitudes towards programming practices

**Abstract.** Programming practices are often labelled "best practice" and "bad practice" by developers. This label can be subjective but we can see trends among developers. A methodology for performing meta-analyses of articles discussing any given practice was created to determine programmers overall attitudes towards any given practice while accounting for factors such as whether they considered alternative approaches.

## Introduction

Programming practices can often be described as *bad practice* or *best practice* depending on whether they have a positive or negative impact on the maintainability of the code in which they are used[1].

For software developers looking for information regarding any given programming practice, sources will vary in their level of detail. For example, a manual page will demonstrate how to use a practice but will not weigh in on the discussion of when or if the practice should be used. Opinion pieces may go into significantly more detail with discussions about pros/cons of the practice, where it's applicable and alternative approaches that can be used to solve the same problem.

If one article labels a programming practice "bad practice" and another "good practice" which should the reader believe?

The level of detail of an article can be used to determine academic rigour. An article suggesting to use the practice but without discussing alternative approaches is not making as strong a case for its use as a similar article which compares the practice to alternatives and explains why one approach is preferred over others.

A scoring system will be created to grade articles on their academic rigour. Once articles are graded, it will be possible to compare two or more articles based on their academic rigour and then perform a meta-analysis of any number of articles discussing a specific bad practice.

If academic rigour were ignored and a meta-analysis carried out using a simple tally of articles with positive/negative/neutral opinions a different conclusion may be drawn compared with an analysis including academic rigour. Academically rigorous articles may be more or less likely to have a favourable opinion of the practice.

## Aims and Objectives

1. Create a scoring system which can be used to:
  1. Grade the analytic rigour of an article/book/paper discussing a particular programming practice.
  2. Compare the academic rigour of different articles for the purposes of meta-analysis.
  3. Compare the overall quality of discussions about a specific

programming practice.

2. Calculating the score should not require reading the article in detail to calculate the score and anything used to calculate the score should be a binary choice.
3. With a scoring system in place, perform proof-of-concept meta-analyses on practices which are well known to be described as "good" and "bad" to demonstrate that the meta-analysis framework is fit for purpose.

## Methodology

### 1. Metric for comparing analytical rigour in programming articles

Differing methodological rigor in sources is a problem which exists when doing any kind of meta-analysis. When performing meta-analysis of clinical trials the Cochrane Collaboration[2] consider methodological rigour an important part of their meta-analysis.

Rather than simply counting the number of trials which show a positive outcome and counting the number of trials which show a negative outcome, they weigh the trials based on methodological rigour. For example in a meta-analysis of a drug they may find that 3 trials show that it is an effective treatment and 8 which say that it is not. Instead of simply counting the numbers on each side, they look at the academic rigour of each study and use that as a factor when building their conclusion of the overall efficacy of the treatment.

In a meta-analysis of the efficacy of homeopathic treatments[3] they found that trials of homeopathy with a poor methodology are much more likely to show a positive outcome whereas trials with a robust methodology are much more likely to conclude that homeopathy is no better than placebo.

This is because methodological rigour can affect the outcome. For example, by putting the most healthy patients in the experimental group and putting the least healthy patients in the control group it's likely that the experimental group will see significant improvement over the control group regardless of whether the drug being tested has any effect[4].

For programming articles, academic rigour can be plotted against whether the article recommends using or avoiding the practice to create a meta-analysis in a similar manner.

It should be possible to draw conclusions such as *as an article's academic rigour increases, it is more likely to recommend using the practice in question*

The created metric was based on the Jadad Scale[5] used for analysis of clinical trials in medicine. The Jadad Scale is a 5 point scale using a 3 question questionnaire which can be used to quickly assess the methodological rigour used in a clinical trial. The questions asked are: *Was the study described as randomized?*, *Was the study described as double blind?* and *Was there a description of withdrawals and dropouts?*. These are then used to generate a score from zero (very poor) to five (rigorous). By citation count the Jadad Scale is the most widely used method of comparing clinical trials in the world[6].

As the Jadad Scale is not applicable for anything other than clinical trials, a new metric was created based on the principles of the Jadad scale to be used in determining the academic rigour of any given article about a programming practice. A seven point scale was chosen with a point awarded if the article does each of the following:

1. Describes how to use the practice
2. Provides a code example of using the practice
3. Discusses potential negative/positive implications of using the practice
4. Describes alternative approaches to the same problem
5. Provides like for like code samples comparing the practice to alternative approaches
6. Discusses pros/cons of the compared approaches
7. Offers a conclusion on when/where/if the practice is suitable

Using this metric, a manual page that describes a practice and provides a sample of how to use it would score two whereas an article that discussed the pros/cons of different approaches and made a recommendation would score seven.

## 2. Meta-analysis

For any meaningful conclusions to be drawn, two axes are required. Clinical trials could be separated by their Jadad score but this alone tells us nothing about the efficacy of the treatment. To produce a conclusion we need to plot the Jadad Score of a trial against outcome.

For example, a set of trials studying the same treatment can be analysed and observations drawn such as *trials with lower Jadad scores are more likely to produce a positive result*, indicating that the stronger the methodological rigor the less likely the treatment is to be shown to be effective.

Programming articles do not produce a result, but they can offer a recommendation to use or avoid the practice being discussed. A manual page won't make a recommendation but an opinion piece will discuss if/when the practice being described should be used.

A five point scale was used to model the recommendation made by an article:

1. Always favour this practice over alternatives
2. Favour this practice over alternatives unless specific (defined\*) circumstances apply
3. Neutral - No recommendation (e.g. a manual page) or no conclusion drawn
4. Only use this practice in specific (defined\*) circumstances
5. Always favour alternative approaches

A five point scale was chosen over a three point scale as there may be cases where an article is concluded with a discussion of trade-offs. For example where an approach may be faster but less flexible an author may conclude their article with something like "use this practice unless performance is a priority".

This meta-analysis will focus on flexibility. If a conclusion is drawn that you

should use a practice when flexibility is preferred over performance (or any other consideration) then the article would be awarded a score of 2 and considered as "Favour this practice unless performance is a paramount concern".

The focus of the analysis could be changed to performance, security or any other metric and results gathered in the same manner.

\* For scores 2 and 4, the specific circumstances have to be described rather than alluded to.

For example Buss[7] writes:

*When designing a system, it's important to pick the right design principle for your model. In many circumstances, it makes sense to prefer composition over inheritance.*

This article only alludes to when using inheritance is preferable and provides only examples where composition is preferred. In this case the article is given a 5 despite the conclusion saying "many circumstances" rather than "all circumstances".

On the other hand, Ericson[8] says:

*If you aren't sure if a class should inherit from another class ask yourself if you can substitute the child class type for the parent class type. For example, if you have a Book class and it has a subclass of ComicBook does that make sense? Is a comic book a kind of book? Yes, a comic book is a kind of book so inheritance makes sense. If it doesn't make sense use association or the has-a relationship instead.*

In this instance, the author clearly states a situation where inheritance should be used over composition so would be given a recommendation score of 4.

### 3. Collecting data

#### Non-academic sources

*In 1950, a vote at the meeting of the British Association for the Advancement of Science showed that about half those present now embraced the idea of continental drift. [...] Interestingly, oil company geologists had known for years that if you wanted to find oil you had to allow for precisely the sort of surface movements that were implied by plate tectonics. But oil geologists didn't write academic papers; they just found oil.*

Bryson[9]

The Singleton has been regarded as bad practice in industry since at least 2003[10] with developers denouncing it ever since[11-21] yet where it is mentioned in academia it is only discussed as having been utilised while developing software rather than discussing whether it should or should not have been used[23-23]. Given the scale of negativity towards the pattern in industry and lack of discussion in academia, it's unsurprising that patterns lesser known within industry are never even mentioned in academic works.

This is likely because practices are encountered by people who spend 8 hours a day working on large projects where they are likely to encounter problems that academics focussing mostly on theory will not. Industry experts tend to work on large software projects which require constant maintenance and enhancement for years or even decades. They are able to determine which practices prevent them performing maintenance efficiently.

Although, like the oil geologists, they don't tend to write academic papers, many developers post articles on websites run by themselves or the company they work for discussing these issues.

As there is little discussion of the merits of most practices in academia yet there are many articles written by companies, developers and technology journalists, a wider search was conducted using Google. As a Google search for *singleton pattern* yields over half a million hits, a complete systematic review was not feasible. Instead, the first 100 relevant results from a Google search for the relevant practice will be used as the sample.

A *relevant result* is defined as an article which is written by a single author or organisation describing or discussing the singleton pattern. Discussion forums, posts on social media and question & answer sites will not be included as these pages will include multiple opinions. Comments sections on articles will be omitted for the same reason. Any article which has a *Jadad* style score of zero will also be deemed irrelevant.

Google was used to act as a randomization tool. A search returns any articles discussing the practice regardless of whether they are for or against its use.

