IMPACTS OF MODULAR GRIME ON TECHNICAL DEBT

by

Melissa Renee Dale

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2014

DEDICATION

I'd like to dedicate this thesis to my family, who have always supported my pursuit of higher education, even when it looks like I'll never finish. My parents Michael and Nancy who have always encouraged me to work hard, my brother Nathan for taking me out for a beer when things were rough, and my grandparents - Wayne Dale, Mary Ann Dale, Leon Bowles, and Rita Bowles - who have given me strong roots and the knowledge I'll always have a home, no matter where I may go or what I may do.

ACKNOWLEDGEMENTS

I'd like to acknowledge the people who've made this research possible. First and foremost my adviser Dr. Clemente Izurieta. I am grateful for his patient guidance, encouragement, and useful critiques of this research work. It was not always a smooth journey, but I am very thankful that he gave me the opportunity to take it.

I would like to offer my special thanks to friend and mentor PhD candidate in Industrial Engineering Jessica Mueller who used her statistical expertise to listen to my proposed statistical analysis methods so that I may make sound statistical design decisions, speak and write in a professional manner, and conduct better research.

I owe a thanks to my team at Webfilings, the XBRL services team. They have allowed me a short time to intern with the company and actively apply technical debt management techniques, as well as grow and learn as a software engineer.

Last but not least, I would like to express my very great appreciation to good friend and colleague Lindsey Hanna, who used her experience as an active software engineer in the field to provide technical input on the design of the grime injector tool used in this research, as well as providing insight to technical debt management in the industry.

TABLE OF CONTENTS

TABLE OF CONTENTS - CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES - CONTINUED

Figure Page

LIST OF EQUATIONS

GLOSSARY

ASA	Automatic Static Analysis
BBQ	Browse-by-Query
CCB	Change Control Boards
HOV	Homogeneity of Variance
PEAG	Persistent External Afferent Grime
PEEG	Persistent External Efferent Grime
PIG	Persistent Internal Grime
RE	Repair Effort
RF	Rework Fraction
RV	Rebuild Value
SIG	Software Improvement Group
SS	System Size
SQALE	Software Quality Analysis based on Lifecycle Expectations
TEAG	Temporary External Afferent Grime
TF	Technology Factor
TEEG	Temporary External Efferent Grime
TIG	Temporary Internal Grime

ABSTRACT

The purpose of this research is to study the effects of code changes that violate a design pattern's intended role on the quality of a project. We use technical debt as an overarching surrogate measure of quality. Technical debt is a metaphor borrowed from the financial domain used to describe the potential cost necessary to refactor a software system to agreed upon coding and design standards. Previous research defined violations in the context of design patterns as grime. Because technical debt can ultimately lead to the downfall of a project, it is important to understand if and how grime may contribute to a system's technical debt.

To investigate this problem, we have developed a grime injector to model grime growth on Java projects. We use SonarQube's technical debt software to compare the technical debt scores of six different types of modular. These six types can be classified along three major dimensions: strength, scope, and direction.

We find that the strength dimension is the most important contributor to the quality of a design and that temporary grime results in higher technical debt scores than persistent grime. This knowledge will help to make design decisions which could help manage a project's technical debt.

INTRODUCTION

Design patterns are used in software engineering to reinforce consistent solutions to common problems. However, as a system ages, changes are introduced as a result of bug fixing or new features being added. As systems evolve, the coupling between pattern and non-pattern classes tends to increase and the intended design patterns can become obscured by code that violates the pattern's intended purpose. Unintended additions were defined by Izurieta and Bieman [1] as modular grime.

We are interested in investigating the effects that modular grime may potentially have on the overall quality of a system when quantified as technical debt. Technical debt is a metaphor borrowed from the financial domain and introduced by Ward Cunningham [2]. It describes the amount of work needed to repay the debt incurred by taking shortcuts, such as choosing decisively negative coding practices in order to meet a deadline. We hypothesize that not all types of modular grime have the same impact on the technical debt of a project. To investigate, we use SonarQube [3] to measure technical debt and construct a grime injector to model instances of modular grime.

An overview of technical debt, including how it occurs, proposed methods for measuring it, and management approaches are described in the Background section, as well as a review of research related of design pattern decay and grime. We discuss the process we use to model grime growth and collect technical debt measurements in the Methodology section. In the Results and Analysis section, we analyze the findings of the experiments and discuss Threats to Validity in the following section. Finally, we summarize our findings and propose areas for future research.

BACKGROUND

This research investigates the relationship between technical debt and modular design pattern grime. In this section we discuss the background of technical debt (section 2.1), including a method to estimate technical debt (section 2.1.1) and a tool that reports technical debt (section 2.1.2), as well as information about design pattern grime (section 2.2).

Technical Debt

The term ‘'technical debt”, coined by Ward Cunningham in 1992 [2], describes the cost (which can be measured in terms of dollars or man-hours) that a design decision will cost in the future at the expense of a short term gain. Like financial debt, technical debt is necessary for a product to advance. For example, a software engineer may decide to design a solution that will require reworking in the future. The engineer is aware that it is not the best solution for the health of the system, but it is an intentional decision that must be made in order to meet a release deadline. There was a short term benefit gained by being able to meet the deadline, but in the future the time and effort that was saved will have to be re-invested. In fact, more time and effort may need to be re-invested than if the shortcut was not taken. This additional effort can be thought of as an interest that must be repaid on the gain made by taking the shortcut.

Like financial debt, if a system incurs too much technical debt without a repayment plan, it may become unstable and unable to be modified without significant effort. Ward [2]states “*Entire engineering organizations can be brought to a stand-still* *under the debt load of unconsolidated implementation".* The decision described above to incur intentional debt results in new system debt accumulation which will need to be managed and repaid at some point in the future with interest.

Before a plan to manage technical debt can be implemented, there must first be a way to quantify it. In this study, we focus only on modular grime, a form of technical debt found in designs. We evaluate this grime by evaluating source code using SonarQube [3], which reports technical debt in both man days (how many 8 hour developer days it takes to correct all the identified issues), and in terms of an estimation of how much it will cost the organization to fix those issues in man days.

There are multiple forms of debt, but this research focuses primarily on design debt (sometimes referred to as architectural debt). In 2004 Kerievsky [4] defined design debt as costs associated with architectural negligence. Neill and Laplante [5] identify needs of managing design debt by pointing out that repairing decaying code often requires more strategic approaches that address design deficiencies than simple syntactic issues or coding standards violations.

What is Technical Debt?

What qualifies as "Technical Debt"? In order to investigate the consequences of technical debt we need to first understand more formally what is technical debt and how it occurs.

Kruchten et al. [6] claim that, "Most authors agree that the major cause of technical debt is schedule pressure," although they also point out that other issues can come into play, such as carelessness, lack of education, and basic incompetence. Klinger et al. [7] claims debt is result of stakeholders that lack effective means to communicate.

Fowler [8] presents a formal explanation on how technical debt can occur. He points out an important distinction between prudent debt and reckless debt, as well as deliberate and inadvertent. The quadrant shown in Figure 1 illustrates these concepts.

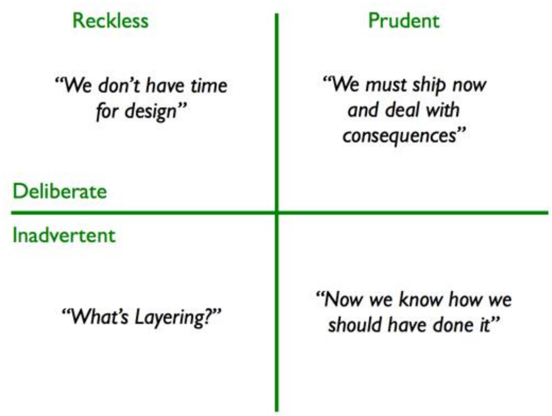


Figure 1: Technical Debt Quadrant [8]

In their study Zazworka et al. [9], find that technical debt has a negative impact on software quality. In other words, if developers desire higher quality software, then technical debt needs be identified and managed closely in the development process.

Quality indicators alone are not sufficient to estimate technical debt. Zazworka et al. [10] compare four different technical debt identification approaches, including code smells, automatic static analysis (ASA) issues, grime buildup (discussed in Background Section 2.2 Design Pattern Grime), and modularity violations. They studied commonalities and differences between these identification techniques, and found that only a small subset of technical debt indicators are related to quality indicators.

How to Measure Technical Debt?

A number of authors have proposed various ways to quantify and measure technical debt. In the following paragraphs we discuss proposed methods for measuring technical debt.

Curtis et al. [11] evaluate technical debt using static analysis of defined good architectural and coding practices that aims to evaluate quality within and across application layers. They then present a formula for estimating the principal in dollars:

TD-Principal =
($\sum$ high severity violations)x.5)x 1hr.)x75$)+
($\sum$ medium severity violations)x.25)x 1hr.)x75$)+
($\sum$ low severity violations)x.1)x 1hr.)x75$)

Equation 1: CAST Equation for Calculating Technical Debt [11]

The ability to customize which violations are considered high severity versus which are considered low severity allows organizations to customize a model to estimate how costly their technical debt is.

Ariadi et al. [12] propose an approach based on an empirical assessment method of software quality developed at the Software Improvement Group (SIG). The core part of the technical debt calculation is constructed on the basis of empirical data of 44 systems that are currently being monitored by SIG. They propose that technical debt may be thought of as the *Repair Effort (RE)*, which can be estimated by using *Rework Fraction (RF) and Rebuild Value (RV).* Where the RF is an estimate of the percentage of lines of code that need to be changed to improve the quality of software to the next quality level (assuming a 5 star quality rating) and RV is an estimate of effort that needs to be spent rebuilding the system. RV is calculated by multiplying the System Size (SS) in lines of code by Technology Factor (TF). The definitions in Figure 2 provide a summary of the equation described above.

$RE = RF * RV * RA$, where
$RF =$ estimate of percentage of lines of codes that need to be changed
$RV = SS * TF$
$RA = \%\ adjustment\ for\ beneficial\ factors, such\ as\ team\ experience$

Figure 2: Nugroho's proposed equations for calculating technical debt [12]

This paper also explores the interest that technical debt occurs. It uses a Maintenance Effort (ME) as a surrogate for interest. $ME = \frac{MF*RF}{QF}$, where MF is the maintenance fraction calculated by historical maintenance information and QF is the quality factor used to account for the level of quality. QF is calculated by $QF = 2^{(\frac{QualityLevel-3}{2}))}$, which gives factors from 1-star to 5-star respectively: 0.5, 0.7, 1.0, 1.4, 2.0.

Groot et al. [13] incorporate Nugroho's methods to determine the production value of software using a Software Value Pyramid. This Pyramid is displayed in Figure 3.

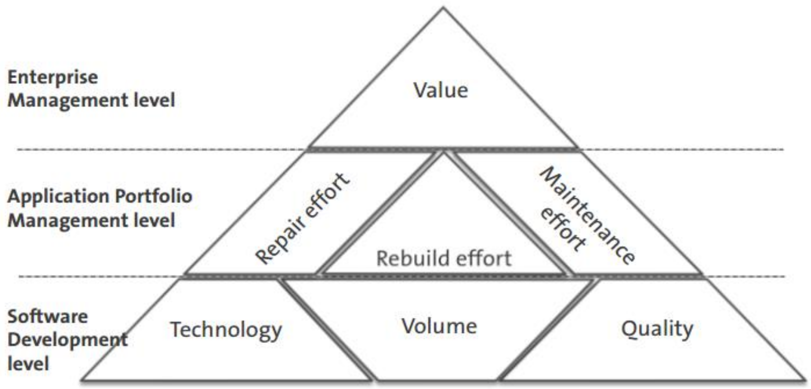


Figure 3: Software Value Pyramid [13]

At the bottom of the pyramid there is the software development level. This level represents the technical state of a system and are the main concern of the software development team: Quality, Volume, and Technology. The next level is the Application Portfolio Management and utilizes Nugroho's equations for calculating technical debt. At the top, the enterprise management level, corporate executives consider software as assets that can be acquired, maintained and exploited, or sold. At this level, the authors propose three models. To illustrate the differences in these models, the authors describe buying a car with a dent.

In the first model the validation model subtracts the repair effort from the repair value, in much the same way one would subtract the cost to repair a dent from the overall cost of a car.

The second model reduces the rebuild value by the fraction of the software system that is of suboptimal quality, like replacing the dented part of the car with a new part altogether.

The third model impairs rebuild value by the increased software maintenance costs due to suboptimal quality. This is analogous to just living with the dent in the car and accepting higher running or maintenance costs.

After applying these three models to a large collection of software, the authors found that all three models report similar values. The authors also conducted several case studies to understand how practitioners view the proposed models. Rather than preferring one model over the other, the practitioners viewed all three models as complementary and improvement over the strictly development cost in evaluating the value of their software.

Another proposed method used to calculate technical debt is the Software Quality Assessment Based on Lifecycle Expectations (SQALE) method. The SQALE method is used in this research. The SQALE method does not account for interest of debt in its calculations and so it may not provide a complete picture of the technical debt of a system. Letouzey [14] presented the SQALE methodology in 2012. SQALE utilizes four key concepts to build a technical debt framework: quality model, analysis model, indices, and indicators.

The SQALE quality model evaluates code quality based on a given set of rules, for example, one rule might state that there should be no commented out blocks of code. The quality model is a hierarchy composed of characteristic, sub-characteristic, and requirement categories. Characteristic and Sub-Characteristic are the categories being evaluated when considering technical debt, such as Maintainability, Readability, Changeability, Security, etc. The Requirement is the rule that the Characteristic and Sub-Characteristic should follow. An example is given in Table 1.

Table 1: Example of SQALE Quality Model

Characteristic	Sub-Characteristic	Requirement
Maintainability	Readability	There is no commented out block of code

The SQALE analysis model uses a normalized remediation index to evaluate how much it will cost to fix the issues reported by the quality model. This model if formed from the rule being checked (Requirement), how to fix the requirement if it is not met (Remediation Details), and an estimate of how long it will take to fix the requirement (Remediation Function). An example of a SQALE analysis model is given in Table 2.

Table 2: SQALE Analysis Model Example

Requirement	Remediation Details	Remediation Function
There is no commented out block of code	Remove (because there is no impact on compiled code)	2 minutes per occurrence

The SQALE Indices are a number of indices that connect data. The main index is a global quality index that connects source code artifacts to the sum of remediation indices (as defined by the remediation function in the SQALE Analysis Model) relating to the characteristics of the quality model. SQALE also provides indices for testability, reliability, changeability, efficiency, security, maintainability, portability, and reusability.

The SQALE Indicators highlight potential areas of concern in a system. They are used for analysis and visual representations, such as dashboards. Two examples given in Letouzey's paper are Rating and SQALE Pyramid. Rating is a high level indicator suggested by Gat [15] that visualizes the ratio between technical debt and development cost. The SQALE Pyramid is used to visualize the distribution of technical debt over the quality model. Figure 4 depicts an example given by Letouzey of Rating indicator (left) and a SQALE Pyramid (right).

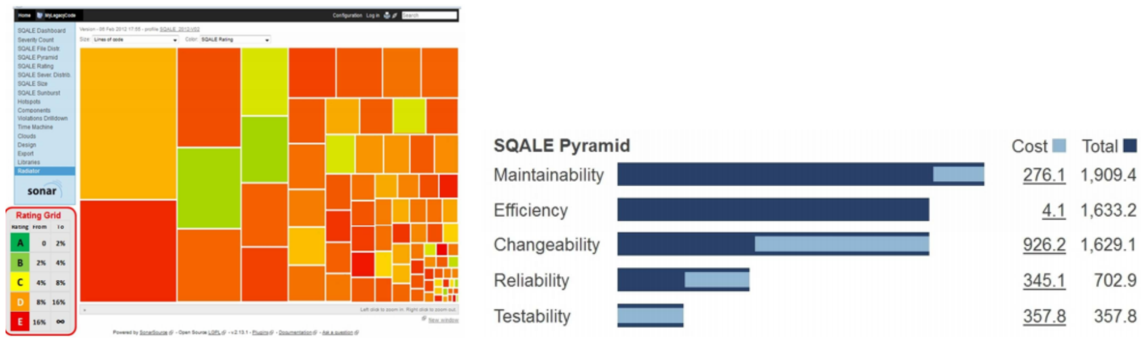


Figure 4: Example of Rating Indicator and Sonar Pyramid [14]

The tool used in this research to calculate technical debt of a project is SonarQube [3]. SonarQube utilizes the SQALE methodology to measure a source code's technical debt. The baseline set of expectations in SonarQube are referred to as the "Developers' Seven Deadly Sins", which are: Bugs and Potential Bugs, Coding Standards Breach, Duplications, Lack of Unit Tests, Bad Distribution of Complexity, Spaghetti Design, and Not Enough or Too Many Comments. Each of the sins are tracked through rules defined in SonarQube's "Quality Profile" setting.

The "Quality Profiles" settings in SonarQube corresponds to the SQALE Quality Model. Figure 5 displays the example Quality Profile for Java given on the SonarQube documentation website [16]. A complete list of rules being checked can be found in Appendix A.