Each article was then given a *Jadad* style score from 0-7 and a score for its recommendation.

### 3.1 Additional considerations

There are several practical issues with collecting data in this manner:

1. To minimise the effect of Google giving user-specific results based on previous searches, results were collected while logged out and using the browser's private browsing mode and closing the browser between each search term.
2. Search results will not be truly random due to the way Google's algorithm works and results will be sorted by *\*relevance\** and the way Google sorts the results may have implicit bias: The most popular links and most cited links will appear first. Although not truly random, this gives a better overview of the zeitgeist than a genuinely randomised sample by putting the most read/cited articles ahead of less read/cited pages. Articles which are widely shared and linked to will be more likely to appear in the first 100 results.
3. A practice may have more than one common name. When this is the case, each name will be searched for and 100 results collected in total. If a practice is known by 4 different names, the first 25 relevant results for each practice were used. If a result lists both names it will only be counted once.
4. Other search engines may yield different results. Google was chosen because of its dominance and likelihood to have indexed more results. Using a search engine such as Qwant[24] which does not offer personalised results would make the results easier to replicate but may not offer as comprehensive results. Regardless of which search engine is used, results will change over time.

Further research is required to determine the extent of which these factors may affect

results.

However, regardless of these factors, results should be indicative of developers attitudes towards the programming practice being analysed.

#### **4. Test methodology**

To verify that the suggested meta-analysis methodology produces meaningful results, a meta-analysis was performed on two practices where the result can be anticipated with a high degree of certainty. If the methodology works as intended, the following hypotheses should be proven true.

##### **4.1.0 Singleton pattern**

The singleton pattern is well known as being considered bad practice among developers[19] and will act as a good benchmark for testing the meta-analysis methodology.

##### **4.1.1 Hypothesis**

Before the results were collected it was expected that articles which had a higher *Jadad style score* (higher academic rigour) would be more likely to suggest avoiding the practice.

##### **4.2.0 Dependency Injection**

Dependency Injection is antithesis to the Singleton Pattern and is much more flexible. Although there are some practical considerations when using Dependency Injection and there is widespread discussion about the best way to implement it, it's widely considered the best approach for flexibility[25].

##### **4.2.1 Hypothesis**

Dependency Injection a well established method of increasing flexibility in code[26]. Because of this, it was expected that there would be few to no negative recommendations and as the *Jadad style score* increases articles should be more likely to suggest favouring dependency injection over alternative approaches.

## 5. Results

### 5.1 Singleton

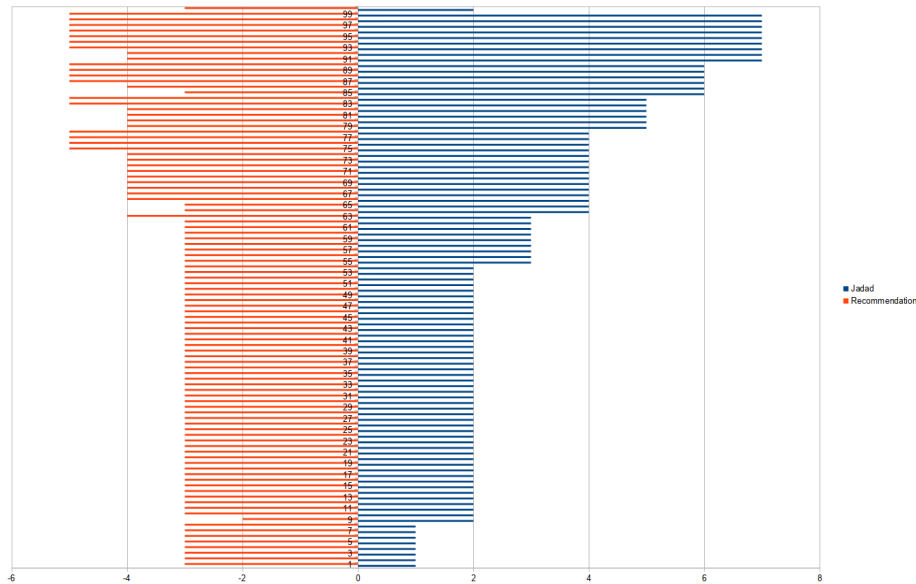


Fig 1. Singleton - Results

Each line represents an article and the left (orange) bar for each article is the recommendation going from 5: Avoid this practice at all costs (Far left) to 1: Favour this practice over alternatives.

The right (blue) bar for each article is the Jadad style score measuring academic rigour. A score of seven means the article describes the practice, provide code examples, discusses alternative approaches, provides like-for-like code samples, discusses the pros/cons of each approach and makes a recommendation of which approach should be used.

Article 1 has a recommendation score of 3 and a Jadad style score of 1. It does not go into detail and its recommendation is neutral; it doesn't suggest either avoiding or favouring use of the Singleton Pattern.

Article 99 on the other hand has a strongly recommends against using the Singleton Pattern and has an Jadad style score of 7, it compares the singleton against alternatives in detail and concludes by strongly recommending against its use (recommendation score of 5).

Raw data is available as appendix 1.

As hypothesised, articles with a high academic rigour are considerably more like to suggest avoiding the singleton pattern.

**Table 1.** Singleton Pattern recommendation score

Recommendation	Number of articles making recommendation
1: Always favour this practice over alternatives	0
2: Favour this practice over alternatives except in specific circumstances	1
3: Neutral/no recommendation	65
4: Favour alternative approaches except in specific circumstances	16
5: Always favour alternative approaches	18

If a simple tally was used, the singleton pattern would appear to have a mostly neutral recommendation score. 65% of articles do not recommend for or against its use.

### 5.1.1 Key findings - Singleton Pattern

- The mode recommendation is neutral. If a developer looked through articles about the singleton pattern, 65% of the articles they read would not recommend against using the Singleton Pattern.
- The mean recommendation score is 3.5. From this alone it could be inferred that the singleton pattern is generally considered to be neutral, slightly discouraged but not widely avoided.
- When the Jadad style score is taken into account, every article which makes a recommendation recommends against using the singleton pattern (recommendation score of 4 or 5).
- Only 22% of articles about the singleton pattern even mention alternative approaches that can be used to solve the same problem
- Of those that recommend against using the pattern, over half say it should be avoided at all cost.
- 55 of the 65 articles which make a neutral recommendation are manual type pages (Jadad style score of 2) which show how to use the pattern but do not weigh in on when, where or if it should be used and do not compare the pattern to alternatives.
- No articles which make a recommendation recommend using the singleton pattern instead of alternative approaches



## 5.2 Dependency Injection

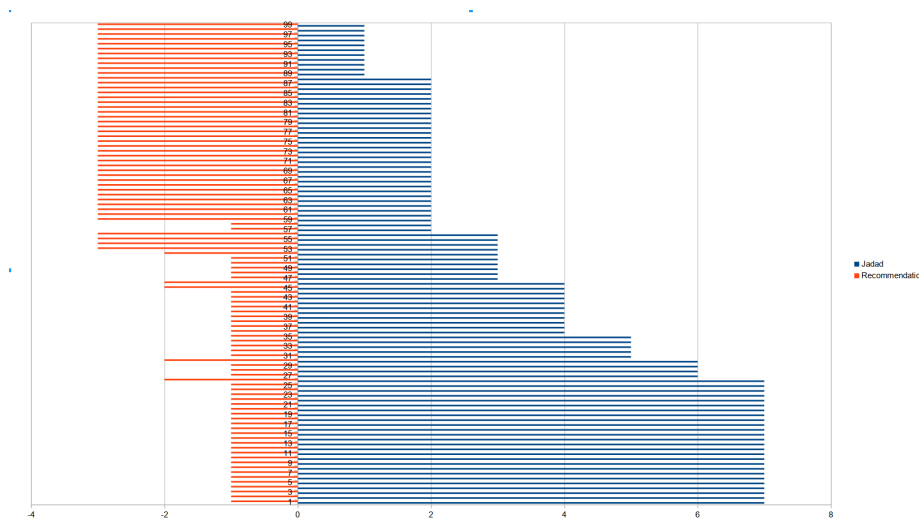


Fig 1. Dependency Injection - Results

Raw data is available in appendix 2.

As hypothesised, Dependency Injection is seen as overwhelmingly positive with zero articles discouraging its general use.

The breakdown of recommendation scores is as follows:

**Table 2.** Dependency Injection recommendation score

Recommendation	Number of articles making recommendation
1: Always favour this practice over alternatives	50
2: Favour this practice over alternatives except in specific circumstances	5
3: Neutral/no recommendation	45
4: Favour alternative approaches except in specific circumstances	0
5: Always favour alternative approaches	0

In a tally of whether articles recommend using or avoiding the practice, 50% of articles recommending using the practice over alternatives.