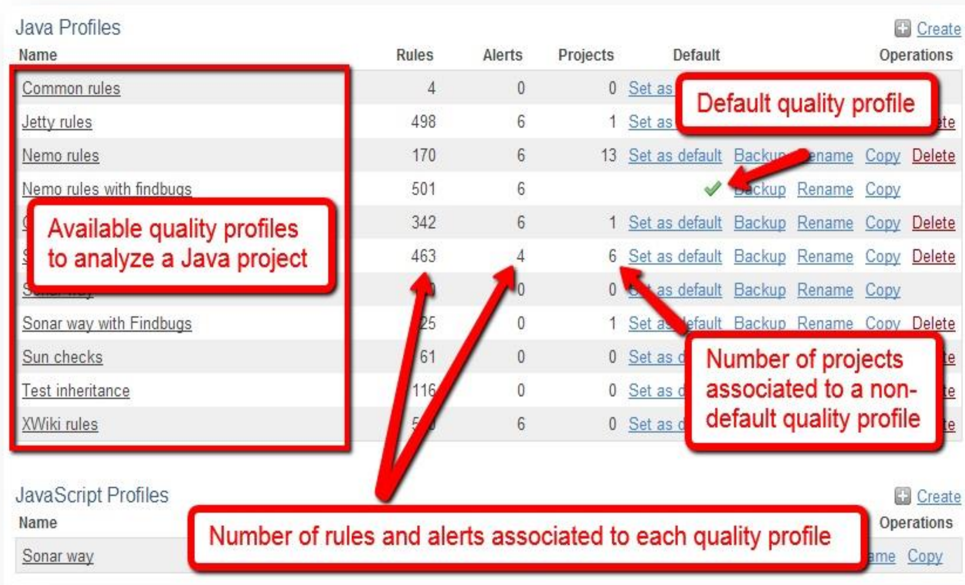


Figure 5: Screenshot of SonarQube Quality Profile [14]

Every time SonarQube finds an instance which does not conform to the rules given in the Quality Profile, it raises an issue. The technical debt value for each issue is set at the rule level of the Quality Profile and is defined by seasoned professionals [17]. The commercial version of SonarQube allows for organizations to define technical debt values that are individualized, but for the purposes of this research, the default values are appropriate for our exploration. These costs relate to the remediation functions of the SQALE Analysis Model. Technical debt is then calculated by summing the technical debt accrued by each issue.

Managing Technical Debt

It is unrealistic to think that developers can simply fix all technical debt artifacts as they are discovered. The following section examines some proposed methods to manage technical debt and incorporate repayment plans in the planning stages.

With so many different technical debt aspects, how do we know how to manage it all? Brown et al. [18] lay the groundwork for understanding the need to manage technical debt. They pose open research questions, including refactoring opportunities, architectural issues, and identifying dominant sources of technical debt, as well as issues that arise when measuring technical debt.

Zazworka et al. [19] and Seaman et al. [20] explore design debt through use of a God class to answer how to prioritize and decide where to refactor based on estimating cost and impact of the refactoring. Zazworka et al. [19] propose a method using cost benefit matrices of refactoring effort and quality impact to help identify which refactoring activities should be performed first because they are likely to be cheap to make have significant effect, and which refactorings should be postponed due to high cost and low payoffs. Seaman et al. [21] expanded on the authors' initial work to include four approaches to incorporate technical debt information into decisions made for release planning. These four approaches are Simple Cost-Benefit Analysis, Analytic Hierarchy Process, Portfolio Approach, and Options.

Simple cost-benefit analysis approach makes use of the cost-benefit matrices discussed above. Analytic hierarchy process involves building a criteria hierarchy of quantitative and qualitative criteria, assigning weights and scales to the criteria, and performing a series of pair wise comparisons between the alternatives against the various criteria. The portfolio approach relates to the financial domain in which investors apply risk management strategies to maximize their return on investment. This approach can be applied to technical debt management by determining the types and amounts of assets that should be invested or divested and when the actions should occur to maximize the return on investment. Lastly the Option approach considers investment in refactoring as analogous to purchasing the option that will allow changes to be made in the future, but with no immediate profit gained. While all approaches consider principal and interest, all require different input from the user, and further investigation needs to be conducted to determine differences in the application of these approaches to the decision making process.

Snipes et al. [22] propose using Software Change Control Boards (CCBs) based on a set of decision factors. A Software Change Control Board is a committee of stakeholders that make decisions regarding whether or not proposed changes to a software project should be implemented. The aim of the study was to determine how a model of cost and benefits of incurring technical debt could be part of the CCB decision process. The authors identified the cost categories and decision factors for fixing and deferring defects as a result of interviews with CCB members and found that the decision factors could incorporate the financial aspects when using the technical debt metaphor.

Ernst [23] explores measuring technical debt in requirements as the distance between the implementation and the actual state of the world. Using the requirements modeling tool RE-KOMBINE, the author represents technical debt using the notion of optimal solutions to a requirements problem.

Technical Debt in Industry

While technical debt is being actively researched in academia, there is also a growing interest in technical debt in industry. The following paragraphs explore how technical debt is being managed in practice and what lessons have been reported by those actively evaluating and managing technical debt on real world systems.

To bridge the gap between theory and application, Lim et al. [24] conducted an interview study to review how software practitioners perceive technical debt and understand the context in which technical debt occurs. After conducting interviews with 35 practitioners, they found that 75 percent of participants weren't familiar with the term "technical debt". After explaining the metaphor in terms of tradeoffs and shortcuts, most participants recognized and understood it immediately. The authors compiled the participants' strategies for dealing with technical debt. The list includes doing nothing, allocate some percentage of each release cycle to addressing technical debt, manage stakeholders' expectations by being open about debt's implications, and conduct audits with entire development teams to make technical debt visible and explicit.

Morgethaler et al. [25] discuss how Google approaches technical debt. Google uses a variety of methods to pay off technical debt, including special Fixit days and teams dedicated to locating and refactoring. For this study, they focus on the technical debt in their build system. They found this debt hurts the company in two ways. First, it results in lower productivity of engineers because of slower builds, brittle targets, and maintenance of abandoned or broken libraries. Second, this debt results in increased computation costs of the build and test infrastructure because of building and running unnecessary code and tests. Furthermore, they suggest that prioritizing and dealing with technical debt cannot always be left to individual teams, since many engineers resist these efforts on the grounds that it would slow them down or encourage code duplication.

The 2011/12 Crash report [26] evaluates structural quality of business application software. They found on average there is $3.61 of debt per line of code, which means $361,000 of debt for 100,000 lines of code. CAST has released a brochure to illustrate their method of calculating technical debt described in Measuring and Managing Technical Debt with CAST AIP [27]. Figure 6 displays the approach CAST takes to calculating technical debt.

A Technical Debt Framework

In 2013, Tom et al. [28] proposed an encompassing framework of technical debt based on a comprehensive survey of current literature. The framework categorizes technical debt across dimensions and attributes, and explores proposed management through precedents and outcomes.

The framework proposes dimensions of code debt, design and architectural debt, environmental debt, documentation debt, and testing debt. It defines technical debt attributes as monetary cost, amnesty, bankruptcy, interest and principal, leverage, and repayment and withdrawal.

Technical Debt Calculation

Our approach for calculating Technical Debt is defined below:

1. The density of coding violations per thousand lines of code (KLOC) is derived from source code analysis using the CAST Application Intelligence Platform. The coding violations highlight issues around Security, Performance, Robustness, Transferability, and Changeability of the code.

2. Coding violations are categorized into low, medium, and high severity violations. In developing the estimate of Technical Debt, it is assumed that only 50% of high severity problems, 25% of moderate severity problems, and 10% of low severity problems will ultimately be corrected in the normal course of operating the application.

3. To be conservative, we assume that low, moderate, and high severity problems would each take one hour to fix, although industry data suggest these numbers should be higher and in many cases is much higher, especially when the fix is applied during operation. We assumed developer cost at an average burdened rate of $75 per hour.

4. Technical Debt is therefore we calculated using the following formula: Technical Debt = (10% of Low Severity Violations + 25% of Medium Severity Violations + 50% of High Severity Violations) * No. of Hours to Fix * Cost/Hr.

Figure 6: CAST method for calculating technical debt [26]

The authors also investigated precedents that influence how organizations take on technical debt. Pragmatism and prioritization are two such precedents, as well as development processes, attitudes, and ignorance and oversight.

Design Pattern Grime

In studying design pattern decay, two key concepts are rot and grime, as identified by Izurieta and Bieman [29]. Rot is the breakdown of structural integrity of a design pattern realization. The term “grime” refers to the accumulation of code that violates the intended role of the design pattern, but does not break the structural integrity of that design pattern. Rot and Grime are mutually exclusive.

Three types of grime were defined by Izurieta and Bieman [1]: organizational, modular, and class. Organizational grime refers to the organization of the files and namespaces that make up a pattern. Class grime refers to individual classes that make up a pattern. This study focuses on modular grime, which refers to coupling between pattern classes or pattern classes and non-pattern classes which violate the pattern’s intended purpose. Izurieta and Bieman depict the landscape of design pattern rot and grime using a Venn diagram, depicted in Figure 7.

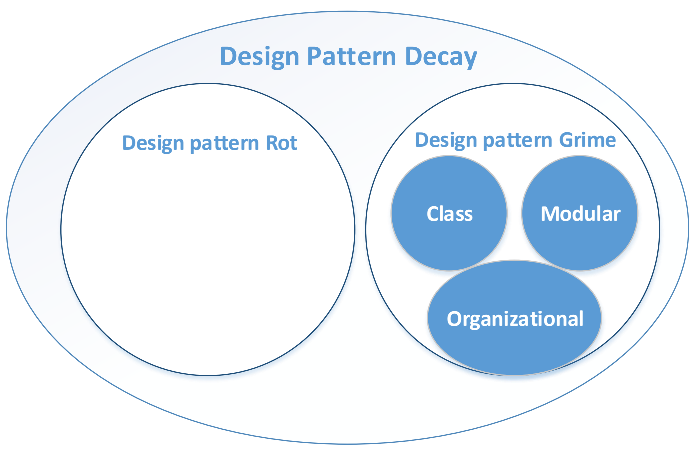


Figure 7: Landscape of Design Pattern Rot and Grime [1]

Schanz and Izurieta [30] defined taxonomy for modular grime along three dimensions: the scope of the coupling, the direction of the coupling, and the strength of the coupling.

Scope: Internal or External

The scope of the coupling refers to where the coupling occurs. If both classes that are coupled reside in the design pattern, the scope is internal. If the coupling connects a non-pattern class to a pattern class, the scope is external.

Direction: Efferent or Afferent

If the grime connects a pattern class to a non-pattern class, the direction of that coupling is classified according to its origination source. An instance of grime that originates inside a pattern and forms a relationship with a non-pattern class, is referred to as efferent. If the grime originates outside of a pattern and forms a relationship with a pattern class, then the grime is referred to as afferent.

Strength: Temporary or Persistent

Strength refers to the difficulty of removing the coupling [31]. Strength may be either temporary or persistent. In temporary couplings, a class A uses a method with a parameter, a return value, or a local variable of another class B. Persistent couplings occur when a class A contains an attribute of class B.

Using these dimensions, Schanz and Izurieta defined six types of grime: Persistent External Afferent Grime (PEAG), Persistent External Efferent Grime (PEEG), Persistent Internal Grime (PIG), Temporary External Afferent Grime (TEAG), Temporary External Efferent Grime (TEEG), and Temporary Internal Grime (TIG). The diagram in Figure 8 depicts the structure of the taxonomy.

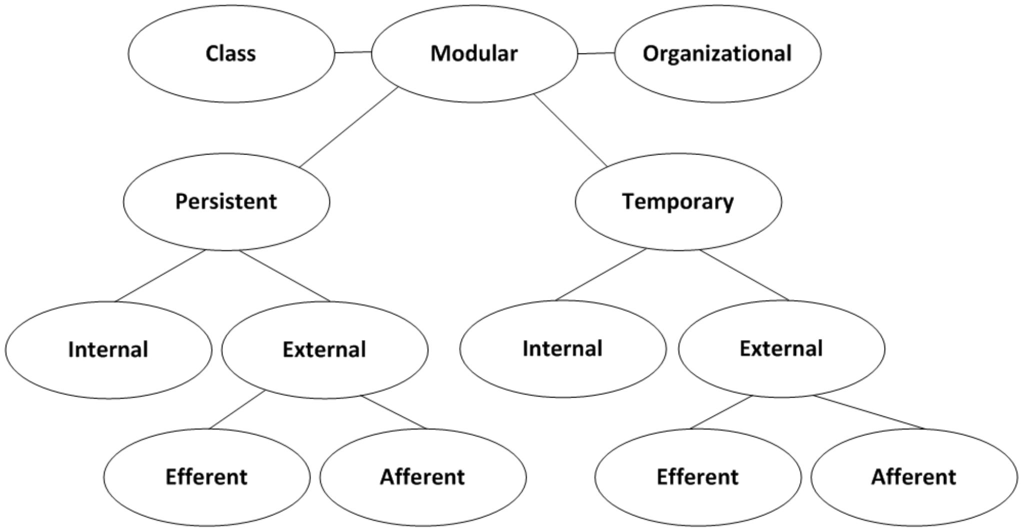


Figure 8: Grime Taxonomy defined by Schanz and Izurieta [30]

Schanz and Izurieta conducted a pilot study using Vuze, a peer-to-peer file sharing client that uses the bittorrent protocol over eight versions. They used a Browse-by-Query (BBQ) plugin for eclipse to determine changes in the number of grime couplings between versions. However, BBQ does not allow the user to differentiate between internal and external scope of couplings. Therefore changes in PIG couplings are reflected in PEAG and PEEG, and changes in TIG couplings are reflected in TEAG and TEEG. After analyzing grime counts over eight versions and 38 months of development, the authors found that in VUZE software instances of TEEG, TEAG, and PEAG tended to increase, while PEAG did not.

PROBLEM STATEMENT

The relationships between technical debt and grime are important to understand when considering the role grime plays in the technical debt of a system. Izurieta et al. [32] identify design pattern grime as a component of the technical debt landscape.

Some initial work has been done to understand the negative impact of grime. Izurieta and Bieman [33] find that as grime grows, so do testing requirements, which can negatively impact system testability. Research to quantify grime in terms of technical debt does not exist. This research will take the first steps in quantifying the effects of modular design pattern grime on technical debt.

METHODOLOGY

Modeling Grime Growth

To study differences in the effects of different types of modular grime on technical debt, we will first model the growth of modular grime on Java projects that use design patterns to produce modified Java projects that can then be used to obtain and analyze technical debt scores.

In order to model grime growth, we take a clean Java project and then create a modified copy for each of the different types of modular grime defined by Schanz and Izurieta [30]. The details of this modification process are described in the following sections, but at a high level we model modular grime by creating couplings between classes that represent that grime type. The process of injecting these couplings and how these couplings differ for each type of modular grime are discussed in the Injection section 4.3.3.

Javassist [34] is used to modify Java programs. It is a class library that allows a developer to edit bytecodes in Java. Using Javassist, we developed a java injector program to modify a given class file's bytecode. Javassist files need modification before they can be analyzed, we describe those modifications and then describe how the grime injector manipulates class files to represent grime growth.

Javassist

When a program is written in Java it is saved to a .java file. When that code is compiled, it is compiled to bytecode for the Java virtual machine (JVM) to execute. This bytecode is saved in a class file (.class) that is executed by the JVM.

To edit a specific class, Javassist examines the JVM path to locate the bytecode of that class. Once it finds the bytecode, the Javassist API can be used to modify the class. For example if you wanted to edit a class named HelloWorld.java, you can use the get() method API of Javassist to locate HelloWorld.class. Once Javassist has a reference to the class file, it is possible to modify the bytecode, including changing existing methods or adding new methods and variables.

The modified bytecode class file can be decompiled back to a .java file. JAD [35] is a freeware java decompiler which takes class files and decompiles them back to java files, which can be analyzed using tools such as SonarQube. JAD is discussed further in the JAD subsection.

Figure 9 shows a diagram of the process described above. We start with a HelloWorld.java source file, and compile it to bytecode (.class), which if executed by the JVM would print “Hello World” to the terminal. However, if we modify the file HelloWorld.class with the injector, we can produce modified bytecode that can be executed on the JVM and would now print “Hello Universe” to the terminal. To analyze the equivalent source (.java) file of a modified bytecode file, we must run it through a decompiler to produce the modHelloWorld.java file.

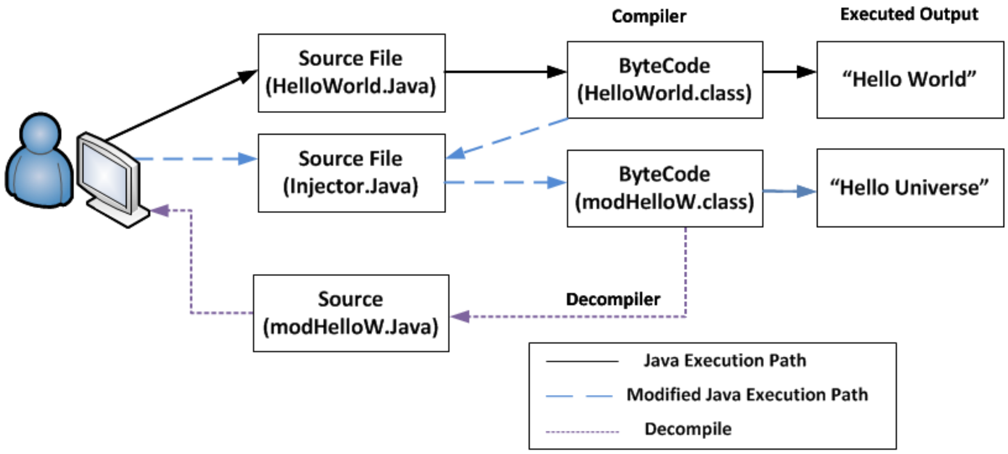


Figure 9: Diagram of Java compilation and decompilation process

Grime-Injector

The tool used to model grime growth is herein referred to as the grime injector. It is written in Java [36]and uses Javassist to perform all grime growth simulations. The following subsections describe how the grime-injector works, including necessary inputs, initialization, a description of how it performs the injections, and outputs. Finally an example is given to illustrate all the aforementioned steps.

Input

To model grime growth, the user of the injector must provide the following information. These items may be specified through the injector GUI, discussed towards the end of this section.

Pattern Class Names and Non Pattern Class Names. The injector uses an arraylist of strings that describe the pattern class names and an arraylist of strings that describe non-pattern classes. Once the string arrays are passed to the injector, Javassist uses the names of these classes to select the corresponding bytecode and create an arraylist of pattern class bytecode files and an arraylist of non-pattern class bytecode files.

Number of Grime Instances. The injector uses an array of integers to specify the number of grime instances to be injected. The array has size six, where each indexed value represents a different type of modular grime (modular grime types are defined in the BACKGROUND section). For example, if the user wants 10 instances of each type of modular grime, then they would pass in an array of 10s [10,10,10,10,10,10]. Values are given in alphabetical order, so if a user wanted to only model 10 instances of PEEG grime type, they would pass in an array with only one 10 in the third index and the rest 0's [0,0,10,0,0,0]. Using the GUI, the user can explicitly state the numbers of each grime type (or a number for each). The GUI will then pass the appropriate array to the injector.

Number of Runs (Repeats). This is an optional parameter integer that specifies the number of times to repeat the injections. This is useful when running experiments and multiple sets of modified projects need to be obtained, such as for running statistical analysis to determine means or determining statistical differences. The default value of this parameter is 1.

Number of Versions (Iterations). The version option is intended to represent the growth of grime over iterations of software. The injection begins by performing the expected number of injections and outputting the injected bytecode into the appropriate directory (the directory structure is explained below). Before exiting the program, the injector will feed the outputted bytecode back into the injection process and inject over the previously injected code thus compounding the grime. It continues this process for the number of specified iterations before moving onto the next run. If no number of versions is specified, the default value is 1.

Initialization

The injector performs a series of initialization steps. First an integer variable is injected into every class file. This variable in injected so that when performing temporary grime injections, the program can inject a variable that is guaranteed to exist.

Because the grime injector cannot at this time handle classes with non-empty constructors, the injector catches the exception that arises when attempting to inject a persistent grime type and it will add an empty constructor to the class. This works because Java allows constructors to be overloaded. For example, a java class with a constructor like: ***Foo(int bar)*** would throw an exception if Javassist attempted to initialize an instance of that class because it does not have the required parameters to initialize it. To avoid this exception, another constructor may be added to class ***Foo*** so that it may be initialized by simply calling ***Foo()***.

Six copies of the pattern and non-pattern initialized bytecode arrays are made, one for each modular grime type. These six identical copies serve as the clean foundations for the modular grime to be modeled. The injector has been designed such that it will be possible in the future to have the option to overlay all the different types of grime on top of each other in a program.

<u>Injection</u>

This processes makes use of the grime taxonomy described in the background section; coupling strength (temporary or persistent), the scope of the grime (internal or external), and the direction of the grime (efferent or afferent). All the types of grime are injected with the same method: *couple (class to, class from, char strength)*.

The strength of the grime is handled through a *char* variable in the couple method. If a “t” or “T” is passed in, the coupling is temporary and a local variable of the “from” class type will be injected into the “to” class, creating a temporary coupling. If a “p” or “P” is passed in, the coupling is persistent and an attribute of type “from” class will be injected into the “to” class. Figure 10 depicts the strength relationship between the ‘from’ and ‘to’ class.

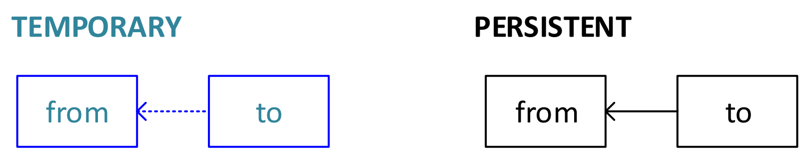


Figure 10: Strength of Coupling

The scope and direction can both be handled with the “to” and “from” classes in the couple method. The coupling is performed by taking an instance of the “from” class and injecting it into the “to” class file. This coupling will either be created by using an attribute of type “from” class or a local variable of the “from” class depending on the strength defined in the *couple* method (as described above).

If the scope is internal, the origin and the destination are irrelevant because both are in the pattern itself. If the direction is afferent, a pattern class is randomly chosen and injected as a “from” class into a randomly selected “to” class from the non-pattern arraylist. If the direction is efferent, the “to” class is randomly selected from the pattern class array and injected (depending on the strength defined in the *couple* method) into a class randomly selected from the non-pattern class array. Figure 11 displays the scope and direction relationships for each strength type.

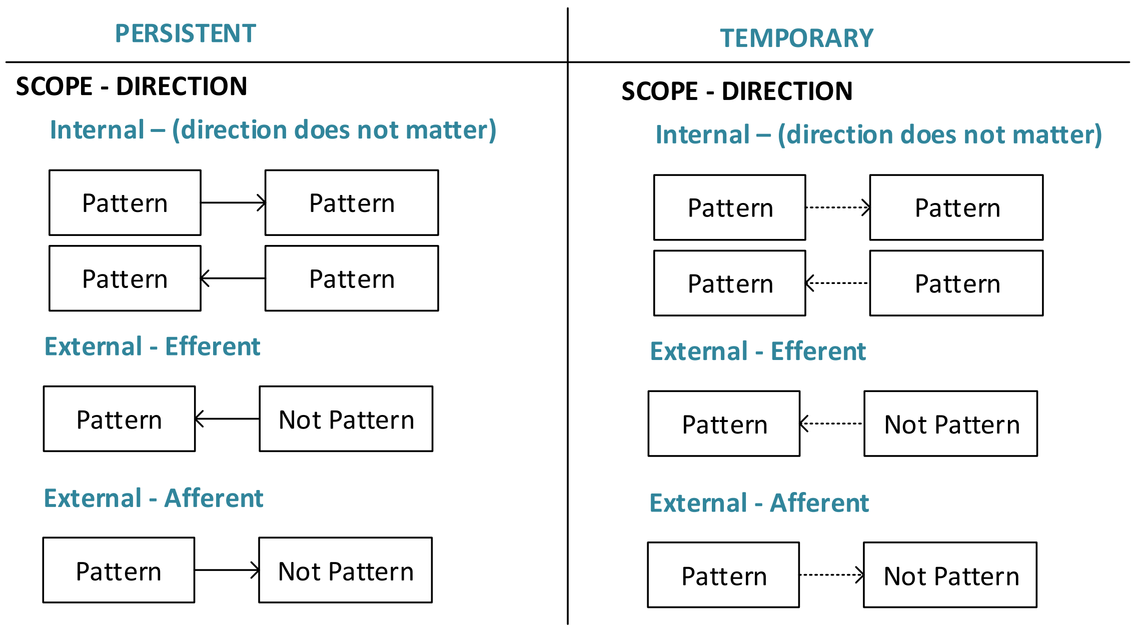


Figure 11: Scope and Direction of Couplings

Overview of Injection Process

Figure 12 depicts the coupling process for *couple(to, from, strength)* for each grime type. For each instance of grime, the *couple* method:

1. Randomly selects a “to” class (from the pattern-class array if direction is internal or afferent, otherwise from the not-pattern-class aray if direction is efferent).
2. Randomly selects a “from” class (from the not pattern class array if direction is afferent, otherwise from the pattern class aray if direction is efferent or if the scope is internal).
3. If strength is persistent, an attribute of type “from” class will be inserted into the “to” class. Else if the strength is temporary, a local variable of the “from” class will be inserted into the “to” class.

<u>Output</u>

Once the injector has completed the modifications, it outputs the modified class files to a “Results” directory in the project’s directory hierarchy. The “Results” directory contains several layers of subdirectories based on the variables passed into the injector.

The first level of subdirectories is the run directories. Each time the injection is repeated (specified by the parameter number of runs) a separate directory is created for the results of each run. Within each run directory, there are versions subdirectories (if more than one version is specified). Lastly, each array of the project’s modified bytecode is written to the appropriate grime type directory, where it is ready to be decompiled by JAD. For each manipulated project, a sonar-properties properties file is generated so that SonarQube may be launched against all the results with a script (a full explanation of this process is given in the SonarQube subsection of the Methodologies section). A diagram of the described directory hierarchy is displayed in Figure 13.

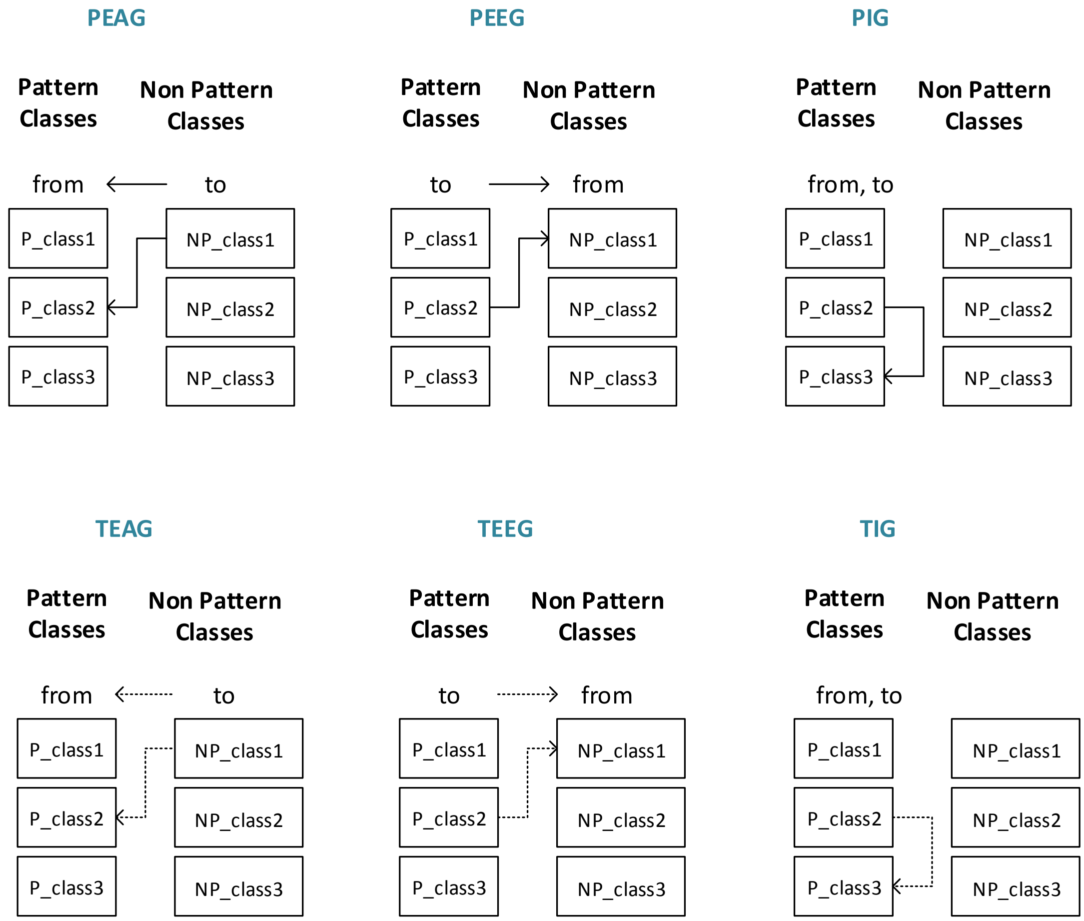


Figure 12: Overview of couple (to, from, strength) for Each Grime Type

JAD

JAD [35] is a command-line java decompiler. Once there is a “Results” directory populated with modified .class files and the injection manipulation process has finished, a batch file is executed that recursively traverses every directory, decompiling each .class file into a .java file of the same name using JAD. Once it has traversed all available directories, there are java files and class files available for analysis. The result is a set of modified java source files that may be analyzed for grime-related and technical debt metrics.

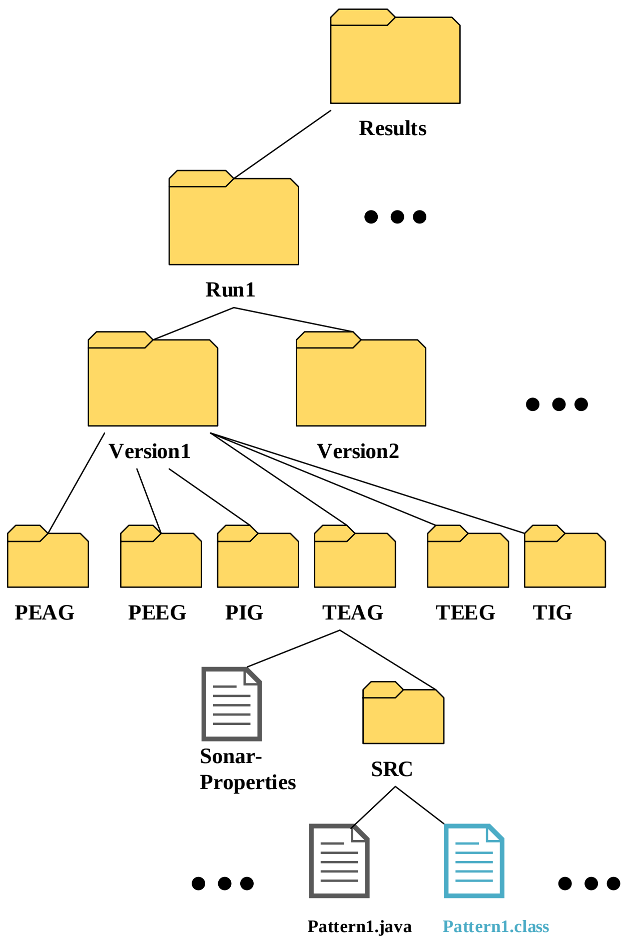


Figure 13: Diagram of outputted directory structure

Modifying Java Projects

The injector is run directly from Eclipse [37] as a Java project. To use the injector, the user simply drops the experimental objects (i.e. the java files) into the “analyze_this” package of the grime-injector program in Eclipse and then runs the GUI.java file.

Graphical User Interface

The grime injector uses a graphical user interface (GUI) to allow the user to specify the desired details of modeling grime growth. The user enters the pattern and non-pattern class names, and the GUI will confirm if it is able to discover the requested classes by displaying the class names in green if it was able to find them and in red if it was unable to locate them. The user can then specify the specific numbers representing each type of grime, or give one number for each grime type. Lastly, the user specifies the number of runs and versions. If these fields are left blank, the default values are set to 1.

Once the user has specified all parameters, they simply click the “Inject” button, and the injector launches. The bytecode is modified and outputted in accordance to the methodology described above. Once the bytecode has been manipulated, the JAD script is automatically launched to decompile the modified bytecode. A Results folder is now in the top level directory of the grime injector and is ready to be used to for analysis.

SonarQube

Once the modified projects are completed, we are ready to evaluate the associated technical debt scores using SonarQube [3]. SonarQube is composed of two pieces: SonarQube server and SonarQube Runner. To collect the scores from the Results directory outputted described in section 4.3.5, the user must:

1. Launch the SonarQube server. The user launches the SonarQube server StartSonar.bat from the command line. Now the user is ready to see the

technical debt analysis output from SonarQube Runner by navigating to http://localhost:9000 in their browser.

2. Launch SonarQube Runner. To perform the technical debt calculations, the SonarQube Runner must be run against a project which has an accompanying sonar-properties.properties file. Similar to the SonarQube server, the SonarQube runner is launched by a batch file from the command line (sonar-runner.bat). An example of a sonar-properties file is given in Figure 14.

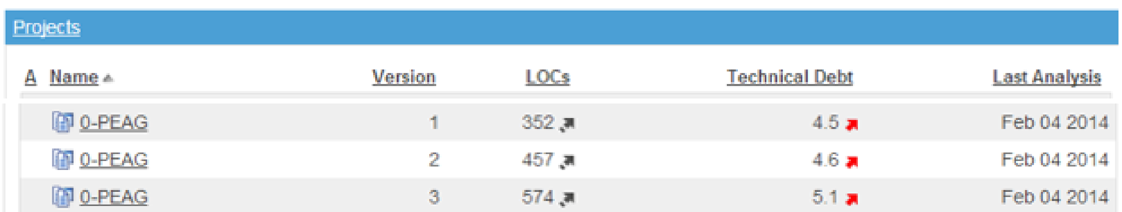
sonar-project.properties

```
# Required metadata
sonar.projectKey=my:project
sonar.projectName=My project
sonar.projectVersion=1.0

# Paths to source directories.
# Paths are relative to the sonar-project.properties file. Replace "\" by "/" on Windows.
# Do not put the "sonar-project.properties" file in the same directory with the source code.
# (i.e. never set the "sonar.sources" property to ".")
sonar.sources=srcDir

# The value of the property must be the key of the language.
sonar.language=cobol

# Encoding of the source code
sonar.sourceEncoding=UTF-8

# Additional parameters
sonar.my.property=value
```

Figure 14: Example Sonar-properties.properties file.

During the injection process described in section 4.3, a unique Sonar-properties.properties file is created for each modified project. Included in the Injector project is a script that recursively traverses the Results directory until it finds a Sonar-properties.properties file, at which point it will launch the SonarQube Runner against the project in that directory. This allows the user to run one script and obtain a technical debt score for each modified project. When the script has finished, the user navigates to http://localhost:9000 and sees the results that SonarQube Runner has collected. The dashboard lists each modified project and the user may investigate individual modified projects by clicking on the link in the dashboard.

A small portion of the dashboard is given in Figure 15. In this figure, we see the results for PEAG grime modeled over three versions. The "0-PEAG" indicates this is the first run for a PEAG model. If the Results directory has multiple runs, the next run would be named "1-PEAG" and so on.



Projects

A Name	Version	LOCs	Technical Debt	Last Analysis
0-PEAG	1	352	4.5	Feb 04 2014
0-PEAG	2	457	4.6	Feb 04 2014
0-PEAG	3	574	5.1	Feb 04 2014

Figure 15: SonarQube Dashboard

Example

Let's say a user wishes to model the growth of TEAG on a program modeled on the science fiction television series Star Trek. The user plans to model grime growth over 3 version releases and then run SonarQube against the modified projects to see if the technical debt score reported increases after the injection of 5 TEAG grime instances on each version.

The user wants to repeat this experiment 5 times to obtain an average technical debt score. Repeating the injection process 5 times will result in 5 modified projects.

Each modified project starts from the same clean foundation and will have the same number of grime instances injected into it, but because the “to” and “from” classes are randomly selected for each grime instance, there may be variability between each of the 5 modified projects.

First the user places a copy of the StarTrek program into the injector’s “analyze_this” package in Eclipse, and then runs GUI.java to specify the details of their desired grime growth model.

The first step is setting up the array of pattern classes and array of non-pattern classes. The user successfully enters Kirk and Romulan (the injector is able to locate Kirk.java and Romulan.java as indicated by the green font), but when the user attempts to enter Klingon as a non-pattern class, the GUI echoes Klingon in a red font, which indicates it is not able to locate Klingon.java and will ignore this entry. Next the user specifies the number of TEAG instances to be injected (per version) while leaving the rest of the fields as blank, indicating they should be 0, and enters 5 into the runs field and 3 into the versions field.