### 5.2.1 Key findings - Dependency Injection

- The mean score is 1.94 which shows that even using a simple tally, the overall recommendation is that Dependency Injection is a favourable pattern among developers.
- Every article with an academic rigour score of 4 or higher recommends using this practice instead of alternative.
- When the Jadad style score is taken into consideration, 47 of the 50 articles with a neutral recommendation are manual style pages which show how the pattern is used but do not discuss when, where or if it should be used.

- Discounting the manual pages, only two of the remaining 53 articles make a neutral recommendation and both of those have a \*Jadad\* style score 3.
- As the Jadad style score increases, the probability that an article will recommend using Dependency Injection over alternatives increases
- Only 5 of the 55 articles in favour of dependency injection suggest there some specific circumstances where alternatives should be used instead (\*Jadad\* style score of two).

## 5. Conclusion

By testing the methodology with practices that the outcome can be predicted for it was possible to validate this meta-analysis methodology.

The methodology produced the expected result. It was shown that if an author considered alternative approaches they were more likely to recommend against using the Singleton Pattern. The inverse was also true for Dependency Injection.

As these were the expected results, the methodology suggested can be shown to work as intended and provide an overview of the attitudes of developers about any given practice.

This meta-analysis methodology gives more insight into the overall opinion of programming practices than a simple tally of for/against/neutral by also accounting for academic rigour.

### 5.2 Additional findings

1. Although a small sample size of two practices were tested, in both cases roughly half of articles analysed do not make a recommendation on when/where the practice should be used. For the singleton pattern only 45% of analysed articles discussed whether the pattern should be used or avoided.
2. Any developer looking for information on a practice will find more information about \*how\* to use a practice than \*when\* or \*where\* the practice is applicable.

#### ## 5.1 Problems Encountered

Data collection using Google became increasingly difficult after around 80 relevant results. The number of irrelevant articles appearing in search results begin to heavily outweigh the relevant articles and there was a significant issue with duplicated content. Articles had been posted on multiple websites, often without dates or author names, making it difficult to keep track of which articles had already been included in the meta-analysis.

Since Dependency Injection and the Singleton pattern are both widely known and discussed programming practices, finding 100 unique relevant results for lesser known practices may be difficult.

### 5.2 Future Research

This research could be continued by running the same meta-analysis on different

search engines and comparing the results or looking into trends over time using article dates. For example, it may be observed that a practice is seen favourably in articles published in 1990s-2000s and then less favourably as time progresses.

This methodology could be abstracted to and used for a meta-analysis of any widely discussed topic by defining the scales for academic rigour and recommendation.

1. Hevery, M. (2008) *Flaw: Constructor Does Real Work* [online]. Available from: <http://misko.hevery.com/code-reviewers-guide/flaw-constructor-does-real-work/>
2. Cochrane, C. (n.d.) *Cochrane* [online]. Available from: <http://www.cochrane.org/>
3. Mathie, R., Frye, J., Fisher, P. (2015) Homeopathic Oscillococcinum® for preventing and treating influenza and influenza-like illness. *Cochrane Database System Rev* 12 .
4. Goldacre, B. (2010) *Bad Science* ISBN: 978-0-00-724019-7. Fourth Estate.
5. Jadad, A., Moore, A., Carroll, D., Jenkinson, C. (1996) Assessing the quality of reports of randomized clinical trials: Is blinding necessary?. *Controlled Clinical Trials* 17(1), pp.1-12. ELSEVIER.
6. Olivo, S., Macedo, L., Caroline, I., Fuentes, J., Magee, D. (2008) Scales to assess the quality of randomized controlled trials: a systematic review.(Research Report). *Physical Therapy* 88(2), pp.156.
7. Buss, M. (2016) *Interfaces vs Inheritance in Swift* [online]. Available from: <https://mikebuss.com/2016/01/10/interfaces-vs-inheritance/>
8. Ericson, B. (1995) *Association vs Inheritance* [online]. Available from: <http://ice-web.cc.gatech.edu/ce21/1/static/JavaReview-RU/OOBasics/ooAssocVsInherit.html>
9. Bryson, B. (2010) *A short history of nearly everything* ISBN: 9780552997041. London : Black Swan.
10. Radford, M. (2003) Singleton - the anti-pattern. *Overload* 57. ACCU.
11. Hevery, M. (2008) *Singletons are Pathological Liars* [online]. Available from: <http://misko.hevery.com/2008/08/17/singletons-are-pathological-liars/>
12. Sayfan, M. (n.d.) *Avoid Global Variables, Environment Variables, and Singletons* [online]. Available from: <https://sites.google.com/site/michaelsafyan/software-engineering/avoid-global-variables-environment-variables-and-singletons>
13. Densmore, S. (2004) *Why Singletons Are Evil* [online]. Available from: <http://blogs.msdn.com/b/scottdensmore/archive/2004/05/25/140827.aspx>
14. Yegge, S. (2004) *Singleton Considered Stupid* [online]. Available from: <https://sites.google.com/site/steveyegge2/singleton-considered-stupid>
15. Ronacher, A. (2009) *Singletons and their problems in Python* [online]. Available from: <http://lucumr.pocoo.org/2009/7/24/singletons-and-their-problems-in-python/>
16. Brown, W. (2013) *Why Singletons are "Bad Patterns"* [online]. Available from: <http://brollace.blogspot.co.uk/2013/04/why-singletons-are-bad-patterns.html>
17. Kofler, P. (2012) *Why Singletons Are Evil* [online]. Available from: <http://blog.code-cop.org/2012/01/why-singletons-are-evil.html>
18. Weaver, R. (2010) *Static methods vs singletons: choose neither* [online]. Available from: <http://www.phparch.com/2010/03/static-methods-vs-singletons-choose-neither/>
19. Knack-Nielsen, T. (2008) *What's so bad about the Singleton?* [online]. Available from: <http://www.sitepoint.com/whats-so-bad-about-the-singleton/>
20. Badu, K. (2008) *What's so evil about Singleton?* [online]. Available from: <http://www.sitepoint.com/forums/showthread.php?530917-What-s-so-evil-about-Singleton>
21. Hart, S. (2011) *Why helper, singletons and utility classes are mostly bad* [online]. Available

## Appendix VIII. Questionnaire questions

Below are the questions asked in the questionnaire used for evaluation purposes in chapter 6:

Question	Type	Possible answers
How would you describe yourself as a programmer?	Single choice	Novice Hobbyist Student Open Source Developer Academic Professional: Junior Developer Professional: Senior Developer
Which languages do write Object-Oriented code in regularly? Please tick all that apply.	Multiple choice	PHP Java Python Ruby Go Javascript C++ Rust Other
Do you use code reviews as part of your workflow?	Single choice	Yes - As a reviewer Yes - My code is reviewed by others Yes - I review others work and my own code is reviewed by others No
Do you use code review tools such as scrutimizer, phpmd, pmd, etc?	Single choice	Always Often Sometimes Rarely Never
During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply	Multiple choice	Performance Issues Bugs User Experience Security Issues Correctly and consistently following coding conventions (e.g. names, brace position) Code flexibility

Question	Type	Possible answers
Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Multiple choice	Encapsulation Tell, Don't ask Law of Demeter (digging into collaborators) Separation of concerns Dependency Injection Loose coupling Favour composition over inheritance Immutability Single Responsibility Principle
Do you try to follow Object-Oriented best-practices when developing software?	Single choice	Always Often Sometimes Rarely Never
Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Single choice	Always Often Sometimes Rarely Never
Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Multiple choice	Global variables Static variables (including private static variables) Public static methods Inheritance Composition Service Locator Dependency Injection God object Singleton Pattern Constructor Injection Annotations for configuration Tight coupling Setter Injection Facade Marker Interface Mutable Objects Visitor Pattern
Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	Text	N/A

Question	Type	Possible answers
The insphpect site is intuitive and easy to use	Single choice	Strongly Agree Agree Don't Know Disagree Strongly Disagree
How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Single choice	Strongly Agree Agree Don't Know Disagree Strongly Disagree
Do you agree with the recommendations made by Insphpect?	Single choice	Strongly Agree Agree Don't Know Disagree Strongly Disagree
How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Single choice	Strongly Agree Agree Don't Know Disagree Strongly Disagree
How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Single choice	Strongly Agree Agree Don't Know Disagree Strongly Disagree
How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Single choice	Strongly Agree Agree Don't Know Disagree Strongly Disagree
Is there anything you think is missing from Insphpect which should be included in a future update?	Text	N/A
Do you have any general comments about Insphpect?	Text	N/A

## **Appendix IX. Industry published article about the tool**

This article was originally published on popular industry website sitepoint.com. It outlines the tool and the uses it has for developers. Some of the sections were originally written for this thesis.

# How to Ensure Flexible, Reusable PHP Code with Insphect

Insphect is a tool I wrote as part of my PhD project. It scans code for Object-Oriented Programming techniques that hinder code reusability and flexibility.