Once the fields are entered, the user clicks the “Inject” button and the grime injector takes over. For simplicity, we exemplify the process using only one pattern class (Kirk.java) and one non pattern class (Romulan.java). The injector will first load the Kirk.class file into the pattern class array and the Romulan.class file into the non pattern class array.

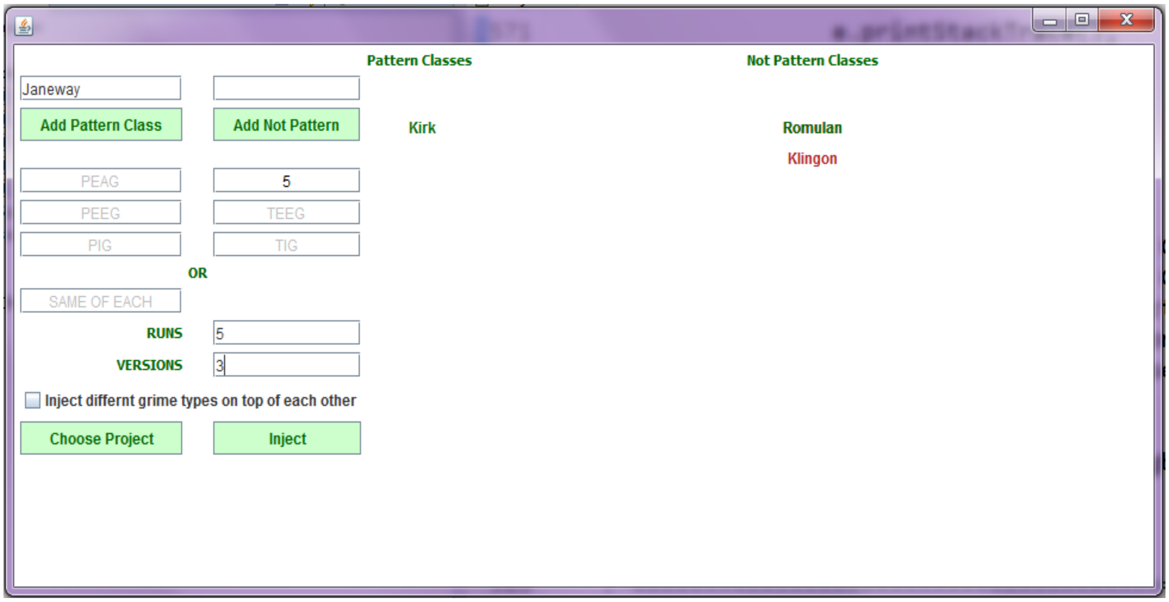


Figure 16: Screenshot of Injector GUI

Next, the injector will perform the initialization steps described in the Initialization subsection. Only one copy is created because the user has specified they are only interested in investigating TEAG. If the user had desired to investigate all types of modular grime, 6 copies would have been created.

For each instance of TEAG we intend to model, a pattern class is randomly chosen and a non pattern class is randomly chosen by the injector. In this case, the user has stated there is 5 instances of TEAG modeled. Because the strength of TEAG is temporary, and there is only one pattern class (Kirk.class) and one non-pattern class (Romulan.class), the injector will use the local variable of Romulan class that was created in the initialization steps and inject it into the Kirk class. This action will be performed 5 times – one time for each TEAG instance specified by the user. To keep collisions from occurring, the injected variable is given the name v#grimed#, with the first # representing the current version number and the second # representing the grime instance number.

After the first round of injections, the following variables are injected: v1grimed1, v1grimed2, v1grimed3, v1grimed4, and v1grimed5. Once injection for this version is complete and written to the Version1 directory of the Results directory, the modified bytecode is inserted into the injector again, and 5 new instances of TEAG couplings are injected overtop of the previously injected code.

Table 3 shows the all the variables created during this process for a single run. Each run will produce the same variable names for each version because each run starts from the clean foundation and there is no danger of collisions between variables of the same name.

Table 3: Injected Variable Names for Version and Instance Number

Version	Injected Variable Names
1	v1grimed1, v1grimed2, v1grimed3, v1grimed4, v1grimed5
2	v1grimed1, v1grimed2, v1grimed3, v1grimed4, v1grimed5, v2grimed1, v2grimed2, v2grimed3, v2grimed4, v2grimed5
3	v1grimed1, v1grimed2, v1grimed3, v1grimed4, v1grimed5, v2grimed1, v2grimed2, v2grimed3, v2grimed4, v2grimed5, v3grimed1, v3grimed2, v3grimed3, v3grimed4, v3grimed5

Now that all the instances for each version has been injected, the injector reverts back to the original unmodified bytecode and performs all the above steps again for the next run. This will happen 5 times in this example, as the user specified this injection process to repeat 5 times.

To perform analysis on the modified bytecode, the user will open the Results folder and see the following hierarchy:

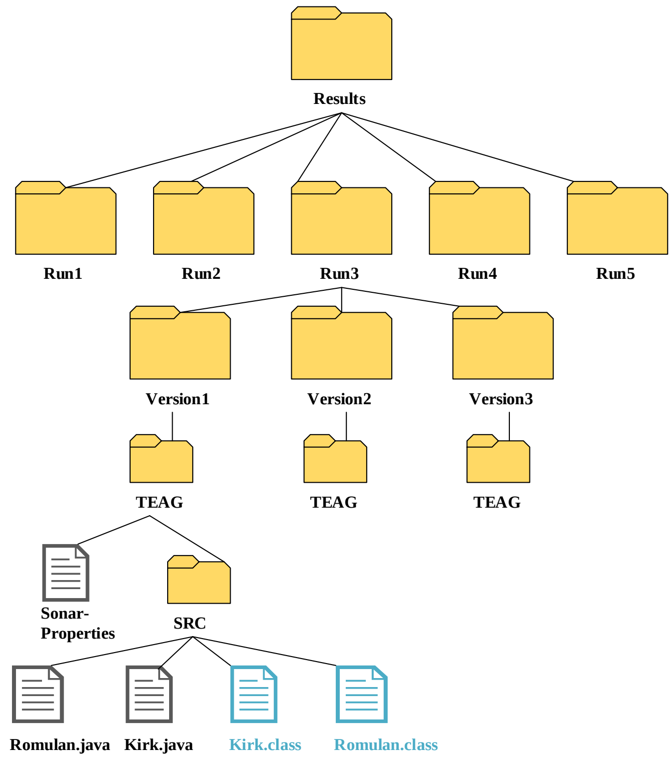


Figure 17: Outputted directory structure for example

The user is now ready to run SonarQube against these modified projects. They start the SonarQube server by running StartSonar.bat from the command line. Next the user launches the sonar_drilldown script included in the Injector package. Once sonar_drilldown has finished, the user can now go to http://localhost:9000 and collect technical debt scores for each of the modified projects.

EXPERIMENT

We investigate the following research question: is there a difference in the technical debt scores reported by SonarQube for the different types of modular grime? Our hypotheses are:

$H_o: \tau_{peag} = \tau_{peeg} = \tau_{pig} = \tau_{teag} = \tau_{teeg} = \tau_{tig}$. That is, there is no difference in the treatment effects of the six different types of modular grime on technical debt.

$H_\alpha: \tau_i \neq \tau_j$ where $i \neq j$. There exists some modular grime type i whose effect on technical debt is different from some other modular grime type j.

Experimental Units

The experimental units are simple programs used to teach design patterns to a software engineering course. We use three kinds of design patterns, one for each of the categories of design patterns: behavioral, structural, and creational [38].

Behavioral design patterns help facilitate communications between objects. For this experiment, an observer pattern is used as the behavioral block. The generic UML for this design pattern is given in Figure 18 and the UML for the implemented pattern used in this experiment is given in Figure 19.

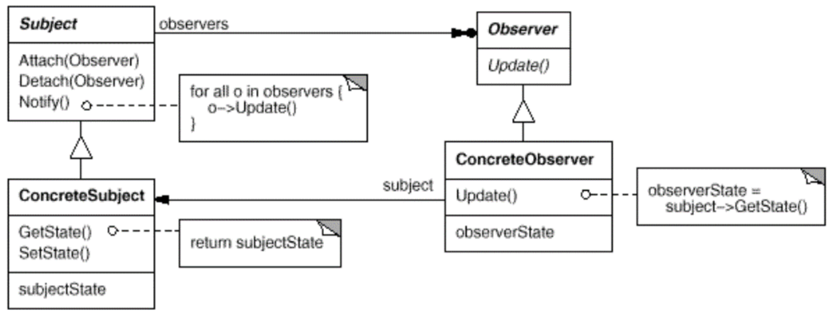


Figure 18: Generic Observer Pattern UML as defined in Design Patterns: Elements of Reusable Object-Oriented Software [38]

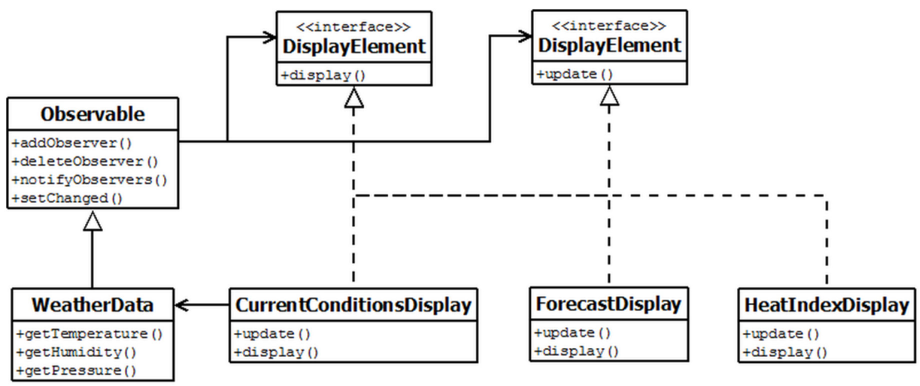


Figure 19: UML Diagram of Implemented Observer Design Pattern [39]

Structural design patterns define structures that enable creation of objects and additional functionality to the objects. For this experiment, a decorator design pattern is used as the structural block. The generic UML for this design pattern is given in Figure 20 and the UML for the implemented pattern used in this experiment is given in Figure 21.

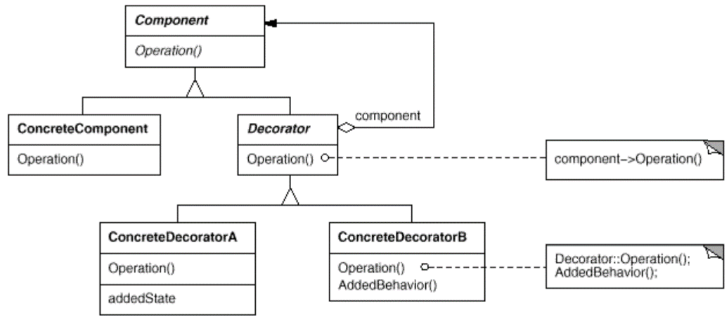


Figure 20: Generic Decorator Pattern UML as defined in Design Patterns: Elements of Reusable Object-Oriented Software [38]

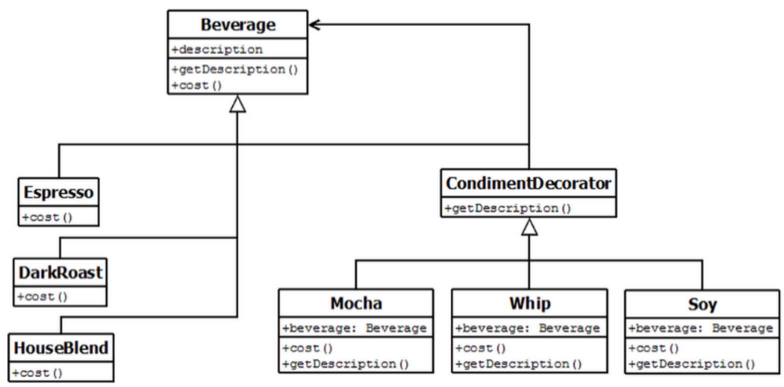


Figure 21: UML for Implemented Decorator Design Pattern [39]

Lastly creational design patterns create objects, as opposed to the developer directly creating them. For this experiment, a factory design pattern is used as the creational block. The generic UML diagram for this design pattern is shown in Figure 22 and the UML for the implemented pattern used in this experiment is given in Figure 23.

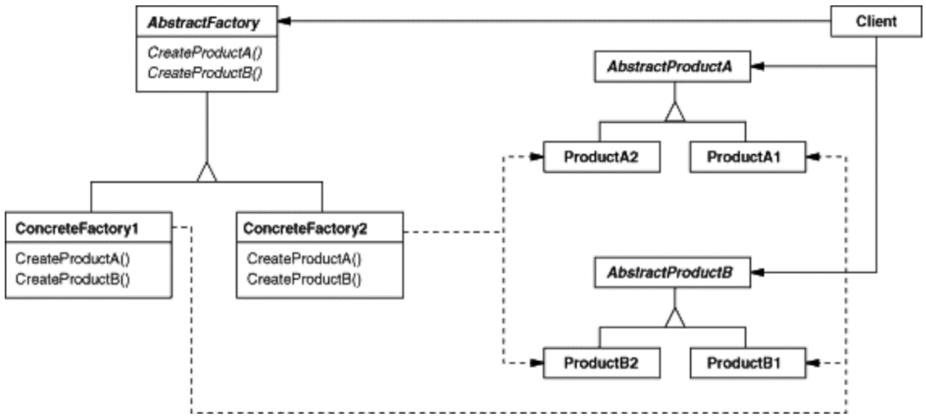


Figure 22: Generic Factory Pattern UML as defined in Design Patterns: Elements of Reusable Object-Oriented Software [38]

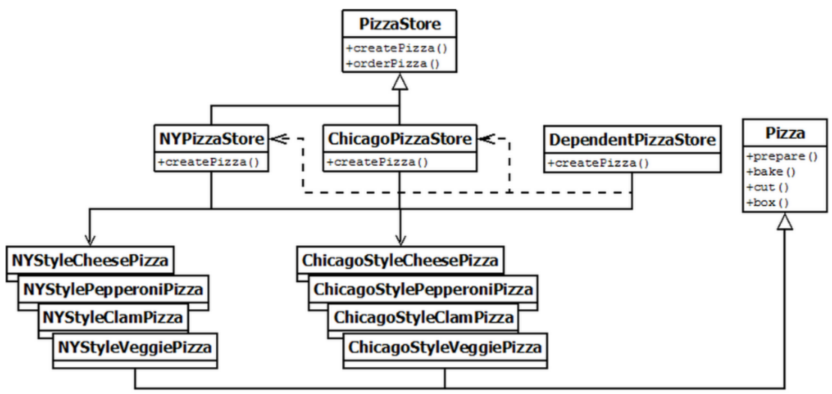


Figure 23: UML Diagram of Factory Pattern [39]

Experimental Design

We use a Randomize Complete Block Design (RCBD) because we would like to control the variability that comes from the different design patterns. The six modular grime types are the treatments for this design, the design pattern categories are the blocks, and the technical debt scores are reported by SonarQube are the response variables. For each block and treatment, 5 scores are generated. Table 4 displays the experimental design.

Table 4: Experiment Treatments and Blocks

Design	RCBD
Independent Variables	Grime Types, Number of Grime instances
Dependent Variables	Technical debt scores
Treatments	PEAG, PEEG, PIG, TEAG, TEEG, TIG
Blocks	Behavioral DP, Creational DP, Structural DP
Alpha Level	0.05
Replications	5

RESULTS AND ANALYSIS

To conduct this analysis, we repeated this experiment three times, one time with 10 instances, one time with 50 instances, and one with 100 instances of each modular grime type. The following sections provide the results and analysis for each experiment.

Assumptions

Before analysis, there are a few assumptions that should be verified so that we can apply parametric statistics. These assumptions include the assumption of a normal distribution and the homogeneity of variance assumptions.

Assumption of Normality

We assume that errors are normally distributed when conducting statistical analysis. To verify this assumption we can inspect the normality plots and a histogram of the residuals. Residuals are the difference between the observed value and the associated predicted value [40]. It tells us how far off the model's prediction is at that point. The pattern in normality plot should be close to linear when the residuals are approximately normally distributed while the histogram should be bell-shaped.

Figure 24 displays the graphs described above for the 3 experimental runs. From left to right it displays the 10 instances, 50 instances, and 100 instances injected. We can see that the plots are reasonably linear and the histograms are approximately bell-shaped, so the assumption of normality appears to hold.

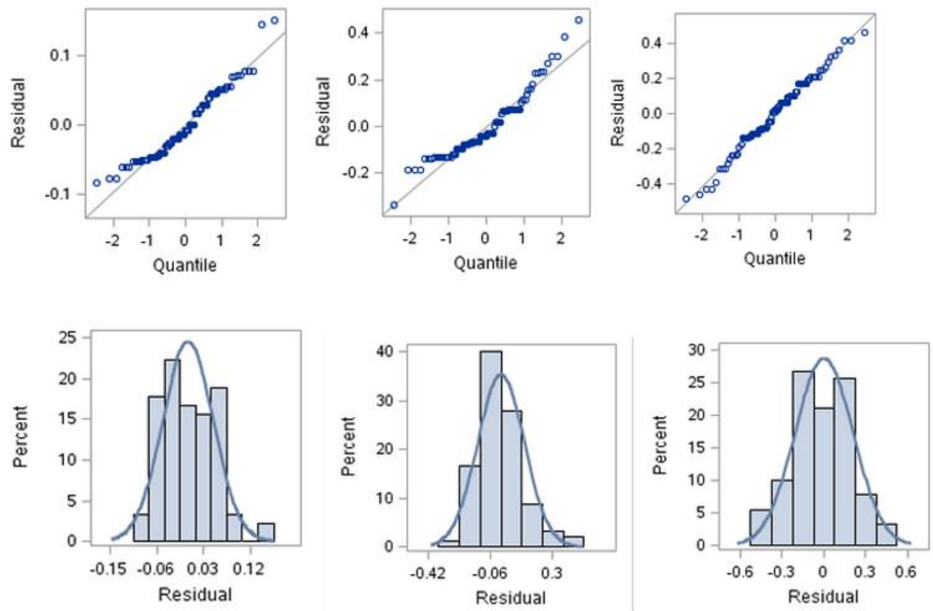


Figure 24: Normality Assumption Analysis Graphs

Assumption of Homogeneity of Variance

The homogeneity of variance (HOV) assumption states that residuals should have the same variance for each treatment. If this assumption is met, the residuals should be centered about 0 and the spread of the residuals should be similar for each treatment.

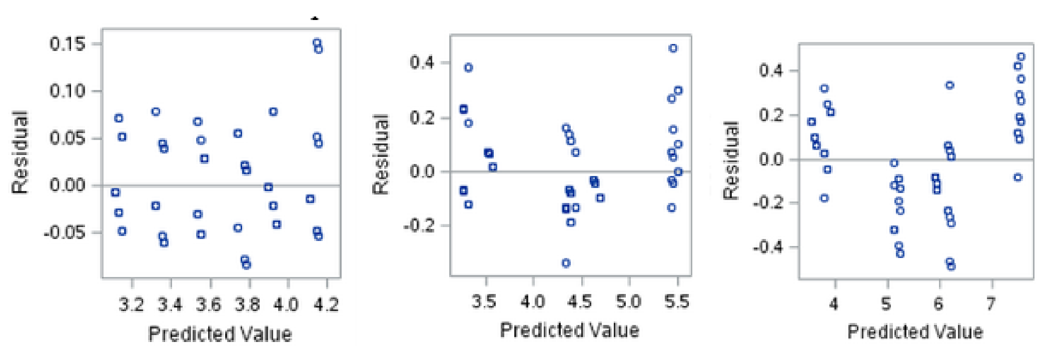


Figure 25: HOV Assumption Analysis Graphs

Figure 25 displays the graphs used to analyze the HOV assumption as described above for the 3 experimental runs. From left to right it displays the 10 instances, 50 instances, and 100 instances injected. We can see that the plots for 10 instances and 50 instances appear to be centered about 0 and the spread of the residuals should be similar for each treatment, so we can reasonably say that there has been no serious violations in the HOV assumption for the 10, 50, and 100 instances of grime.

Results

Once verified, we can run our statistical tests on the measurements collected for our randomized complete block design tests. The technical debt measurements reported by SonarQube can be found in Appendix B.

We use standard ANOVA tests to analyze variations. ANOVA (Analysis of Variance) tests are used to analyze treatment effects between treatments. We can see that for all cases, 10, 50, and 100 instances of modular grime there is sufficient evidence to reject the null hypothesis that all types of grime have the same effect on technical debt. Both cases have a p-value of <0.001, less than an alpha value of 0.05. In other words, there is less than a .01% chance we observed these results purely by chance. Figure 26, Figure 27, and Figure 28 show the associated ANOVAS for 10, 50, and 100 instances of grime respectively.

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	10.54777778	1.50682540	583.44	<.0001
Error	82	0.21177778	0.00258266		
Corrected Total	89	10.75955556			

Figure 26: ANOVA for 10 instances of grime.

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	46.14333333	6.59190476	329.46	<.0001
Error	82	1.64066667	0.02000813		
Corrected Total	89	47.78400000			

Figure 27: ANOVA for 50 instances of grime.

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	165.7464444	23.6780635	503.03	<.0001
Error	82	3.8597778	0.0470705		
Corrected Total	89	169.6062222			

Figure 28 ANOVA for 100 instances of grime.

Now that we have rejected the null hypothesis that all treatment effects are equal using ANOVA tests, we can perform a Tukey’s test to test all pairwise mean comparisons to see which treatment effects are statistically different from each other. SAS Software organizes these pairwise comparisons into groups that are statistically different from each other (Figure 29 displays the results of the Tukey’s Test. From left to right, the tables are for the 10 instances, 50 instances, and 100 instances of modeled modular grime growth). We find all three types of persistent grime (PEAG, PEEG, PIG) showed significantly higher technical debt scores than all three types of temporary grime (TEAG, TEEG, TIG).

Means with the same letter are not significantly different.			
Tukey Grouping	Mean	N	GrimeType
A	3.76667	15	TIG
A			
A	3.76000	15	TEEG
A			
A	3.72667	15	TEAG
B	3.55333	15	PEAG
B			
B	3.53333	15	PEEG
B			
B	3.51333	15	PIG

Means with the same letter are not significantly different.			
Tukey Grouping	Mean	N	GrimeType
A	4.87333	15	TEEG
A			
A	4.82000	15	TIG
A			
A	4.80667	15	TEAG
B	3.76000	15	PEAG
B			
B	3.71333	15	PIG
B			
B	3.70667	15	PEEG

Means with the same letter are not significantly different.			
Tukey Grouping	Mean	N	GrimeType
A	6.55333	15	TIG
A			
A	6.52667	15	TEAG
A			
A	6.50000	15	TEEG
B	4.25333	15	PIG
B			
B	4.21333	15	PEAG
B			
B	4.14000	15	PEEG

Figure 29: Tukey Grouping Test Results
The full statistical results given by SAS [41] can be viewed in Appendix C.

DISCUSSION

Our findings suggests that temporary coupling results in a higher technical debt score than persistent coupling as reported by SonarQube. Further research can help provide insight into the rates at which technical debt scores increase for the different modular grime types, as well as metrics that identify grime as it occurs. These metrics would allow for an automated means of identifying grime as it occurs and give practitioners with a tool that provides the current state of technical debt of their system in relation to modular grime.

The results obtained here were obtained from the SQALE Method for technical debt. This methodology does not include calculations to incorporate interest, such as Nugroho's proposed methodology. This suggests that perhaps further investigation is warranted to explore the possibility of more sophisticated and complete means of evaluating the true cost of the technical debt incurred. Investigation into the maintenance cost associated with modular grime could provide a starting point to incorporate the interest accrued on the principal of technical debt incurred by modular grime.

The results of their pilot study, Schanz and Izurieta [30] found that TEEG, TEAG, and PEAG tended to increase, while PEAG did not. This finding, if found to be true for most systems, is concerning because it will result in a larger increase in the technical debt score reported by SonarQube. Further research to understand the rates at which modular grime occurs in practice will allow us to understand the current state of grime and technical debt in the industry.

The findings of this study may be incorporated into technical debt management plans. With the understanding that temporary grime types results in higher technical debt scores than persistent grime types, care can be taken to avoid temporary grime types and keep track of temporary grime types if it is impossible to avoid, so that it may be managed with other known technical debt items.

THREATS TO VALIDITY

Construct Validity

Construct validity concerns the validity of measurements and observations collected on the construct being investigated. Feldt and Magazinus [42] summarize construct validity using the following questions: Does the treatment correspond to the actual cause we are interested in? Does the outcome correspond to the effect we are interested in?

As discussed in the Background section, there is no agreed upon method for measuring technical debt. Because there is no benchmark, the response variable (technical debt) being reported by SonarQube is potentially a threat to the construct validity of this research. SonarQube's ability to accurately measure technical debt may not accurately reflect the technical debt of a system.

The injector tool we created for this research has not yet been evaluated to assess if it accurately represents grime growth. A possible inconsistency is the potential to inject false-positive grime. Because the injector works by selecting two random classes, there is no assurance that the coupling of these two classes violate the design pattern's intended purpose and they may not in actuality be considered grime.

Another possible threat to the construct validity are the experimental units we've chosen. They are simple programs used demonstrate a design pattern's use, but may not accurately represent design patterns used in practice.

Internal Validity

Internal validity refers to the extent that results are attributable to the independent variable and not some other factor. Feldt and Magazinus [42] summarize internal validity using the following questions: Did the treatment/change we introduced cause the effect on the outcome? Can other factors also have had an effect?

Javassist allows us to model grime growth by manipulating Java bytecode, but going through this process manipulates the code in ways that potentially poses threats to the internal validity. Elements of the original code, such as comments, are lost during the compilation process. When the modified bytecode is decompiled to perform analysis, JAD inserts its own comments to the decompiled code. When calculating the technical debt scores, a portion of the score is calculated by the ratio of comments to code. Because comments have been stripped away and then added again, it is possible the ratio of comments to code in the decompiled code does not accurately represent the comment to code ratio of the original, unmodified code. When analyzing differences between the grime types, the risk is minimized by the fact that it will be equally skewed between each modeled grime type. If attempting to perform analysis between original code and modeled code, this factor needs to be taken into account.

External Validity

External validity is the degree to which the results of an experiment can be generalized. Feldt and Magazinus [42] summarize external validity using the following questions: Is the cause and effect relationship we have shown valid in other situations? Can we generalize our results? Do the results apply in other contexts?

The research conducted used solely Java projects, therefore any findings can only be generalized to Java projects. Further research will be needed to be able to generalize findings to a larger code population.

We have only measured technical debt using SonarQube. This is a threat to the external validity as we cannot speak to how other means of calculating technical debt might compare.

Another threat to the external validity is that we have only used one representative pattern for the design pattern categories construction, behavioral, and structural.

CONCLUSION

Understanding the role modular grime plays in the technical debt field will help lead to better understanding of the financial cost associated with grime and technical debt management. Knowing the effects of different types of grime will allow software engineers to make design decisions that result in lower technical debt and a more comprehensive technical debt framework.

In this paper, we have discussed current techniques for identifying and managing technical debt in the Background section. Grime has been identified as a design debt that has negative impacts on the quality of a project. Our research is the first step to quantifying those negative consequences in terms of technical debt.

For our research, we used SonarQube to calculate a technical debt score for Java projects modified by our grime injector to represent modular grime growth. We then performed an ANOVA analysis on the collected technical debt scores to find that not all types of modular grime results in equivalent treatment effects on technical debt. Furthermore, Tukey's test shows that that every type of temporary grime (TEAG, TEEG, TIG) is statistically significantly higher technical debt score (as reported by SonarQube) than every type of persistent grime (PEAG, PEEG, PIG).

Previous work has shown grime to correspond to negative software quality in regards to testability and should be monitored as systems development to ensure higher quality system. Our research provides further support for reasons to care not only about grime, but also the type of grime. Knowing temporary grime types can be more costly than persistent grime types, engineers can make better informed design decisions or repayment decisions that will result in lower technical debt.

The injector tool created for this research also has the potential to expand findings to include research into metrics which may correspond to grime growth, which could alert engineers when grime occurs so that they may add it to their list of known technical debt items to manage.

Quantifying grime in terms of technical debt is the first step to including grime in a technical debt management plan. The findings of this research form a foundation to continue exploring the relationship between design pattern grime and technical debt. Further research will explore ways to provide a more holistic view of grime and technical debt.

FUTURE RESEARCH

While the research presented here are the first steps towards understanding the role design pattern grime plays on technical debt, there is still more investigation to be conducted. The following few paragraphs describe possible areas of future research.

These experiments have only investigated Java programs. Expanding this research to include different programming languages would provide a more complete picture of the relationship between technical debt and grime.

As discussed in the background section, there are two other forms of design grime: organizational grime and class grime. This research could be expanded to include investigations to the relationships of these other types of grime to technical debt and to each other.

Here we investigated differences between the different types of modular grime on technical debt. Some of our findings here suggest that temporary grime types are not only more costly in technical debt scores reported by SonarQube, but also accrues debt at a quicker rate. Further investigation can be conducted to understand the rates at which technical debt grows as grime instances increase.

Lastly, SonarQube is only one tool that evaluates technical debt. Investigating modified programs with other tools will expand our understanding of the true role grime growth plays in technical debt.

REFERENCES CITED

[1] C. Izurieta and J. M. Bieman, "How Software Designs Decay: A Pilot Study of Pattern Evolution," in *ESEM*, Madrid, Spain, 2007.

[2] W. Cunningham, "The WyCash portfolio management system," *OOPSLA '92 Experience Report,* vol. 4, no. 2, pp. 29-30, 1992.

[3] "SonarQube 3.7.4," 20 December 2013. [Online]. Available: http://www.sonarqube.org/downloads/. [Accessed 13 January 2014].

[4] J. Kerievsky, Refactoring to patterns, Pearson Deutschland GmbH, 2005.

[5] C. J. Neill and P. A. Laplante, "Paying down design debt with strategic refactoring," *Trans. Software Eng,* vol. 27, no. 1, 2006.

[6] P. Kruchten, R. L. Nord and a. I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice," *IEEE Software*, vol. 6, no. 29, 2012.

[7] "An Enterprise Perspective on Technical Debt," *Proceedings of the 2nd Workshop on Managing Technical Debt,* pp. 35-38, 2011.

[8] M. Fowler, "TechnicalDebtQuadrant," 14 October 2009. [Online]. Available: http://www.martinfowler.com/bliki/TechnicalDebtQuadrant.html. [Accessed 25 March 2014].

[9] N. M. A. S. F. S. a. C. S. Zazworka, "Investigating the impact of design debt on software quality," *Proceedings of the 2nd Workshop on Managing Technical Debt,* Vols. 17-23, 2011.

[10] N. Zazworka, A. Vetro, C. Izurieta, S. Wong, Y. Cai, C. Seaman and F. Shull, "Comparing Four Approaches for Technical Debt Identification," *Software Quality Journal*, pp. 1-24, 2013.

[11] B. Curtis, J. Sappidi and A. Szynkarski, "Estimating the Size, Cost, and Types of Technical Debt," *Managing Technical Debt (MTD), 2012 Third International Workshop on.,* pp. 49-53, 2012.

[12] N. Ariadi, V. Joost and K. Tobias, "An Empirical Model of Technical Debt and Interest," *Proceedings of the 2nd Workshop on Managing Technical Debt,* pp. 1-8, 2011.

[13] J. de Groot, A. Nugroho, T. Back and J. Visser, "What is the value of your software?," *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pp. 37-44, 2012.

[14] J.-L. Letouzey, "The SQALE Method for Evaluating Technical Debt," in *I C S E*, Z u r i c h , S w i t z e r l a n d , 2012.

[15] I. Gat, "Revolution in Software: Using Technical Debt Techniques to Govern the Software Development Process," *Agile Product and Project Management, Cutter Consortium Executive Report*, vol. 11, no. 4, 2010.

[16] O. Gaudin, "Quality Profiles," SonarQube, 05 Dec 2013. [Online]. Available: http://docs.codehaus.org/display/SONAR/Quality+Profiles. [Accessed 21 February 2014].

[17] D. Racodon, "Technical Debt," SonarQube, 03 December 2013. [Online]. Available: http://docs.codehaus.org/display/SONAR/Technical+Debt. [Accessed 08 March 2014].

[18] N. Brown, C. Yuanfang, G. Yuepu, K. Rick, K. Miryung, K. Philippe, L. Erin and e. al, "Managing Technical Debt in Software-Reliant Systems," *Proceedings of the FSE/SDP workshop on Future of software engineering research*, pp. 47-52, 2010.

[19] N. Zazworka, S. Carolyn and S. Forrest, "Prioritizing design debt investment opportunities," *Proceedings of the 2nd Workshop on Managing Technical Debt*, pp. 39-42, 2011.

[20] N. Zazworka, M. A. Shaw, F. Shull and Seaman.Carolyn, "Investigating the impact of design debt on software quality," *Proceedings of the 2nd Workshop on Managing Technical Debt*, pp. 17-23, 2011.

[21] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull and A. Vetrò, "Using technical debt data in decision making: Potential decision approaches," *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pp. 45-48, 2012.

[22] W. Snipes, B. Robinson, Y. Guo and C. Seaman, "Defining the Decision Factors for Managing Defects: A Technical Debt Perspective," *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pp. 54-60, 2012.

[23] N. A. Ernst, "On the role of requirements in understanding and managing technical debt," *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pp. 61-64, 2012.

[24] E. Lim, N. Taksande and C. Seaman, "A balancing act: what software practitioners have to say about technical debt," *Software, IEEE*, vol. 6, no. 29, pp. 22-27, 2012.

[25] J. D. Morgenthaler, M. Gridnev, R. Sauciuc and S. Bhansali, "Searching for build debt: Experiences managing technical debt at google," *Managing Technical Debt (MTD), 2012 Third International Workshop on,* pp. 1-6, 2012.

[26] CAST , "The CRASH Report - 2011/12 (CAST Report on Application Software Health)," 2012.

[27] CAST, "Measuring and Managing Technical Debt," www.castsoftware.com.

[28] E. Tom, A. Aurum and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software* , vol. 6, no. 86, pp. 1498-1516, 2013.

[29] C. Izurieta and J. M. Bieman, "A multiple case study of design pattern decay, grime, and rot in evolving software systems," *Software Quality Journal,* vol. 21, no. 2, pp. 289-323, 2013.

[30] T. Schanz and C. Izurieta, "Object Oriented Design Pattern Decay: A Taxonomy," in *ESEM2010*, Bolzano-Bozen, Italy , 2010.

[31] J. Vlissides, R. Helm, R. Johnson and E. Gamma, "Design Patterns: Elements of Reusable Object-Oriented Software," 1995.

[32] C. Izurieta, A. Vetró, N. Zazworka, Y. Cai, C. Seaman and F. Shull, "Organizing the Technical Debt Landscape," *Managing Technical Debt (MTD) 2012 Third International Workshop,* pp. 23-36, 2012.

[33] C. Izurieta and a. J. M. Bieman, "Testing consequences of grime buildup in object oriented design patterns," *Software Testing, Verification, and Validation, 2008 1st International Conference on.*, 2008.

[34] S. Chiba, "Javassist.org," JBoss, 3 June 2013. [Online]. Available: http://www.csg.ci.i.u-tokyo.ac.jp/~chiba/javassist/. [Accessed 18 FEB 2014].

[35] P. Kouznetsov, "Jad - the fast JAva Decompiler," February 2013. [Online]. Available: http://varaneckas.com/jad/. [Accessed 15 April 2013].

[36] "Java Platform Standard Edition 7," ORACLE.

[37] "Eclipse Juno 4.2," http://help.eclipse.org/juno/index.jsp.