## Why?

Let me begin with two mundane observations:

1. Business requirements change over time.
2. Programmers are not clairvoyant.

New product launches, emergency lockdown regulations, expanding into new markets, economic factors, updated data protection laws - There are a lot potential causes for business software to need updating.

From those two observations we can infer that programmers know that the code they write is going to change but what those changes will be or when they will happen.

Writing code in such a way that it can be easily adapted is a skill that takes years to master.

You're probably already familiar with programming practices that come back and haunt you. Novice programmers quickly realize that global variables are more trouble than they're worth and the once incredibly popular [Singleton Pattern has been a dirty word for the last decade](<https://www.sitepoint.com/whats-so-bad-about-the-singleton/>).

How you code your application has a big impact on how easy it is to adapt to meet new requirements. As you progress through your career you learn techniques that make adapting code easier. Once the grasp fundamentals of Object-Oriented Programming you wonder how you ever did without it!

If you ask 10 developers to produce software, given the same requirements you'll get 10 different solutions. Some of those solutions will inevitably be better than others.

Consider a ship in a bottle and a model ship made of Lego. Both are model ships but changing the sails on the ship in a bottle is very difficult and reusing the parts, near impossible. However, with a lego ship, you can easily swap out the sails or use the same components to build a model rocket, house or a car.

Certain programming techniques lead to the *ship-in-a-bottle* approach and make your code difficult to change and adapt.

## Insphect

[Insphect](<https://insphect.com>) is a tool which scans your code for programming practices that lead to this kind of a ship in a bottle design.

It grades your code based on how flexible it is and highlights areas where flexibility can be improved.



## What does Insphpect look for?

Currently, Insphpect looks for the following:

- [Tight coupling](<https://insphpect.com/traits/tight-coupling>)
- Hardcoded configuration
- Singletons
- Setter Injection
- Using the new keyword in a constructor
- Service locators
- Inheritance
- Static methods
- Global state
- Files that have more than one role (e.g. defining a class and running some code)

If it detects anything it identifies as inflexible it highlights the code, explains why it highlighted the issue then grades your whole project and individual classes on a score of 0-100 (with 100 being no issues detected). As a proof-of-concept, for some detections it is able to automatically generate a patch file that re-writes the code to remove the inflexibility entirely.

[Take a look a sample report here](<https://insphpect.com/report/5e63a943a8da7>)

Insphpect is currently in the testing phase and it would really help my research progress if you can check it out and complete the survey in the "Give your feedback" section of the site.

## Background

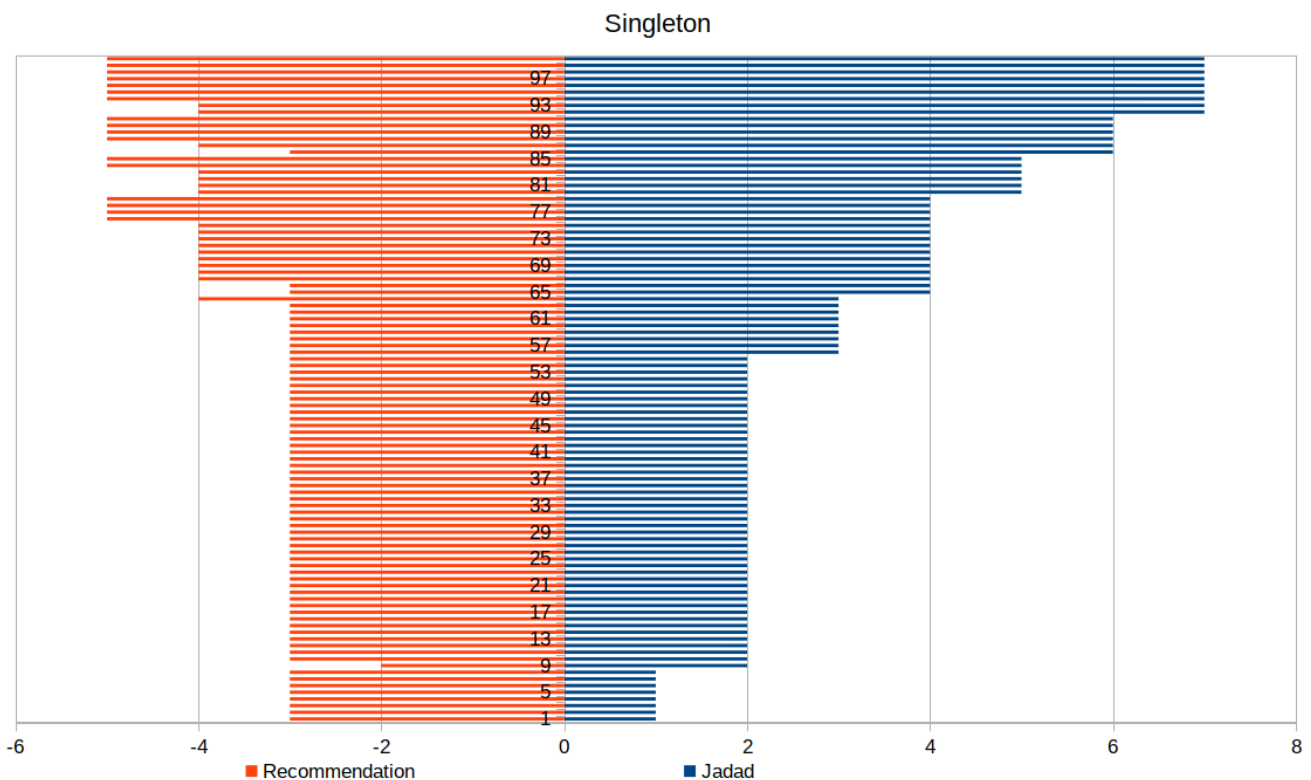
Are those bad practices really bad though?

This was one of the more difficult parts of the background research and you can read about how this was done in detail on the [Insphpect Website](<https://insphpect.com/background>).

However, this can be summarized as:

- The opinions of each bad practice were collected from 100 authors per practice.
- The author's opinion on the practice was graded on a scale of 1-5.
- The authors methodological rigor was graded on a scale of 1-7 based on the Jadad score used for clinical trials,

These were then plotted like the graph below:



Each horizontal line represents an article and the left (orange) bar for each article is the recommendation going from 5: Avoid this practice at all costs (Far left) to 1: Favor this practice over alternatives.

The right (blue) bar for each article is the Jadad style score measuring analytic rigor. A score of seven means the article describes the practice, provide code examples, discusses alternative approaches, provides like-for-like code samples, discusses the pros/cons of each approach and makes a recommendation of which approach should be used.

In the case of the Singleton above, authors who compare the singleton to alternative approaches, discuss the pros/cons, etc are significantly more likely to suggest using alternative approaches.

## Walkthrough

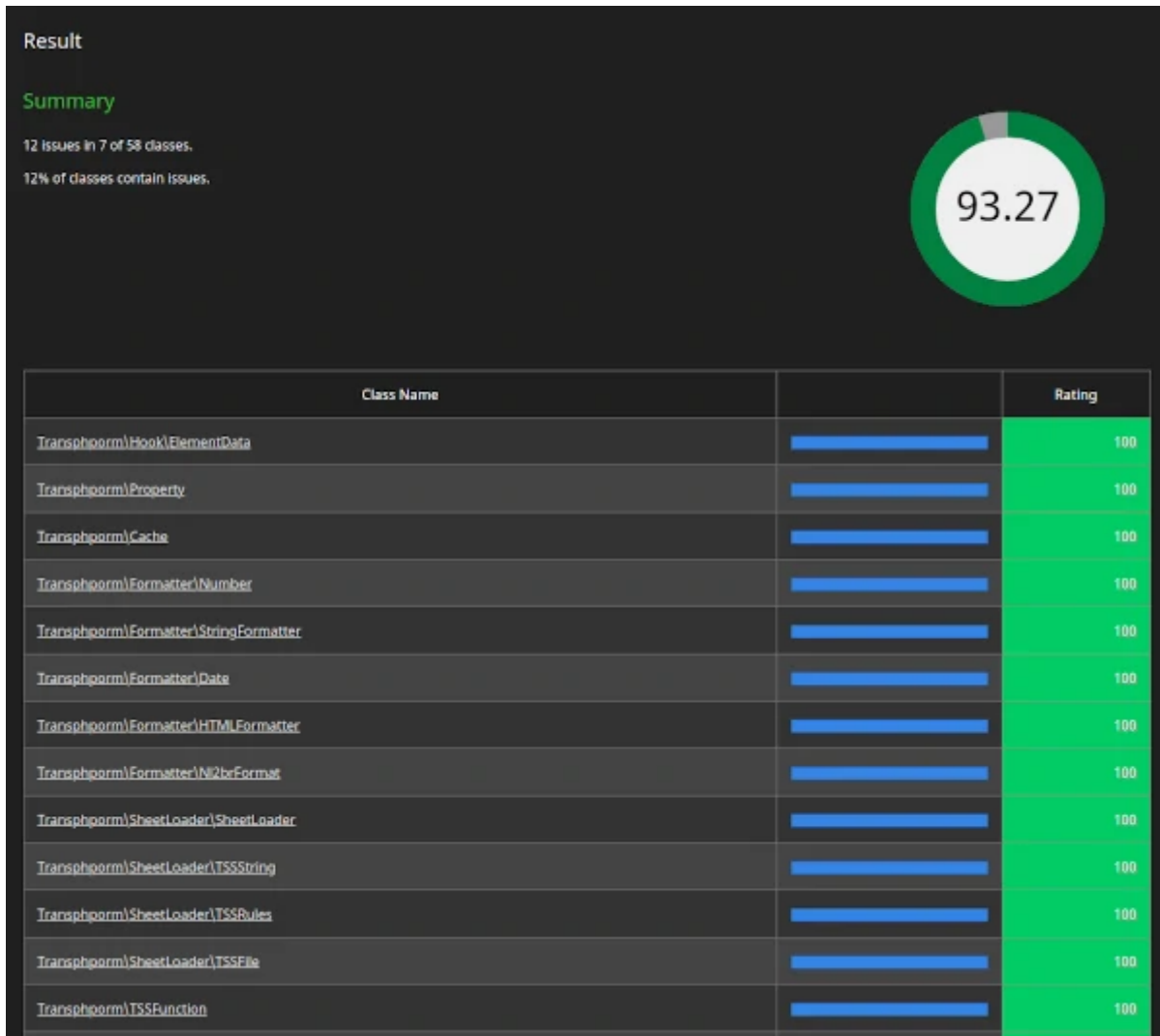
Currently Insphpect allows uploading code via a git repository URL or a zip file.