[38] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns: Elements of reusable object-oriented software, Addison-Wesley Professional, 1994.

[39] E. Freeman and E. Freeman, Head First Design Patterns, Sebastopol: O'Reilly, 2004.

[40] R. De Veaux, P. Velleman and D. Bock, Stats Data and Models, Pearson Education, Inc., 2008.

[41] "SAS Web Editor 2.5," SAS Institute Inc., C, Cary, NC, USA, 2013.

[42] R. Feldt and A. Magazinius, "Validity Threats in Empirical Software Engineering Research-An Initial Survey," *SEKE,* pp. 374-379, 2010.

APPENDICES

APPENDIX A

JAVA CHECKS PERFORMED BY DEFAULT QUALITY PROFILE IN SONARQUBE

title	Key	plugin	priority	status
Abstract Class Without Abstract Method	AbstractClassWithoutAbstractMethod	pmd	MAJOR	ACTIVE
Abstract class without any methods	AbstractClassWithoutAnyMethod	pmd	MAJOR	ACTIVE
Abstract naming	AbstractNaming	pmd	MAJOR	ACTIVE
Accessor Class Generation	AccessorClassGeneration	pmd	MAJOR	ACTIVE
Add Empty String	AddEmptyString	pmd	MAJOR	ACTIVE
Anon Inner Length	com.puppycrawl.tools.checkstyle.checks.sizes.AnonInnerLengthCheck	checkstyle	MAJOR	ACTIVE
Append Character With Char	AppendCharacterWithChar	pmd	MINOR	ACTIVE
Assignment In Operand	AssignmentInOperand	pmd	MAJOR	ACTIVE
Assignment To Non Final Static	AssignmentToNonFinalStatic	pmd	MAJOR	ACTIVE
At Least One Constructor	AtLeastOneConstructor	pmd	MAJOR	ACTIVE
Avoid Accessibility Alteration	AvoidAccessibilityAlteration	pmd	MAJOR	ACTIVE
Avoid Array Loops	AvoidArrayLoops	pmd	MAJOR	ACTIVE
Avoid Assert As Identifier	AvoidAssertAsIdentifier	pmd	MAJOR	ACTIVE
Avoid Calling Finalize	AvoidCallingFinalize	pmd	MAJOR	ACTIVE
Avoid Catching NPE	AvoidCatchingNPE	pmd	MAJOR	ACTIVE
Avoid Catching Throwable	AvoidCatchingThrowable	pmd	CRITICAL	ACTIVE
Avoid commented-out lines of code	CommentedOutCodeLine	squid	MAJOR	ACTIVE
Avoid Constants Interface	AvoidConstantsInterface	pmd	MAJOR	ACTIVE
Avoid Decimal Literals In Big Decimal Constructor	AvoidDecimalLiteralsInBigDecimalConstructor	pmd	MAJOR	ACTIVE
Avoid Deeply Nested If Stmts	AvoidDeeplyNestedIfStmts	pmd	MAJOR	ACTIVE
Avoid Duplicate Literals	AvoidDuplicateLiterals	pmd	MAJOR	ACTIVE
Avoid Enum As Identifier	AvoidEnumAsIdentifier	pmd	MAJOR	ACTIVE
Avoid Final Local Variable	AvoidFinalLocalVariable	pmd	MAJOR	ACTIVE
Avoid Instanceof Checks In Catch Clause	AvoidInstanceofChecksInCatchClause	pmd	MINOR	ACTIVE
Avoid instantiating objects in loops	AvoidInstantiatingObjectsInLoops	pmd	MINOR	ACTIVE
Avoid Multiple Unary Operators	AvoidMultipleUnaryOperators	pmd	MAJOR	ACTIVE
Avoid Print Stack Trace	AvoidPrintStackTrace	pmd	MAJOR	ACTIVE
Avoid Protected Field In Final Class	AvoidProtectedFieldInFinalClass	pmd	MAJOR	ACTIVE
Avoid Reassigning Parameters	AvoidReassigningParameters	pmd	MAJOR	ACTIVE

title	Key	plugin	priority	status
Avoid Rethrowing Exception	AvoidRethrowingException	pmd	MAJOR	ACTIVE
Avoid StringBuffer field	AvoidStringBufferField	pmd	MAJOR	ACTIVE
Avoid Synchronized At Method Level	AvoidSynchronizedAtMethodLevel	pmd	MAJOR	ACTIVE
Avoid Thread Group	AvoidThreadGroup	pmd	CRITICAL	ACTIVE
Avoid Throwing Null Pointer Exception	AvoidThrowingNullPointerException	pmd	MAJOR	ACTIVE
Avoid Throwing Raw Exception Types	AvoidThrowingRawExceptionTypes	pmd	MAJOR	ACTIVE
Avoid use of deprecated method	CallToDeprecatedMethod	squid	MINOR	ACTIVE
Avoid Using Hard Coded IP	AvoidUsingHardCodedIP	pmd	MAJOR	ACTIVE
Avoid Using Native Code	AvoidUsingNativeCode	pmd	MAJOR	ACTIVE
Avoid Using Octal Values	AvoidUsingOctalValues	pmd	MAJOR	ACTIVE
Avoid Using Short Type	AvoidUsingShortType	pmd	MAJOR	ACTIVE
Avoid Using Volatile	AvoidUsingVolatile	pmd	MAJOR	ACTIVE
Bad Comparison	BadComparison	pmd	MAJOR	ACTIVE
Bad practice - Abstract class defines covariant compareTo() method	CO_ABSTRACT_SELF	findbugs	MAJOR	ACTIVE
Bad practice - Abstract class defines covariant equals() method	EQ_ABSTRACT_SELF	findbugs	MAJOR	ACTIVE
Bad practice - Certain swing methods needs to be invoked in Swing thread	SW_SWING_METHODS_INVOKED_IN_SWING_THREAD	findbugs	MAJOR	ACTIVE
Bad practice - Check for sign of bitwise operation	BIT_SIGNED_CHECK	findbugs	CRITICAL	ACTIVE
Bad practice - Class defines clone() but doesn't implement Cloneable	CN_IMPLEMENTS_CLONE_BUT_NOT_CLONEABLE	findbugs	MAJOR	ACTIVE
Bad practice - Class defines compareTo(...) and uses Object.equals()	EQ_COMPARETO_USE_OBJECT_EQUALS	findbugs	CRITICAL	ACTIVE
Bad practice - Class defines equals() and uses Object.hashCode()	HE_EQUALS_USE_HASHCODE	findbugs	CRITICAL	ACTIVE
Bad practice - Class defines equals() but not hashCode()	HE_EQUALS_NO_HASHCODE	findbugs	MAJOR	ACTIVE
Bad practice - Class defines hashCode() and uses Object.equals()	HE_HASHCODE_USE_OBJECT_EQUALS	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Bad practice - Class defines hashCode() but not equals()	HE_HASHCODE_NO_EQUALS	findbugs	CRITICAL	ACTIVE
Bad practice - Class implements Cloneable but does not define or use clone method	CN_IDIOM	findbugs	MAJOR	ACTIVE
Bad practice - Class inherits equals() and uses Object.hashCode()	HE_INHERITS_EQUALS_USE_HASHCODE	findbugs	CRITICAL	ACTIVE
Bad practice - Class is Externalizable but doesn't define a void constructor	SE_NO_SUITABLE_CONSTRUCTOR_FOR_EXTERNALIZATION	findbugs	MAJOR	ACTIVE
Bad practice - Class is not derived from an Exception, even though it is named as such	NM_CLASS_NOT_EXCEPTION	findbugs	MAJOR	ACTIVE
Bad practice - Class is Serializable but its superclass doesn't define a void constructor	SE_NO_SUITABLE_CONSTRUCTOR	findbugs	MAJOR	ACTIVE
Bad practice - Class names shouldn't shadow simple name of implemented interface	NM_SAME_SIMPLE_NAME_AS_INTERFACE	findbugs	MAJOR	ACTIVE
Bad practice - Class names shouldn't shadow simple name of superclass	NM_SAME_SIMPLE_NAME_AS_SUPERCLASS	findbugs	MAJOR	ACTIVE
Bad practice - Classloaders should only be created inside doPrivileged block	DP_CREATE_CLASSLOADER_INSIDE_DO_PRIVILEGED	findbugs	MAJOR	ACTIVE
Bad practice - clone method does not call super.clone()	CN_IDIOM_NO_SUPER_CALL	findbugs	MAJOR	ACTIVE
Bad practice - Clone method may return null	NP_CLONE_COULD_RETURN_NULL	findbugs	CRITICAL	ACTIVE
Bad practice - Comparator doesn't implement Serializable	SE_COMPARATOR_SHOULD_BE_SERIALIZABLE	findbugs	MAJOR	ACTIVE
Bad practice - Comparison of String objects using == or !=	ES_COMPARING_STRINGS_WITH_EQ	findbugs	MAJOR	ACTIVE
Bad practice - Comparison of String parameter using == or !=	ES_COMPARING_PARAMETER_STRING_WITH_EQ	findbugs	MAJOR	ACTIVE
Bad practice - Confusing	NM_CONFUSING	findbugs	MAJOR	ACTIVE

title	Key	plugin	priority	status
method names				
Bad practice - Covariant compareTo() method defined	CO_SELF_NO_OBJECT	findbugs	MAJOR	ACTIVE
Bad practice - Covariant equals() method defined	EQ_SELF_NO_OBJECT	findbugs	MAJOR	ACTIVE
Bad practice - Creates an empty jar file entry	AM_CREATES_EMPTY_JAR_FIL E_ENTRY	findbugs	MAJOR	ACTIVE
Bad practice - Creates an empty zip file entry	AM_CREATES_EMPTY_ZIP_FIL E_ENTRY	findbugs	MAJOR	ACTIVE
Bad practice - Dubious catching of IllegalMonitorStateException	IMSE_DONT_CATCH_IMSE	findbugs	MAJOR	ACTIVE
Bad practice - Empty finalizer should be deleted	FI_EMPTY	findbugs	MAJOR	ACTIVE
Bad practice - Equals checks for noncompatible operand	EQ_CHECK_FOR_OPERAND_N OT_COMPATIBLE_WITH_THIS	findbugs	MAJOR	ACTIVE
Bad practice - equals method fails for subtypes	EQ_GETCLASS_AND_CLASS_C ONSTANT	findbugs	CRITICAL	ACTIVE
Bad practice - Equals method should not assume anything about the type of its argument	BC_EQUALS_METHOD_SHOUL D_WORK_FOR_ALL_OBJECTS	findbugs	CRITICAL	ACTIVE
Bad practice - equals() method does not check for null argument	NP_EQUALS_SHOULD_HANDL E_NULL_ARGUMENT	findbugs	CRITICAL	ACTIVE
Bad practice - Explicit invocation of finalizer	FI_EXPLICIT_INVOCATION	findbugs	MAJOR	ACTIVE
Bad practice - Fields of immutable classes should be final	JCIP_FIELD_ISNT_FINAL_IN_I MMUTABLE_CLASS	findbugs	MINOR	ACTIVE
Bad practice - Finalizer does not call superclass finalizer	FI_MISSING_SUPER_CALL	findbugs	MAJOR	ACTIVE
Bad practice - Finalizer does nothing but call superclass finalizer	FI_USELESS	findbugs	MINOR	ACTIVE
Bad practice - Finalizer nullifies superclass finalizer	FI_NULLIFY_SUPER	findbugs	CRITICAL	ACTIVE
Bad practice - Finalizer nulls fields	FI_FINALIZER_NULLS_FIELDS	findbugs	MAJOR	ACTIVE
Bad practice - Finalizer only nulls fields	FI_FINALIZER_ONLY_NULLS_FI ELDS	findbugs	MAJOR	ACTIVE

title	Key	plugin	priority	status
Bad practice - Iterator next() method can't throw NoSuchElementException	IT_NO_SUCH_ELEMENT	findbugs	MINOR	ACTIVE
Bad practice - Method doesn't override method in superclass due to wrong package for parameter	NM_WRONG_PACKAGE_INTE NTIONAL	findbugs	MAJOR	ACTIVE
Bad practice - Method ignores exceptional return value	RV_RETURN_VALUE_IGNORED _BAD_PRACTICE	findbugs	MAJOR	ACTIVE
Bad practice - Method ignores results of InputStream.read()	RR_NOT_CHECKED	findbugs	MAJOR	ACTIVE
Bad practice - Method ignores results of InputStream.skip()	SR_NOT_CHECKED	findbugs	MAJOR	ACTIVE
Bad practice - Method invoked that should be only be invoked inside a doPrivileged block	DP_DO_INSIDE_DO_PRIVILEG ED	findbugs	MAJOR	ACTIVE
Bad practice - Method invokes dangerous method runFinalizersOnExit	DM_RUN_FINALIZERS_ON_EXI T	findbugs	MAJOR	ACTIVE
Bad practice - Method invokes System.exit(...)	DM_EXIT	findbugs	MAJOR	ACTIVE
Bad practice - Method may fail to close database resource	ODR_OPEN_DATABASE_RESO URCE	findbugs	CRITICAL	ACTIVE
Bad practice - Method may fail to close database resource on exception	ODR_OPEN_DATABASE_RESO URCE_EXCEPTION_PATH	findbugs	CRITICAL	ACTIVE
Bad practice - Method may fail to close stream	OS_OPEN_STREAM	findbugs	CRITICAL	ACTIVE
Bad practice - Method may fail to close stream on exception	OS_OPEN_STREAM_EXCEPTIO N_PATH	findbugs	CRITICAL	ACTIVE
Bad practice - Method might drop exception	DE_MIGHT_DROP	findbugs	MAJOR	ACTIVE
Bad practice - Method might ignore exception	DE_MIGHT_IGNORE	findbugs	MAJOR	ACTIVE
Bad practice - Method with Boolean return type returns explicit null	NP_BOOLEAN_RETURN_NULL	findbugs	MAJOR	ACTIVE

title	Key	plugin	priority	status
Bad practice - Needless instantiation of class that only supplies static methods	ISC_INSTANTIATE_STATIC_CLASS	findbugs	MAJOR	ACTIVE
Bad practice - Non-serializable class has a serializable inner class	SE_BAD_FIELD_INNER_CLASS	findbugs	MINOR	ACTIVE
Bad practice - Non-serializable value stored into instance field of a serializable class	SE_BAD_FIELD_STORE	findbugs	CRITICAL	ACTIVE
Bad practice - Random object created and used only once	DMI_RANDOM_USED_ONLY_ONCE	findbugs	CRITICAL	ACTIVE
Bad practice - Serializable inner class	SE_INNER_CLASS	findbugs	MAJOR	ACTIVE
Bad practice - serialVersionUID isn't final	SE_NONFINAL_SERIALVERSIONID	findbugs	CRITICAL	ACTIVE
Bad practice - serialVersionUID isn't long	SE_NONLONG_SERIALVERSIONID	findbugs	MAJOR	ACTIVE
Bad practice - serialVersionUID isn't static	SE_NONSTATIC_SERIALVERSIONID	findbugs	MAJOR	ACTIVE
Bad practice - Static initializer creates instance before all static final fields assigned	SI_INSTANCE_BEFORE_FINALS_ASSIGNED	findbugs	CRITICAL	ACTIVE
Bad practice - Store of non serializable object into HttpSession	J2EE_STORE_OF_NON_SERIALIZABLE_OBJECT_INTO_SESSION	findbugs	CRITICAL	ACTIVE
Bad practice - Superclass uses subclass during initialization	IC_SUPERCLASS_USES_SUBCLASS_DURING_INITIALIZATION	findbugs	MAJOR	ACTIVE
Bad practice - Suspicious reference comparison	RC_REF_COMPARISON	findbugs	CRITICAL	ACTIVE
Bad practice - The readResolve method must be declared with a return type of Object.	SE_READ_RESOLVE_MUST_RETURN_OBJECT	findbugs	MAJOR	ACTIVE
Bad practice - toString method may return null	NP_TOSTRING_COULD_RETURN_NULL	findbugs	CRITICAL	ACTIVE
Bad practice - Transient field that isn't set by deserialization.	SE_TRANSIENT_FIELD_NOT_RESTORED	findbugs	MAJOR	ACTIVE
Bad practice - Unchecked type in generic call	GC_UNCHECKED_TYPE_IN_GENERIC_CALL	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Bad practice - Usage of GetResource may be unsafe if class is extended	UI_INHERITANCE_UNSAFE_GETRESOURCE	findbugs	MAJOR	ACTIVE
Bad practice - Use of identifier that is a keyword in later versions of Java	NM_FUTURE_KEYWORD_USED_AS_IDENTIFIER	findbugs	MAJOR	ACTIVE
Bad practice - Use of identifier that is a keyword in later versions of Java	NM_FUTURE_KEYWORD_USED_AS_MEMBER_IDENTIFIER	findbugs	MAJOR	ACTIVE
Bad practice - Very confusing method names (but perhaps intentional)	NM_VERY_CONFUSING_INTENTIONAL	findbugs	MAJOR	ACTIVE
Big Integer Instantiation	BigIntegerInstantiation	pmd	MAJOR	ACTIVE
Boolean Expression Complexity	com.puppycrawl.tools.checkstyle.checks.metrics.BooleanExpressionComplexityCheck	checkstyle	MAJOR	ACTIVE
Boolean Get Method Name	BooleanGetMethodName	pmd	MAJOR	ACTIVE
Boolean Instantiation	BooleanInstantiation	pmd	MAJOR	ACTIVE
Boolean Inversion	BooleanInversion	pmd	MAJOR	ACTIVE
Broken Null Check	BrokenNullCheck	pmd	CRITICAL	ACTIVE
Call Super In Constructor	CallSuperInConstructor	pmd	MINOR	ACTIVE
Check ResultSet	CheckResultSet	pmd	MAJOR	ACTIVE
Class Cast Exception With To Array	ClassCastExceptionWithToArray	pmd	MAJOR	ACTIVE
Clone method must implement Cloneable	CloneMethodMustImplementCloneable	pmd	MAJOR	ACTIVE
Clone Throws Clone Not Supported Exception	CloneThrowsCloneNotSupportedException	pmd	MAJOR	ACTIVE
Collapsible If Statements	CollapsibleIfStatements	pmd	MINOR	ACTIVE
Compare Objects With Equals	CompareObjectsWithEquals	pmd	MAJOR	ACTIVE
Confusing Ternary	ConfusingTernary	pmd	MAJOR	ACTIVE
Consecutive Literal Appends	ConsecutiveLiteralAppends	pmd	MINOR	ACTIVE
Constant Name	com.puppycrawl.tools.checkstyle.checks.naming.ConstantNameCheck	checkstyle	MINOR	ACTIVE
Constructor Calls Overridable Method	ConstructorCallsOverridableMethod	pmd	MAJOR	ACTIVE
Correctness - "." used for regular expression	RE_POSSIBLE_UNINTENDED_PATTERN	findbugs	CRITICAL	ACTIVE
Correctness - A collection is added to itself	IL_CONTAINER_ADDED_TO_ITSELF	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Correctness - A known null value is checked to see if it is an instance of a type	NP_NULL_INSTANCEOF	findbugs	BLOCKER	ACTIVE
Correctness - A parameter is dead upon entry to a method but overwritten	IP_PARAMETER_IS_DEAD_BU T_OVERWRITTEN	findbugs	CRITICAL	ACTIVE
Correctness - An apparent infinite loop	IL_INFINITE_LOOP	findbugs	CRITICAL	ACTIVE
Correctness - An apparent infinite recursive loop	IL_INFINITE_RECURSIVE_LOOP	findbugs	CRITICAL	ACTIVE
Correctness - Apparent method/constructor confusion	NM_METHOD_CONSTRUCTOR _CONFUSION	findbugs	MAJOR	ACTIVE
Correctness - Array formatted in useless way using format string	VA_FORMAT_STRING_BAD_C ONVERSION_FROM_ARRAY	findbugs	MAJOR	ACTIVE
Correctness - Bad attempt to compute absolute value of signed 32-bit hashcode	RV_ABSOLUTE_VALUE_OF_HA SHCODE	findbugs	CRITICAL	ACTIVE
Correctness - Bad attempt to compute absolute value of signed 32-bit random integer	RV_ABSOLUTE_VALUE_OF_RA NDOM_INT	findbugs	CRITICAL	ACTIVE
Correctness - Bad comparison of nonnegative value with negative constant	INT_BAD_COMPARISON_WIT H_NONNEGATIVE_VALUE	findbugs	CRITICAL	ACTIVE
Correctness - Bad comparison of signed byte	INT_BAD_COMPARISON_WIT H_SIGNED_BYTE	findbugs	CRITICAL	ACTIVE
Correctness - Bad constant value for month	DMI_BAD_MONTH	findbugs	CRITICAL	ACTIVE
Correctness - Bitwise add of signed byte value	BIT_ADD_OF_SIGNED_BYTE	findbugs	CRITICAL	ACTIVE
Correctness - Bitwise OR of signed byte value	BIT_IOR_OF_SIGNED_BYTE	findbugs	CRITICAL	ACTIVE
Correctness - Call to equals() comparing different interface types	EC_UNRELATED_INTERFACES	findbugs	CRITICAL	ACTIVE
Correctness - Call to equals() comparing different types	EC_UNRELATED_TYPES	findbugs	CRITICAL	ACTIVE
Correctness - Call to equals() comparing unrelated class and interface	EC_UNRELATED_CLASS_AND_I NTERFACE	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Correctness - Call to equals() with null argument	EC_NULL_ARG	findbugs	CRITICAL	ACTIVE
Correctness - Can't use reflection to check for presence of annotation without runtime retention	DMI_ANNOTATION_IS_NOT_VISIBLE_TO_REFLECTION	findbugs	MAJOR	ACTIVE
Correctness - Check for sign of bitwise operation	BIT_SIGNED_CHECK_HIGH_BIT	findbugs	CRITICAL	ACTIVE
Correctness - Check to see if ((...) & 0) == 0	BIT_AND_ZZ	findbugs	CRITICAL	ACTIVE
Correctness - Class defines field that masks a superclass field	MF_CLASS_MASKS_FIELD	findbugs	MAJOR	ACTIVE
Correctness - Class overrides a method implemented in super class Adapter wrongly	BOA_BADLY_OVERRIDDEN_ADAPTER	findbugs	CRITICAL	ACTIVE
Correctness - close() invoked on a value that is always null	NP_CLOSING_NULL	findbugs	BLOCKER	ACTIVE
Correctness - Collections should not contain themselves	DMI_COLLECTIONS_SHOULD_NOT_CONTAIN_THEMSELVES	findbugs	CRITICAL	ACTIVE
Correctness - Covariant equals() method defined for enum	EQ_DONT_DEFINE_EQUALS_FOR_ENUM	findbugs	MAJOR	ACTIVE
Correctness - Covariant equals() method defined, Object.equals(Object) inherited	EQ_SELF_USE_OBJECT	findbugs	MAJOR	ACTIVE
Correctness - Creation of ScheduledThreadPoolExecutor with zero core threads	DMI_SCHEDULED_THREAD_POOL_EXECUTOR_WITH_ZERO_CORE_THREADS	findbugs	MINOR	ACTIVE
Correctness - Dead store of class literal	DLS_DEAD_STORE_OF_CLASS_LITERAL	findbugs	CRITICAL	ACTIVE
Correctness - Deadly embrace of non-static inner class and thread local	SIC_THREADLOCAL_DEADLY_EMBRACE	findbugs	MAJOR	ACTIVE
Correctness - Don't use removeAll to clear a collection	DMI_USING_REMOVEALL_TO_CLEAR_COLLECTION	findbugs	CRITICAL	ACTIVE
Correctness - Doomed attempt to append to an object output stream	IO_APPENDING_TO_OBJECT_OUTPUT_STREAM	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Correctness - Doomed test for equality to NaN	FE_TEST_IF_EQUAL_TO_NOT_A_NUMBER	findbugs	CRITICAL	ACTIVE
Correctness - Double assignment of field	SA_FIELD_DOUBLE_ASSIGNMENT	findbugs	CRITICAL	ACTIVE
Correctness - Double.longBitsToDouble invoked on an int	DMI_LONG_BITS_TO_DOUBLE_INVOKED_ON_INT	findbugs	CRITICAL	ACTIVE
Correctness - equals method always returns false	EQ_ALWAYS_FALSE	findbugs	BLOCKER	ACTIVE
Correctness - equals method always returns true	EQ_ALWAYS_TRUE	findbugs	BLOCKER	ACTIVE
Correctness - equals method compares class names rather than class objects	EQ_COMPARING_CLASS_NAMES	findbugs	MAJOR	ACTIVE
Correctness - equals method overrides equals in superclass and may not be symmetric	EQ_OVERRIDING_EQUALS_NOT_SYMMETRIC	findbugs	MAJOR	ACTIVE
Correctness - equals() method defined that doesn't override equals(Object)	EQ_OTHER_NO_OBJECT	findbugs	MAJOR	ACTIVE
Correctness - equals() method defined that doesn't override Object.equals(Object)	EQ_OTHER_USE_OBJECT	findbugs	MAJOR	ACTIVE
Correctness - equals() used to compare array and nonarray	EC_ARRAY_AND_NONARRAY	findbugs	CRITICAL	ACTIVE
Correctness - equals(...) used to compare incompatible arrays	EC_INCOMPATIBLE_ARRAY_COMPARE	findbugs	BLOCKER	ACTIVE
Correctness - Exception created and dropped rather than thrown	RV_EXCEPTION_NOT_THROWN	findbugs	CRITICAL	ACTIVE
Correctness - Field not initialized in constructor	UWF_FIELD_NOT_INITIALIZED_IN_CONSTRUCTOR	findbugs	MINOR	ACTIVE
Correctness - Field only ever set to null	UWF_NULL_FIELD	findbugs	CRITICAL	ACTIVE
Correctness - File.separator used for regular expression	RE_CANT_USE_FILE_SEPARATOR_AS_REGULAR_EXPRESSION	findbugs	CRITICAL	ACTIVE
Correctness - Format string placeholder incompatible with passed argument	VA_FORMAT_STRING_BAD_ARGUMENT	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Correctness - Format string references missing argument	VA_FORMAT_STRING_MISSING_ARGUMENT	findbugs	CRITICAL	ACTIVE
Correctness - Futile attempt to change max pool size of ScheduledThreadPoolExecutor	DMI_FUTILE_ATTEMPT_TO_CHANGE_MAXPOOL_SIZE_OF_SCHEDULED_THREAD_POOL_EXECUTOR	findbugs	MINOR	ACTIVE
Correctness - hasNext method invokes next	DMI_CALLING_NEXT_FROM_HASNEXT	findbugs	CRITICAL	ACTIVE
Correctness - Illegal format string	VA_FORMAT_STRING_ILLEGAL	findbugs	CRITICAL	ACTIVE
Correctness - Impossible cast	BC_IMPOSSIBLE_CAST	findbugs	BLOCKER	ACTIVE
Correctness - Impossible downcast	BC_IMPOSSIBLE_DOWNCAST	findbugs	BLOCKER	ACTIVE
Correctness - Impossible downcast of toArray() result	BC_IMPOSSIBLE_DOWNCAST_OF_TOARRAY	findbugs	BLOCKER	ACTIVE
Correctness - Incompatible bit masks (BIT_AND)	BIT_AND	findbugs	CRITICAL	ACTIVE
Correctness - Incompatible bit masks (BIT_IOR)	BIT_IOR	findbugs	CRITICAL	ACTIVE
Correctness - instanceof will always return false	BC_IMPOSSIBLE_INSTANCEOF	findbugs	CRITICAL	ACTIVE
Correctness - int value cast to double and then passed to Math.ceil	ICAST_INT_CAST_TO_DOUBLE_PASSED_TO_CEIL	findbugs	CRITICAL	ACTIVE
Correctness - int value cast to float and then passed to Math.round	ICAST_INT_CAST_TO_FLOAT_PASSED_TO_ROUND	findbugs	CRITICAL	ACTIVE
Correctness - Integer multiply of result of integer remainder	IM_MULTIPLYING_RESULT_OF_IREM	findbugs	CRITICAL	ACTIVE
Correctness - Integer remainder modulo 1	INT_BAD_REM_BY_1	findbugs	CRITICAL	ACTIVE
Correctness - Integer shift by an amount not in the range 0..31	ICAST_BAD_SHIFT_AMOUNT	findbugs	CRITICAL	ACTIVE
Correctness - Invalid syntax for regular expression	RE_BAD_SYNTAX_FOR_REGULAR_EXPRESSION	findbugs	CRITICAL	ACTIVE
Correctness - Invocation of equals() on an array, which is equivalent to ==	EC_BAD_ARRAY_COMPARE	findbugs	CRITICAL	ACTIVE
Correctness - Invocation of hashCode on an array	DMI_INVOKING_HASHCODE_ON_ARRAY	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Correctness - Invocation of toString on an anonymous array	DMI_INVOKING_TOSTRING_ON_ANONYMOUS_ARRAY	findbugs	CRITICAL	ACTIVE
Correctness - Invocation of toString on an array	DMI_INVOKING_TOSTRING_ON_ARRAY	findbugs	CRITICAL	ACTIVE
Correctness - JUnit assertion in run method will not be noticed by JUnit	IJU_ASSERT_METHOD_INVOKED_FROM_RUN_METHOD	findbugs	CRITICAL	ACTIVE
Correctness - MessageFormat supplied where printf style format expected	VA_FORMAT_STRING_EXPECTED_MESSAGE_FORMAT_SUPPLIED	findbugs	MAJOR	ACTIVE
Correctness - Method assigns boolean literal in boolean expression	QBA_QUESTIONABLE_BOOLEAN_ASSIGNMENT	findbugs	CRITICAL	ACTIVE
Correctness - Method attempts to access a prepared statement parameter with index 0	SQL_BAD_PREPARED_STATEMENT_ACCESS	findbugs	CRITICAL	ACTIVE
Correctness - Method attempts to access a result set field with index 0	SQL_BAD_RESULTSET_ACCESS	findbugs	CRITICAL	ACTIVE
Correctness - Method call passes null for nonnull parameter	NP_NULL_PARAM_DEREF	findbugs	CRITICAL	ACTIVE
Correctness - Method call passes null for nonnull parameter (ALL_TARGETS_DANGEROUS)	NP_NULL_PARAM_DEREF_ALL_TARGETS_DANGEROUS	findbugs	CRITICAL	ACTIVE
Correctness - Method call passes null to a nonnull parameter	NP_NONNULL_PARAM_VIOLATION	findbugs	CRITICAL	ACTIVE
Correctness - Method defines a variable that obscures a field	MF_METHOD_MASKS_FIELD	findbugs	MAJOR	ACTIVE
Correctness - Method does not check for null argument	NP_ARGUMENT_MIGHT_BE_NULL	findbugs	MAJOR	ACTIVE
Correctness - Method doesn't override method in superclass due to wrong package for parameter	NM_WRONG_PACKAGE	findbugs	MAJOR	ACTIVE
Correctness - Method ignores return value	RV_RETURN_VALUE_IGNORED	findbugs	MINOR	ACTIVE

title	Key	plugin	priority	status
Correctness - Method ignores return value	RV_RETURN_VALUE_IGNORED 2	findbugs	MAJOR	ACTIVE
Correctness - Method may return null, but is declared @NonNull	NP_NONNULL_RETURN_VIOL ATION	findbugs	CRITICAL	ACTIVE
Correctness - Method must be private in order for serialization to work	SE_METHOD_MUST_BE_PRIV ATE	findbugs	MAJOR	ACTIVE
Correctness - Method performs math using floating point precision	FL_MATH_USING_FLOAT_PRE CISION	findbugs	CRITICAL	ACTIVE
Correctness - More arguments are passed that are actually used in the format string	VA_FORMAT_STRING_EXTRA_ ARGUMENTS_PASSED	findbugs	MAJOR	ACTIVE
Correctness - No previous argument for format string	VA_FORMAT_STRING_NO_PR EVIOUS_ARGUMENT	findbugs	CRITICAL	ACTIVE
Correctness - No relationship between generic parameter and method argument	GC_UNRELATED_TYPES	findbugs	CRITICAL	ACTIVE
Correctness - Non-virtual method call passes null for nonnull parameter	NP_NULL_PARAM_DEREF_NO NVIRTUAL	findbugs	CRITICAL	ACTIVE
Correctness - Nonsensical self computation involving a field (e.g., x & x)	SA_FIELD_SELF_COMPUTATIO N	findbugs	CRITICAL	ACTIVE
Correctness - Nonsensical self computation involving a variable (e.g., x & x)	SA_LOCAL_SELF_COMPUTATI ON	findbugs	CRITICAL	ACTIVE
Correctness - Null pointer dereference	NP_ALWAYS_NULL	findbugs	CRITICAL	ACTIVE
Correctness - Null pointer dereference in method on exception path	NP_ALWAYS_NULL_EXCEPTIO N	findbugs	CRITICAL	ACTIVE
Correctness - Null value is guaranteed to be dereferenced	NP_GUARANTEED_DEREF	findbugs	BLOCKER	ACTIVE
Correctness - Nullcheck of value previously dereferenced	RCN_REDUNDANT_NULLCHEC K_WOULD_HAVE_BEEN_A_NP E	findbugs	CRITICAL	ACTIVE
Correctness - Number of format-string arguments does not correspond to	VA_FORMAT_STRING_ARG_M ISMATCH	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
number of placeholders				
Correctness - Overwritten increment	DLS_OVERWRITTEN_INCREMENT	findbugs	CRITICAL	ACTIVE
Correctness - Possible null pointer dereference	NP_NULL_ON_SOME_PATH	findbugs	CRITICAL	ACTIVE
Correctness - Possible null pointer dereference in method on exception path	NP_NULL_ON_SOME_PATH_EXCEPTION	findbugs	CRITICAL	ACTIVE
Correctness - Primitive array passed to function expecting a variable number of object arguments	VA_PRIMITIVE_ARRAY_PASSED_TO_OBJECT_VARARG	findbugs	CRITICAL	ACTIVE
Correctness - Primitive value is unboxed and coerced for ternary operator	BX_UNBOXED_AND_COERCED_FOR_TERNARY_OPERATOR	findbugs	MAJOR	ACTIVE
Correctness - Random value from 0 to 1 is coerced to the integer 0	RV_01_TO_INT	findbugs	MAJOR	ACTIVE
Correctness - Read of unwritten field	NP_UNWRITTEN_FIELD	findbugs	MAJOR	ACTIVE
Correctness - Repeated conditional tests	RpC_REPEATED_CONDITIONAL_TEST	findbugs	MAJOR	ACTIVE
Correctness - Return value of putIfAbsent ignored, value passed to putIfAbsent reused	RV_RETURN_VALUE_OF_PUTIFABSENT_IGNORED	findbugs	MAJOR	ACTIVE
Correctness - Self assignment of field	SA_FIELD_SELF_ASSIGNMENT	findbugs	CRITICAL	ACTIVE
Correctness - Self comparison of field with itself	SA_FIELD_SELF_COMPARISON	findbugs	CRITICAL	ACTIVE
Correctness - Self comparison of value with itself	SA_LOCAL_SELF_COMPARISON	findbugs	CRITICAL	ACTIVE
Correctness - Signature declares use of unhashable class in hashed construct	HE_SIGNATURE_DECLARES_HASHING_OF_UNHASHABLE_CLASS	findbugs	CRITICAL	ACTIVE
Correctness - Static Thread.interrupted() method invoked on thread instance	STI_INTERRUPTED_ON_UNKNOWNTHREAD	findbugs	CRITICAL	ACTIVE
Correctness - Store of null value into field annotated NonNull	NP_STORE_INTO_NONNULL_FIELD	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Correctness - Suspicious reference comparison of Boolean values	RC_REF_COMPARISON_BAD_PRACTICE_BOOLEAN	findbugs	MAJOR	ACTIVE
Correctness - Suspicious reference comparison to constant	RC_REF_COMPARISON_BAD_PRACTICE	findbugs	MAJOR	ACTIVE
Correctness - TestCase declares a bad suite method	IJU_BAD_SUITE_METHOD	findbugs	CRITICAL	ACTIVE
Correctness - TestCase defines setUp that doesn't call super.setUp()	IJU_SETUP_NO_SUPER	findbugs	CRITICAL	ACTIVE
Correctness - TestCase defines tearDown that doesn't call super.tearDown()	IJU_TEARDOWN_NO_SUPER	findbugs	CRITICAL	ACTIVE
Correctness - TestCase has no tests	IJU_NO_TESTS	findbugs	CRITICAL	ACTIVE
Correctness - TestCase implements a non-static suite method	IJU_SUITE_NOT_STATIC	findbugs	CRITICAL	ACTIVE
Correctness - The readResolve method must not be declared as a static method.	SE_READ_RESOLVE_IS_STATIC	findbugs	MAJOR	ACTIVE
Correctness - The type of a supplied argument doesn't match format specifier	VA_FORMAT_STRING_BAD_CONVERSION	findbugs	CRITICAL	ACTIVE
Correctness - Uncallable method defined in anonymous class	UMAC_UNCALLABLE_METHOD_OF_ANONYMOUS_CLASS	findbugs	CRITICAL	ACTIVE
Correctness - Uninitialized read of field in constructor	UR_UNINIT_READ	findbugs	MAJOR	ACTIVE
Correctness - Uninitialized read of field method called from constructor of superclass	UR_UNINIT_READ_CALLED_FROM_SUPER_CONSTRUCTOR	findbugs	MAJOR	ACTIVE
Correctness - Unnecessary type check done using instanceof operator	SIO_SUPERFLUOUS_INSTANCEOF	findbugs	CRITICAL	ACTIVE
Correctness - Unneeded use of currentThread() call, to call interrupted()	STI_INTERRUPTED_ON_CURRENTTHREAD	findbugs	CRITICAL	ACTIVE
Correctness - Unwritten field	UWF_UNWRITTEN_FIELD	findbugs	MINOR	ACTIVE