So not to point out flaws in other people's work, let's take a look at one of my own projects to see what it identifies.

We'll use [<https://github.com/Level-2/Transhporm>](<https://github.com/Level-2/Transhporm>) as an example project.

This is quite a good example because it has a very high score on another code-quality tool [Scrutinizer](<https://scrutinizer-ci.com/g/Level-2/Transhporm/>).

Firstly, enter the git URL `https://github.com/Level-2/Transhporm` into the text box and at the top of the home page and press "Go". It will take a few seconds to minutes, depending on the size of the project and generate a report which looks something like this:



Once you're on the report page you'll see a summary at the top with an overall grade out of 100 with 100 being very good and 0 being very poor.

Underneath the summary, you'll see a list of all the classes in the project, each with its own grade.

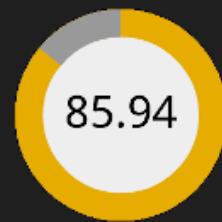
Don't worry if your code doesn't get a perfect score, it's unlikely that it will. Remember, Insphect is a tool that identifies flexibility in your code. There are like parts of your code (like the entry point) where flexibility isn't warranted.

For Transphorm, it has highlighted issues in 7 classes.

Let's take a look at some of those. Scroll down to Transphorm\Parser\CssToXPath and click the link. You'll see a score for that particular class and a list of issues which have been identified.

In this case, it has identified a static variable and a static method. Clicking on one of the red lines will reveal an explanation of why the line was flagged up.

For example, clicking line 12 will give an explanation of why static variables are less flexible than instance variables.



### Detected issues

Issue	Method	Line number
Global/Static variables	NA	12
Use of static methods	processAttr	40

### Code

Click highlighted lines for details

```

1<?php
2/* @description    Transformation Style Sheets - Revolutionising PHP templating    *
3 * @author        Tom Butler tom@r.je    *
4 * @copyright     2017 Tom Butler <tom@r.je> | https://r.je/    *
5 * @license       http://www.opensource.org/licenses/bsd-license.php    BSD License    *
6 * @version       1.2    */
7namespace Transphorm\Parser;
8class CssToXpath {
9    private $specialChars = [Tokenizer::WHITESPACE, Tokenizer::DOT, Tokenizer::GREATER_THAN,
10        '\w+', Tokenizer::NUM_SIGN, Tokenizer::OPEN_SQUARE_BRACKET, Tokenizer::MULTIPLY];
11    private $translators = [];
12    - private static $instances = [];

```

Why this impedes flexibility

Global variables

*Note: A future update will differentiate between `private static` variables and `public static` or `global` variables as `private static` variables do not cause as much of a problem.*

**Summary**

• Hidden dependencies

Although there is a more in-depth explanation of the issues caused by static properties on the [report](<https://insphpect.com/report/class/5e63a8f479b6d/Transphorm%5CParser%5CCssToXpath>) as a quick refresher, static variables have one value which is shared across all the instances of the class.

This is inherently less flexible than an instance variable because using an instance variable allows each instance to have a different value.

For example, consider the following:

```

class User {
    public static $db;
    public $id;
    public $name;
    public $email;
    public function save() {
        $stmt = self::$db->prepare('REPLACE INTO user (id, name,
email) VALUES (:id, :name, :email)');
$stmt->execute([

```

```

        'id' => $this->id,
        'name' => $this->name,
        'email' => $this->email
    });
}
}

```

Because `$db` is static, every instance of this class shares the same `$db` instance and records will always be inserted into the same database.

While this sounds reasonable, let me give you a real-world example.

One of our clients was a recruitment agency. About 2 years after we developed their site, they took over another smaller company. They wanted to retain the second company's website and branding because it was quite well known in the niche they were in.

Our client asked us the following:

"On the second company's site, can you add a checkbox when adding a job that also adds the job to our database so people viewing our site can also see the job and visa versa"

A fairly simple request. Run an insert query into two different database.

But, because the website used a static global database instance this was needlessly difficult!

The developers of that site wrote the code confident that only one database connection would ever be needed. They were wrong.

Remember, you are not clairvoyant and it is impossible to anticipate what flexibility may be needed in the future

As suggested by Insphpect, the solution to this is using instance variables:

```

class User {
    private $db;
    public $id;
    public $name;
    public $email;
public function __construct(\PDO $db) {
    $this->db = $db;
}
public function save() {
    $stmt = self::$db->prepare('REPLACE INTO user (id, name,
email) VALUES (:id, :name, :email)');
    $stmt->execute([
        'id' => $this->id,
        'name' => $this->name,
        'email' => $this->email
    ]);
}
}

```

```
}
```

Now a User instance can be used with different database instances:


```
new User($database1);  
new User($database2);
```

For Transphorm\Parser\CssToXpath we could do the same, remove the static variable and consider making it an instance variable rather than a static variable.

## Using new in constructor

Let's take a look at one of the other classes: Transphorm\Builder

Transphorm\Builder



### Detected issues

Issue	Method	Line number
Using new in constructor	__construct	28
Using new in constructor	__construct	29
Using new in constructor	__construct	31

### Code

Click highlighted lines for details

```
1 <?php  
2 /* @description Transformation Style Sheets - Revolutionising PHP templating *  
3 * @author Tom Butler tom@r.je *  
4 * @copyright 2017 Tom Butler <tom@r.je> | https://r.je/ *  
5 * @license http://www.opensource.org/licenses/bsd-license.php BSD License *  
6 * @version 1.2 */  
7 namespace Transphorm;  
8 /** Builds a Transphorm instance from the 3 constituent parts. XML template string, TSS string and data */  
9 class Builder {  
10     private $template;  
11     private $tss;  
12     private $cache;  
13     private $tline;  
14     private $modules = [];  
15     private $config;  
16     private $filePath;  
17     private $cacheKey;  
18     private $defaultModules = [  
19         '\\Transphorm\\Module\\Basics',  
20         '\\Transphorm\\Module\\Pseudo',  
21         '\\Transphorm\\Module\\Format',  
22         '\\Transphorm\\Module\\Functions'  
23     ];  
24  
25     public function __construct($template, $tss = '', $modules = null) {  
26         $this->template = $template;  
27         $this->tss = $tss;  
28 +         $this->cache = new Cache(new \ArrayObject());  
29 +         $this->filePath = new FilePath();  
30         $modules = is_array($modules) ? $modules : $this->defaultModules;  
31 +         foreach ($modules as $module) $this->loadModule(new $module);  
32     }  
33  
34     //Allow setting the time used by Transphorm for caching. This is for testing purposes  
35     //Would be better if PHP allowed setting the script clock, but this is the simplest way of overriding it  
36     public function setTime($time) {
```

This has a score of zero, that's rather poor. Examining the report in detail, Insphpect has picked up the same issue 3 times: Using the new keyword in a constructor.

[Google Programming Coach Misko Hevery does a great job at explaining why this is a poor programming practice](<http://misko.hevery.com/code-reviewers-guide/flaw-constructor-does-real-work/>) but here is a simple example from Insphpect's output:

```
class Car {
    private $engine;
    public function __construct() {
        $this->engine = new PetrolEngine();
    }
}
```

Here whenever an instance of `Car` is created, an instance of `PetrolEngine` is created. That makes it very inflexible because there is no way to construct a `Car` with a different engine type. Every car modeled in this system must have a `PetrolEngine`

Instead, if Dependency Injection were used:

```
class Car {
    private $engine;
    public function __construct($engine) {
        $this->engine = $engine;
    }
}
```

Different cars can be created with an instance of `PetrolEngine`, `DieselEngine`, `ElectricEngine`, `JetEngine` or any other engine type that exists in the project.

To fix this error in the `Transphorm\Builder`, all of the variables which currently have hard-coded class names should use constructor arguments instead.

There are other issues identified by Insphpect but you can try it out for yourself and see how your project fares.

## Behind the scenes

You might be wondering how the scores are calculated and why this class got a zero. At the present time, the weightings are subject to change once more projects have been scanned and more feedback has been provided.

The scores are designed to be indicative for comparing one project/class to another.

The overall project score is just an average of all the classes in the project. This was implemented because a project with 2 issues in 1000 classes is a lot better overall than a project with 2 issues in 2 classes.

Each bad practice is weighted based on whether it impedes flexibility for the entire class or only impedes flexibility for a method.

## **Conclusion**

Insphect can be used to identify areas of your code which make future changes more difficult than they could be and it offers suggestions on how to write the code in a more flexible manner.

Remember, you're not clairvoyant and have no way to know how your code is going to need to change!

Insphect is currently a work in progress and the more people who use it (and complete the survey) the better it will become.

How did your project or favorite library score? Be sure to complete the survey as it and provide valuable data for my PhD project and help the tool improve!



## Appendix X. Questionnaire results raw data

This appendix contains the raw data gathered from the questionnaire used for evaluation in chapter 6.

### Response #1

Question	Answer
1. How would you describe yourself as a programmer?	Open Source Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	Yes - I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Dependency Injection, God object, Singleton Pattern, Constructor Injection, Annotations for configuration, Setter Injection, Facade, Marker Interface, Mutable Objects, Visitor Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #2

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Disagree
13. Do you agree with the recommendations made by Insphpect?	Disagree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Disagree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Disagree
13. Do you agree with the recommendations made by Insphpect?	Disagree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Disagree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree

17. Is there anything you think is missing from Insphpect which should be included in a future update?	<p>As a tool designed specifically for inspecting PHP code, I would recommend disregarding Inheritance issues when the class in question extends one of the built-in Exception classes, as there is no other way in the language to throw custom exceptions.</p> <p>And for a more opinionated suggestion, you may want to add an option to disregard Inheritance issues when the class in question extends an abstract class. (An option for advanced users, as opposed to by default, because it should still be discouraged in general.)</p> <p>Those two changes alone should take care of the overwhelming majority of false positives in the inspection.</p>
--	--

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Disagree
13. Do you agree with the recommendations made by Insphpect?	Disagree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Disagree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree

17. Is there anything you think is missing from Insphpect which should be included in a future update?	<p>As a tool designed specifically for inspecting PHP code, I would recommend disregarding Inheritance issues when the class in question extends one of the built-in Exception classes, as there is no other way in the language to throw custom exceptions.</p> <p>And for a more opinionated suggestion, you may want to add an option to disregard Inheritance issues when the class in question extends an abstract class. (An option for advanced users, as opposed to by default, because it should still be discouraged in general.)</p> <p>Those two changes alone should take care of the overwhelming majority of false positives in the inspection I ran.</p> <p style="text-align: right;">Thomas Butler</p>
--	--

### Response #3

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes - I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection, Facade
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

### Response #4

Question	Answer
1. How would you describe yourself as a programmer?	Hobbyist
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	Python, Javascript, Other
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scruitinizer, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Immutability
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, God object
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #5

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, C++
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Dependency Injection, Loose coupling
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Inheritance, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Don't Know
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #6

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Python
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Never
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	N/A
18. Do you have any general comments about Insphpect?	

## Response #7



Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Python, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Often
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	I'd like to be able to analyse a private repository and keep the reports behind a login system. It looks like all code uploaded is available on public URIs which prevents me from uploading code I write for work.

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Python, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmd, pmd, etc?	Often
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	I'd like to be able to analyse a private repository and keep the reports behind a login system. It looks like all code uploaded is available on public URIs which prevents me from uploading code I write for work.

18. Do you have any general comments about Insphpect?	On the couple of repositories I looked at there were definitely a few debatable red lines. For example, private static, sure it does introduce global but it's unlikely to cause any real world issues as the scope for damage is limited to the class it's used in and you'd hope that static was used for a good reason.
---	--

## Response #8

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Nothing I can think of, its great
18. Do you have any general comments about Insphpect?	Very easy to use

## Response #9

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	Other
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables), Public static methods, Composition, Service Locator, Dependency Injection, God object, Singleton Pattern, Constructor Injection, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	Found the background research really interesting as like you mentioned best practice can be very much open to interpretation! Was a really insightful read
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Only the option for more languages!
18. Do you have any general comments about Insphpect?	I tried to open the website from my phone but the scale was off, the header banner spanned to big so wasn't very responsive. Also I am not a big fan of the full page menu, I would prefer a smaller pane on the right rather than the whole screen. Really love the colour scheme and the layout and overall looks and performs very well! Well done Tom :)

## Response #10

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, God object, Singleton Pattern, Setter Injection, Facade, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #11

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Javascript, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Never
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables)
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	Easy to use.

## Response #12

Question	Answer
1. How would you describe yourself as a programmer?	Hobbyist
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Inheritance, God object
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Additional detail about how grades are achieved would be interesting, particularly from a learning/education perspective, however, I don't believe this hinders the solution at all.
18. Do you have any general comments about Insphpect?	The concept is really interesting. I'd be really interested to see how this impacts the quality of code in the future if it is something budding coders are introduced to early on in their learning.

### Response #13

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Python, Javascript
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #14



Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Public static methods, God object
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Disagree
13. Do you agree with the recommendations made by Insphpect?	Disagree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Disagree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Currently, it seems to trigger if it locates the keyword "new", it might be a good idea to exclude internal PHP functions that is initiated by it.
18. Do you have any general comments about Insphpect?	It is an interesting project, though I am afraid if it stays in its current format, that is what it will stay as.  Since you are unable to run it locally (or on servers you control) it is not possible to run it on any client work, due to restrictions in the contract about who you can give access of the code to. In other words, as it is, it can really only be utilized for open source projects.

## Response #15

Question	Answer
1. How would you describe yourself as a programmer?	Novice
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Public static methods, Inheritance, Composition, Singleton Pattern, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #16

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	Javascript, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, God object, Singleton Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Don't Know
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Disagree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	I wouldn't consider it as something that is missing, but possibly a cut-down version of the explanation. A summary of the issue without too going in-depth with the explanation
18. Do you have any general comments about Insphpect?	I think it's a great idea and I would love to see a similar tool for other languages that would apply more to what I use

## Response #17

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Don't Know
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	The responsive mode is not good.
18. Do you have any general comments about Insphpect?	Add suggestion and code correction for red highlighted lines

## Response #18

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmnd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, God object, Annotations for configuration, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	The site doesn't work well on mobile

## Response #19

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java
3. Do you use code reviews as part of your workflow?	Yes - My code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Favour composition over inheritance
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Tight coupling, Visitor Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	I feel the background research was very interesting to read. It is always nice to come up with tools that are not available out there. I believe this tool is great and I would like to use it in the future. Good job Tom!
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Everything looks good. The only thing I would change a little bit is the main text on the homepage it looks shifted on the right leaving a gap on the left. Perhaps, you could make it centred or shift it a little bit towards the left?
18. Do you have any general comments about Insphpect?	

## Response #20

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Inheritance, God object, Singleton Pattern, Annotations for configuration, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Can't run locally. It's fine for testing some toy project or open source library, but I can't possibly use it at work like this.

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Inheritance, God object, Singleton Pattern, Annotations for configuration, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Can't run locally. It's fine for testing some toy project or open source library, but I can't possibly use it at work like this.

18. Do you have any general comments about Insphpect?	<p>It's a cool idea, but fairly limited right now.</p> <ul style="list-style-type: none"> <li>- Small number of very broad inspections - marking every use of inheritance or static methods as a bad practice is a bit too much and not super helpful. You need to identify legitimate, justified uses of inheritance/static methods/whatever if you want this tool to be useful.</li> <li>- You also need to expand the explanations accordingly. They're pretty good, but quite one-sided. They should include a section on when inheritance/static methods/whatever can and should be used too.</li> </ul>
---	---



## Response #21

Question	Answer
1. How would you describe yourself as a programmer?	Academic
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Javascript, Other
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	User Experience, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, Singleton Pattern, Facade
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	I think, as a testing tool, that I think is valuable to good code quality - it should be possible to create an adapter that promotes Test-Driven Development and Behaviour-Driven Design. This would really provide for a cool feature to base a Code Review Service and so on and so forth.
18. Do you have any general comments about Insphpect?	I am hoping the tool is easy to integrate with given it has things like GuzzleHttp port and that of Symfony Routing

## Response #22

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Junior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes - My code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Loose coupling, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, God object, Singleton Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #23

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Python, Javascript
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #24

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, Dependency Injection, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection, Marker Interface, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	I'm happy that it picks up Service Locators. I have a constant debate with colleagues about their use and the explanation provided makes the point better than I can.

## Response #25

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables)
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	The tool states X amount of issues within Y amount of classes, it would be interesting to see these issues, or some form of summary about them
18. Do you have any general comments about Insphpect?	

## Response #26

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Junior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes - My code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), God object, Singleton Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #27

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Javascript, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Disagree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	<ul style="list-style-type: none"> <li>- For tooling like this to be valuable, it should be able to be run in an automated fashion from the CLI, for inclusion in build processes.</li> <li>- The detail offered for why to tackle a certain approach is very heavy duty, and could probably do with a short summary inline, plus a "read more" link to an article detailed the</li> <li>- The tool misidentifies `@inheritDoc` as a configuration-based annotation, which it isn't.</li> </ul>
18. Do you have any general comments about Insphpect?	

## Response #28

Question	Answer
1. How would you describe yourself as a programmer?	Hobbyist
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	
7. Do you try to follow Object-Oriented best-practices when developing software?	Never
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Never
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Don't Know
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #29



Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, User Experience, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Separation of concerns, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables), Public static methods, Constructor Injection, Setter Injection
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Disagree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	I think the information given about an error is too much.
18. Do you have any general comments about Insphpect?	I commented that on the previous question. I was afraid of not getting another opportunity to give feedback.

## Response #30

Question	Answer
1. How would you describe yourself as a programmer?	Open Source Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizor, phpmid, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Service Locator, God object, Singleton Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	n/a
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Disagree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Maybe some kind of visual depiction of how tightly the classes are coupled.
18. Do you have any general comments about Insphpect?	<p>PHP requires thrown objects to be a subclass of Exception so perhaps it should not complain about that.</p> <p>----</p> <p>One place I find static methods useful is for implementing pure helper functions. Since PHP does not support autoloading functions, I stuff them into a final class and use the class as sort of a namespace. (Example I recently added to selfoss: <a href="https://insphpect.com/report/class/5eb24ebf7acd8/hel pers%5CMisc">https://insphpect.com/report/class/5eb24ebf7acd8/hel pers%5CMisc</a>)</p> <p>And yes, it introduces tight coupling but some things are just primitive enough or specific enough it is not worth decoupling them. They can always be easily decoupled by injecting the function as callable through the constructor. It might not be as convenient as having a dependency container inject a named class but we are slowly getting there. With static analysis tools like Psalm, we can finally check the type signature of callables for compatibility, replacing the classic Interface subtype checks. (The added flexibility can be both a good and a bad thing but as the strength of the type systems increases, developers will be able to choose as much rigor or flexibility as they want.)</p>

## Response #31

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	Javascript, Other
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, God object, Constructor Injection, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #32

Question	Answer
1. How would you describe yourself as a programmer?	
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, C++, Other
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	User Experience
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Dependency Injection, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	A proactive tool, helpful for every user, independently of their experience in programming. Well done.

### Response #33

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Rarely
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables), Singleton Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #34

Question	Answer
1. How would you describe yourself as a programmer?	Novice
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmnd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, User Experience, Security Issues
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling
7. Do you try to follow Object-Oriented best-practices when developing software?	Rarely
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	n/a
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #35

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Python, Go
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Composition, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #36

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables)
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	I believe it is a nice tool which helps improve best practices
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #37



Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Dependency Injection, Loose coupling, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Public static methods, God object, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Disagree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	It would be nice to customise the rules, so certain practices which the tool may consider bad could be ignored for a project that needs to use that practice for a valid reason. A tool I use called Sonar for C# has a concept of "Ways" which would be similar.  The ability to send in recommendations for bad practices that might not be covered in the tool would be nice also
18. Do you have any general comments about Insphpect?	The website is quite difficult to use on iPhone 8 Safari (Haven't tried other browsers)

## Response #38

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #39

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #40

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Often
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Public static methods, Service Locator, God object, Annotations for configuration, Tight coupling, Setter Injection
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	There are a few (very few) valid uses for the Singleton, but most of the time it is an anti-pattern.  Inheritance is has even more valid uses, and judging when to use it or not use it is probably extremely difficult to judge in a tool like this.
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Perhaps the output could briefly describe why a particular flaw is flagged, and also provide the weighting used for it (I noticed that one static method reduced the score for a class far less than having the class inherit from a parent class).
18. Do you have any general comments about Insphpect?	Would love to have a command line version to add to my tooling, along with phpstan, phpmd, phpunit, phpmd, etc. :-)

## Response #41

Question	Answer
1. How would you describe yourself as a programmer?	Hobbyist
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Javascript
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Separation of concerns, Dependency Injection
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Rarely
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables)
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #42

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	Java, Go, C++, Rust
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Loose coupling, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Annotations for configuration, Setter Injection
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Don't Know
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #43

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Junior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes - My code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Dependency Injection, Loose coupling
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Inheritance, God object, Singleton Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #44

Question	Answer
1. How would you describe yourself as a programmer?	Novice
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	Java, Other
3. Do you use code reviews as part of your workflow?	Yes - My code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, User Experience, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Never
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/a
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #45



Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java
3. Do you use code reviews as part of your workflow?	Yes - My code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Tell, Don't ask, Separation of concerns, Loose coupling
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables), God object, Singleton Pattern, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	
17. Is there anything you think is missing from Insphpect which should be included in a future update?	easier back and forth navigation when navigating between analyzed files
18. Do you have any general comments about Insphpect?	nice work dude!

## Response #46

Question	Answer
1. How would you describe yourself as a programmer?	Hobbyist
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Don't Know
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #47

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, C++
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmnd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Separation of concerns, Dependency Injection
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	It looks fabulous and it is very simple and easy to use. I can quickly see where there are issues in what files and how to improve it

## Response #48

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Ruby, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, God object, Singleton Pattern, Tight coupling, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	I think it should be available in an offline version. I'm reluctant in uploading private code on the web

## Response #49

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, God object, Annotations for configuration, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	` The feedback is very informative but I feel like having a huge piece of text and examples opening when clicking a line is a bit jarring. Maybe these should be short examples and link off to external articles that are more detailed. Also it would be great for CI purposes if this was a Composer package that people could use to maybe automatically add GitHub comments or something along those lines? Just some ideas. Great project!
11. The insphpect site is intuitive and easy to use	Don't Know
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Disagree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Disagree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #50

Question	Answer
1. How would you describe yourself as a programmer?	Open Source Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes - I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection, Marker Interface, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/a
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	site looks nice, not sure the animations give that professional feel though. The tool works well and it picks up on the things I'd normally look for during code reviews.

## Response #51

Question	Answer
1. How would you describe yourself as a programmer?	Hobbyist
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	
7. Do you try to follow Object-Oriented best-practices when developing software?	Never
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Don't Know
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #52

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Often
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree



Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Often
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree

17. Is there anything you think is missing from Insphpect which should be included in a future update?  Thomas Butler	I'd like to see it handle inheritance differently. There are some cases where PHP forces you to use inheritance (e.g. extending InvalidArgumentException) and it's flagged up as bad as extending one of your own classes. The explanation popup should at least mention this or grade the code differently when extending an inbuilt class. Another example is PDO. Sometimes it's useful to add functionality to it and the only way that can currently be achieved is with inheritance, if you wrap it you can't then pass it into code that expects a \PDO instance so any code which does this is going to lose
---	--

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Often
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree

17. Is there anything you think is missing from Insphpect which should be included in a future update?	I'd like to see it handle inheritance differently. There are some cases where PHP forces you to use inheritance (e.g. extending InvalidArgumentException) and it's flagged up as bad as extending one of your own classes. The explanation popup should at least mention this or grade the code differently when extending an inbuilt class. Another example is PDO. Sometimes it's useful to add functionality to it and the only way that can currently be achieved is with inheritance, if you wrap it you can't then pass it into code that expects a \PDO instance so any code which does this is going to lose
--	--

## Response #53

Question	Answer
1. How would you describe yourself as a programmer?	Academic
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Python, C++, Other
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Separation of concerns, Immutability
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Public static methods, Tight coupling, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #54

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Junior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Dependency Injection
7. Do you try to follow Object-Oriented best-practices when developing software?	Rarely
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Annotations for configuration, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Disagree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	<p>A good feature would be to be able to click on a row in "Detected Issue" and scroll down to the issue.</p> <p>Having less information when first clicking on the issue with an option to read-more? e.g. just display the summary, then have another button to expand the data.</p> <p>Don't know if I missed it, but is there a disclaimer of what happens with uploaded files? Could users use this tool with code that has sensitive information?</p>
18. Do you have any general comments about Insphpect?	

## Response #55

Question	Answer
1. How would you describe yourself as a programmer?	Hobbyist
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Dependency Injection, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables), Public static methods, God object, Singleton Pattern, Annotations for configuration, Facade
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Disagree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	I've uploaded a simple/dirty webapp using Fatfree Framework, which is a mature and useful framework for a hobbyist like me. However it uses global variables, god classes, magic methods, singletons etc, so do I in uploaded classes. And this tool showed 100%.
18. Do you have any general comments about Insphpect?	I trust in your work, and learned a lot from articles on r.je. Thank you.

## Response #56

Question	Answer
1. How would you describe yourself as a programmer?	Academic
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Javascript, C++, Other
3. Do you use code reviews as part of your workflow?	Yes - As a reviewer
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Rarely
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Composition, God object, Singleton Pattern, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #57

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Javascript, Other
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpm, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables)
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	A very useful tool. One possible concern with making code as generalised as possible is it could potentially lead to new developers to a project taking a bit longer to get up to speed.  Also, on the front page in the "How is it different from Scrutinizer/phpmd/etc?" section, number is spelt wrongly

## Response #58

Question	Answer
1. How would you describe yourself as a programmer?	Hobbyist
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	No
18. Do you have any general comments about Insphpect?	NO

## Response #59



Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, God object, Singleton Pattern, Tight coupling, Facade
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Flexibility is so important, (Composition over Inheritance) I add it in my best practices list! Thanks a lot for you work, future PhD!
18. Do you have any general comments about Insphpect?	Nice tool!

## Response #60

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Python, Go
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Often
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, Dependency Injection, God object, Singleton Pattern, Constructor Injection, Tight coupling, Setter Injection, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	It would be useful to be able to add some exceptions to the rules. In PHP there are several times when you are forced to use inheritance or mutable objects because of PHP's inbuilt classes. I can submit a PR or elaborate if the code is available somewhere. It seems wrong to deduct points for things that cannot be avoided.
18. Do you have any general comments about Insphpect?	Nice tool, I'll definitely be using this on my projects moving forward.

## Response #61

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Javascript
3. Do you use code reviews as part of your workflow?	Yes - I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), God object, Singleton Pattern
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Don't Know
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #62

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, C++
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection, Marker Interface
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #63

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Javascript, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), God object, Setter Injection, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	Once analysis is shown, in the "Detected issues" table, I expected to get clickable links to the given lines being highlighted in the code. I was forced to scroll to locate the red arrow markers and missed some until I came back up to tally how many red arrows to go and find. This is a minor UI/UX issue but may prove helpful in a future update.
18. Do you have any general comments about Insphpect?	I would be helpful if the tool collapsed code comments of a given length to prevent the code comments cluttering the output of Insphpect. This would allow us to only see the code as-is and spot the guidance being given by Insphpect much more easily.

## Response #64

Question	Answer
1. How would you describe yourself as a programmer?	Student
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Javascript
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	n/a
18. Do you have any general comments about Insphpect?	<p>An interesting and cool tool and proof of concept. The website design is perhaps not really what I would have gone with but it's certainly quirky and the animation give it an interactive and fun feel. The background on the project is really interesting as is the methods used to assess programming best/worst practices. Great name too!</p> <p>Would love to see this tool developed into a plugin for common IDE's and editors and become available for other languages.</p>

## Response #65

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	Yes - I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Annotations for configuration, Tight coupling, Setter Injection, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Strongly Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	

18. Do you have any general comments about Insphpect?	<p>I've been following your work for years and learnt a lot. This is a clever progression from what you've been doing previously. Great job!</p> <p>I use Scrutinizer regularly and find it to be helpful but rather dumb. As you point out, there are thresholds for grades which seem to have been chosen at random. I like Insphpect's approach of identifying antipatterns a lot more as it has a more solid foundation for the grades given. Thomas Butler</p>
---	---



## Response #66

Question	Answer
1. How would you describe yourself as a programmer?	Novice
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java
3. Do you use code reviews as part of your workflow?	Yes - I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	User Experience
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables)
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	Very helpful tool.

## Response #67

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, God object, Annotations for configuration, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #68

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern



Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Ruby, Go, Javascript, C++, Rust, Other
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade, Marker Interface, Visitor Pattern

## Response #69

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP
3. Do you use code reviews as part of your workflow?	No
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Always
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection, Facade
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Don't Know
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	I am still playing with it so I will report it again later
18. Do you have any general comments about Insphpect?	Totally not agree with annotations and red warning about inheritance. Modern programming will always depend on some other library; it is not enough just to implement interface.

## Response #70

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	PHP, Javascript, Other
3. Do you use code reviews as part of your workflow?	Yes - I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Bugs, User Experience, Security Issues
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, God object, Singleton Pattern, Annotations for configuration, Tight coupling
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	N/A
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Disagree
13. Do you agree with the recommendations made by Insphpect?	Disagree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Disagree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Disagree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Don't Know
17. Is there anything you think is missing from Insphpect which should be included in a future update?	No.
18. Do you have any general comments about Insphpect?	You can't judge whole code by a single word. Your tool is unable to deal with named constructors or immutable classes, to name a few.

## Response #71

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Junior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	Python, Go
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Dependency Injection, Favour composition over inheritance
7. Do you try to follow Object-Oriented best-practices when developing software?	Sometimes
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Inheritance
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #72

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Often
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Always
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Always
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Public static methods, Inheritance, Service Locator, Dependency Injection, Singleton Pattern, Annotations for configuration, Setter Injection, Mutable Objects
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	You should make the raw data available so we can see the "good" and "bad" articles
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Strongly Agree
13. Do you agree with the recommendations made by Insphpect?	Strongly Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	I'd like to be able to run it locally rather than trusting a site with my private code
18. Do you have any general comments about Insphpect?	Neat idea. The FAQ and site doesn't contain any legal information or details about how the tool will use my code once I upload it. This means I won't test it with my own code - was very curious how a private repo I'm working in would perform in your tests.



## Response #73

Question	Answer
1. How would you describe yourself as a programmer?	Novice
2. Which languages do you write Object-Oriented code in regularly? Please tick all that apply.	Other
3. Do you use code reviews as part of your workflow?	Yes - I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Never
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, Correctly and consistently following coding conventions (e.g. names, brace position)
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Sometimes
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Static variables (including private static variables)
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibiltiy of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #74

Question	Answer
1. How would you describe yourself as a programmer?	Open Source Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Python, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizier, phpmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Correctly and consistently following coding conventions (e.g. names, brace position), Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Dependency Injection, Favour composition over inheritance, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibilitiy of the code analysed."	Agree
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	

## Response #75

Question	Answer
1. How would you describe yourself as a programmer?	Professional: Senior Developer
2. Which languages do write Object-Oriented code in regularly? Please tick all that apply.	PHP, Java, Javascript
3. Do you use code reviews as part of your workflow?	Yes – I review others work and my own code is reviewed by others
4. Do you use code review tools such as scrutinizer, phpmmd, pmd, etc?	Sometimes
5. During code reviews, or when writing your own code, do you look for any of the following? Please tick all that apply.	Performance Issues, Bugs, User Experience, Security Issues, Code flexibility
6. Are you familiar with any of the following Object-Oriented best practices? Please tick all that apply	Encapsulation, Tell, Don't ask, Law of Demeter (digging into collaborators), Separation of concerns, Dependency Injection, Loose coupling, Favour composition over inheritance, Immutability, Single Responsibility Principle
7. Do you try to follow Object-Oriented best-practices when developing software?	Often
8. Do you actively try to avoid programming practices which go against best practice principles? (For example, do you actively avoid global variables and singletons)	Often
9. Which, if any, programming practices do you actively avoid using (tick all that apply, ignore any you are unfamiliar with)	Global variables, Static variables (including private static variables), Inheritance, Service Locator, God object, Singleton Pattern, Tight coupling, Setter Injection
10. Do you have any comments on the background research of this project (If you didn't read the background research, please enter N/A, if you read it but have no comments, please leave blank)	
11. The insphpect site is intuitive and easy to use	Strongly Agree
12. How much do you agree with the statement: "Overall, the suggestions made by Insphpect are helpful"?	Agree
13. Do you agree with the recommendations made by Insphpect?	Agree
14. How much do you agree with the following statement: "The explanations of why identified bad practices should be avoided are clear and helpful."	Strongly Agree
15. How much do you agree with the following statement: "The grade given is a fair evaluation of the flexibility of the code analysed."	Don't Know
16. How much do you agree with the following statement: "I would like to see a similar tool built for other programming languages."	Strongly Agree
17. Is there anything you think is missing from Insphpect which should be included in a future update?	
18. Do you have any general comments about Insphpect?	