title	Key	plugin	priority	status
Correctness - Use of class without a hashCode() method in a hashed data structure	HE_USE_OF_UNHASHABLE_CLASS	findbugs	CRITICAL	ACTIVE
Correctness - Useless assignment in return statement	DLS_DEAD_LOCAL_STORE_IN_RETURN	findbugs	CRITICAL	ACTIVE
Correctness - Useless control flow to next line	UCF_USELESS_CONTROL_FLOW_NEXT_LINE	findbugs	CRITICAL	ACTIVE
Correctness - Using pointer equality to compare different types	EC_UNRELATED_TYPES_USING_POINTER_EQUALITY	findbugs	CRITICAL	ACTIVE
Correctness - Vacuous call to collections	DMI_VACUOUS_SELF_COLLECTION_CALL	findbugs	CRITICAL	ACTIVE
Correctness - Value annotated as carrying a type qualifier used where a value that must not carry that qualifier is required	TQ_ALWAYS_VALUE_USED_WHERE_NEVER_REQUIRED	findbugs	CRITICAL	ACTIVE
Correctness - Value annotated as never carrying a type qualifier used where value carrying that qualifier is required	TQ_NEVER_VALUE_USED_WHERE_ALWAYS_REQUIRED	findbugs	CRITICAL	ACTIVE
Correctness - Value is null and guaranteed to be dereferenced on exception path	NP_GUARANTEED_DEREF_ON_EXCEPTION_PATH	findbugs	CRITICAL	ACTIVE
Correctness - Value required to have type qualifier, but marked as unknown	TQ_EXPLICIT_UNKNOWN_SOURCE_VALUE_REACHES_ALWAYS_SINK	findbugs	CRITICAL	ACTIVE
Correctness - Value required to not have type qualifier, but marked as unknown	TQ_EXPLICIT_UNKNOWN_SOURCE_VALUE_REACHES_NEVER_SINK	findbugs	CRITICAL	ACTIVE
Correctness - Value that might carry a type qualifier is always used in a way prohibits it from having that type qualifier	TQ_MAYBE_SOURCE_VALUE_REACHES_NEVER_SINK	findbugs	CRITICAL	ACTIVE
Correctness - Value that might not carry a type qualifier is always used in a way requires that type qualifier	TQ_MAYBE_SOURCE_VALUE_REACHES_ALWAYS_SINK	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Correctness - Very confusing method names	NM_VERY_CONFUSING	findbugs	MAJOR	ACTIVE
Coupling - excessive imports	ExcessiveImports	pmd	MAJOR	ACTIVE
Coupling between objects	CouplingBetweenObjects	pmd	MAJOR	ACTIVE
Cyclomatic Complexity	com.puppycrawl.tools.checkstyle.checks.metrics.CyclomaticComplexityCheck	checkstyle	MAJOR	ACTIVE
Dataflow Anomaly Analysis	DataflowAnomalyAnalysis	pmd	INFO	ACTIVE
Default label not last in switch statement	DefaultLabelNotLastInSwitchStmt	pmd	MAJOR	ACTIVE
Default Package	DefaultPackage	pmd	MINOR	ACTIVE
Design For Extension	com.puppycrawl.tools.checkstyle.checks.design.DesignForExtensionCheck	checkstyle	INFO	ACTIVE
Do not call garbage collection explicitly	DoNotCallGarbageCollectionExplicitly	pmd	CRITICAL	ACTIVE
Do Not Extend Java Lang Error	DoNotExtendJavaLangError	pmd	MAJOR	ACTIVE
Do Not Use Threads	DoNotUseThreads	pmd	MAJOR	ACTIVE
Dodgy - Ambiguous invocation of either an inherited or outer method	IA_AMBIGUOUS_INVOCATION_OF_INHERITED_OR_OUTER_METHOD	findbugs	MAJOR	ACTIVE
Dodgy - Call to unsupported method	DMI_UNSUPPORTED_METHOD	findbugs	MAJOR	ACTIVE
Dodgy - Check for oddness that won't work for negative numbers	IM_BAD_CHECK_FOR_ODD	findbugs	CRITICAL	ACTIVE
Dodgy - Class exposes synchronization and semaphores in its public interface	PS_PUBLIC_SEMAPHORES	findbugs	CRITICAL	ACTIVE
Dodgy - Class extends Servlet class and uses instance variables	MTIA_SUSPECT_SERVLET_INSTANCE_FIELD	findbugs	CRITICAL	ACTIVE
Dodgy - Class extends Struts Action class and uses instance variables	MTIA_SUSPECT_STRUTS_INSTANCE_FIELD	findbugs	CRITICAL	ACTIVE
Dodgy - Class implements same interface as superclass	RI_REDUNDANT_INTERFACES	findbugs	MAJOR	ACTIVE
Dodgy - Class is final but declares protected field	CI_CONFUSED_INHERITANCE	findbugs	MINOR	ACTIVE
Dodgy - Class too big for	SKIPPED_CLASS_TOO_BIG	findbugs	MINOR	ACTIVE

title	Key	plugin	priority	status
analysis				
Dodgy - Code contains a hard coded reference to an absolute pathname	DMI_HARDCODED_ABSOLUTE_FILENAME	findbugs	CRITICAL	ACTIVE
Dodgy - Complicated, subtle or wrong increment in for-loop	QF_QUESTIONABLE_FOR_LOOP	findbugs	CRITICAL	ACTIVE
Dodgy - Computation of average could overflow	IM_AVERAGE_COMPUTATION_COULD_OVERFLOW	findbugs	CRITICAL	ACTIVE
Dodgy - Consider returning a zero length array rather than null	PZLA_PREFER_ZERO_LENGTH_ARRAYS	findbugs	MAJOR	ACTIVE
Dodgy - Dead store of null to local variable	DLS_DEAD_LOCAL_STORE_OF_NULL	findbugs	CRITICAL	ACTIVE
Dodgy - Dead store to local variable	DLS_DEAD_LOCAL_STORE	findbugs	CRITICAL	ACTIVE
Dodgy - Dereference of the result of readLine() without nullcheck	NP_DEREFERENCE_OF_READLINE_VALUE	findbugs	CRITICAL	ACTIVE
Dodgy - Double assignment of local variable	SA_LOCAL_DOUBLE_ASSIGNMENT	findbugs	CRITICAL	ACTIVE
Dodgy - Exception is caught when Exception is not thrown	REC_CATCH_EXCEPTION	findbugs	MAJOR	ACTIVE
Dodgy - Immediate dereference of the result of readLine()	NP_IMMEDIATE_DEREFERENCE_OF_READLINE	findbugs	CRITICAL	ACTIVE
Dodgy - Initialization circularity	IC_INIT_CIRCULARITY	findbugs	CRITICAL	ACTIVE
Dodgy - instanceof will always return true	BC_VACUOUS_INSTANCEOF	findbugs	CRITICAL	ACTIVE
Dodgy - int division result cast to double or float	ICAST_IDIV_CAST_TO_DOUBLE	findbugs	CRITICAL	ACTIVE
Dodgy - Invocation of substring(0), which returns the original value	DMI_USELESS_SUBSTRING	findbugs	CRITICAL	ACTIVE
Dodgy - Load of known null value	NP_LOAD_OF_KNOWN_NULL_VALUE	findbugs	CRITICAL	ACTIVE
Dodgy - Method checks to see if result of String.indexOf is positive	RV_CHECK_FOR_POSITIVE_INDEXOF	findbugs	MINOR	ACTIVE

title	Key	plugin	priority	status
Dodgy - Method directly allocates a specific implementation of xml interfaces	XFB_XML_FACTORY_BYPASS	findbugs	CRITICAL	ACTIVE
Dodgy - Method discards result of readLine after checking if it is nonnull	RV_DONT_JUST_NULL_CHECK_READLINE	findbugs	MAJOR	ACTIVE
Dodgy - Method uses the same code for two branches	DB_DUPLICATE_BRANCHES	findbugs	CRITICAL	ACTIVE
Dodgy - Method uses the same code for two switch clauses	DB_DUPLICATE_SWITCH_CLAUSES	findbugs	CRITICAL	ACTIVE
Dodgy - Non serializable object written to ObjectOutput	DMI_NONSERIALIZABLE_OBJECT_WRITTEN	findbugs	CRITICAL	ACTIVE
Dodgy - Non-Boolean argument formatted using %b format specifier	VA_FORMAT_STRING_BAD_CONVERSION_TO_BOOLEAN	findbugs	MAJOR	ACTIVE
Dodgy - Parameter must be nonnull but is marked as nullable	NP_PARAMETER_MUST_BE_NONNULL_BUT_MARKED_AS_NULLABLE	findbugs	CRITICAL	ACTIVE
Dodgy - Possible null pointer dereference due to return value of called method	NP_NULL_ON_SOME_PATH_FROM_RETURN_VALUE	findbugs	CRITICAL	ACTIVE
Dodgy - Possible null pointer dereference on path that might be infeasible	NP_NULL_ON_SOME_PATH_MIGHT_BE_INFEASIBLE	findbugs	CRITICAL	ACTIVE
Dodgy - Potentially dangerous use of non-short-circuit logic	NS_DANGEROUS_NON_SHORT_CIRCUIT	findbugs	CRITICAL	ACTIVE
Dodgy - private readResolve method not inherited by subclasses	SE_PRIVATE_READ_RESOLVE_NOT_INHERITED	findbugs	MAJOR	ACTIVE
Dodgy - Questionable cast to abstract collection	BC_BAD_CAST_TO_ABSTRACT_COLLECTION	findbugs	MAJOR	ACTIVE
Dodgy - Questionable cast to concrete collection	BC_BAD_CAST_TO_CONCRETE_COLLECTION	findbugs	CRITICAL	ACTIVE
Dodgy - Questionable use of non-short-circuit logic	NS_NON_SHORT_CIRCUIT	findbugs	MAJOR	ACTIVE
Dodgy - Redundant comparison of non-null value to null	RCN_REDUNDANT_COMPARISON_OF_NULL_AND_NONNULL_VALUE	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Dodgy - Redundant comparison of two null values	RCN_REDUNDANT_COMPARISON_TWO_NULL_VALUES	findbugs	CRITICAL	ACTIVE
Dodgy - Redundant nullcheck of value known to be non-null	RCN_REDUNDANT_NULLCHECK_OF_NONNULL_VALUE	findbugs	CRITICAL	ACTIVE
Dodgy - Redundant nullcheck of value known to be null	RCN_REDUNDANT_NULLCHECK_OF_NULL_VALUE	findbugs	CRITICAL	ACTIVE
Dodgy - Remainder of 32-bit signed random integer	RV_REM_OF_RANDOM_INT	findbugs	CRITICAL	ACTIVE
Dodgy - Remainder of hashCode could be negative	RV_REM_OF_HASHCODE	findbugs	CRITICAL	ACTIVE
Dodgy - Result of integer multiplication cast to long	ICAST_INTEGER_MULTIPLY_CAST_TO_LONG	findbugs	CRITICAL	ACTIVE
Dodgy - Self assignment of local variable	SA_LOCAL_SELF_ASSIGNMENT	findbugs	CRITICAL	ACTIVE
Dodgy - Test for floating point equality	FE_FLOATING_POINT_EQUALITY	findbugs	CRITICAL	ACTIVE
Dodgy - Thread passed where Runnable expected	DMI_THREAD_PASSED_WHERE_RUNNABLE_EXPECTED	findbugs	MAJOR	ACTIVE
Dodgy - Transient field of class that isn't Serializable.	SE_TRANSIENT_FIELD_OF_NONSERIALIZABLE_CLASS	findbugs	MAJOR	ACTIVE
Dodgy - Unchecked/unconfirmed cast	BC_UNCONFIRMED_CAST	findbugs	CRITICAL	ACTIVE
Dodgy - Unsigned right shift cast to short/byte	ICAST_QUESTIONABLE_UNSIGNED_RIGHT_SHIFT	findbugs	CRITICAL	ACTIVE
Dodgy - Unusual equals method	EQ_UNUSUAL	findbugs	MINOR	ACTIVE
Dodgy - Vacuous bit mask operation on integer value	INT_VACUOUS_BIT_OPERATION	findbugs	CRITICAL	ACTIVE
Dodgy - Vacuous comparison of integer value	INT_VACUOUS_COMPARISON	findbugs	CRITICAL	ACTIVE
Dodgy - Write to static field from instance method	ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD	findbugs	CRITICAL	ACTIVE
Dont Import Java Lang	DontImportJavaLang	pmd	MINOR	ACTIVE
Dont Import Sun	DontImportSun	pmd	MINOR	ACTIVE
Dont Nest Jsf In Jstl Iteration	DontNestJsfInJstlIteration	pmd	MAJOR	ACTIVE
Double checked locking	DoubleCheckedLocking	pmd	MAJOR	ACTIVE
Duplicate Imports	DuplicateImports	pmd	MINOR	ACTIVE
Empty Catch Block	EmptyCatchBlock	pmd	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Empty Finalizer	EmptyFinalizer	pmd	MAJOR	ACTIVE
Empty Finally Block	EmptyFinallyBlock	pmd	CRITICAL	ACTIVE
Empty If Stmt	EmptyIfStmt	pmd	CRITICAL	ACTIVE
Empty Method In Abstract Class Should Be Abstract	EmptyMethodInAbstractClass ShouldBeAbstract	pmd	MAJOR	ACTIVE
Empty Statement	com.puppycrawl.tools.checkstyle.checks.coding.EmptyStatementCheck	checkstyle	MINOR	ACTIVE
Empty Statement Not In Loop	EmptyStatementNotInLoop	pmd	MAJOR	ACTIVE
Empty Static Initializer	EmptyStaticInitializer	pmd	MAJOR	ACTIVE
Empty Switch Statements	EmptySwitchStatements	pmd	MAJOR	ACTIVE
Empty Synchronized Block	EmptySynchronizedBlock	pmd	CRITICAL	ACTIVE
Empty Try Block	EmptyTryBlock	pmd	MAJOR	ACTIVE
Empty While Stmt	EmptyWhileStmt	pmd	CRITICAL	ACTIVE
Equals Hash Code	com.puppycrawl.tools.checkstyle.checks.coding.EqualsHashCodeCheck	checkstyle	CRITICAL	ACTIVE
Equals Null	EqualsNull	pmd	CRITICAL	ACTIVE
Exception As Flow Control	ExceptionAsFlowControl	pmd	MAJOR	ACTIVE
Excessive Parameter List	ExcessiveParameterList	pmd	MAJOR	ACTIVE
Excessive Public Count	ExcessivePublicCount	pmd	MAJOR	ACTIVE
Final Class	com.puppycrawl.tools.checkstyle.checks.design.FinalClassCheck	checkstyle	MAJOR	ACTIVE
Final Field Could Be Static	FinalFieldCouldBeStatic	pmd	MINOR	ACTIVE
Finalize Does Not Call Super Finalize	FinalizeDoesNotCallSuperFinalize	pmd	MAJOR	ACTIVE
Finalize Only Calls Super Finalize	FinalizeOnlyCallsSuperFinalize	pmd	MAJOR	ACTIVE
Finalize Overloaded	FinalizeOverloaded	pmd	MAJOR	ACTIVE
Finalize Should Be Protected	FinalizeShouldBeProtected	pmd	MAJOR	ACTIVE
For Loop Should Be While Loop	ForLoopShouldBeWhileLoop	pmd	MINOR	ACTIVE
For Loops Must Use Braces	ForLoopsMustUseBraces	pmd	MAJOR	ACTIVE
Hidden Field	com.puppycrawl.tools.checkstyle.checks.coding.HiddenFieldCheck	checkstyle	MAJOR	ACTIVE
Hide Utility Class Constructor	com.puppycrawl.tools.checkstyle.checks.design.HideUtilityClassConstructorCheck	checkstyle	MAJOR	ACTIVE
Idempotent Operations	IdempotentOperations	pmd	MAJOR	ACTIVE

title	Key	plugin	priority	status
If Else Stmts Must Use Braces	IfElseStmtsMustUseBraces	pmd	MAJOR	ACTIVE
If Stmts Must Use Braces	IfStmtsMustUseBraces	pmd	MAJOR	ACTIVE
Illegal Throws	com.puppycrawl.tools.checkstyle.checks.coding.IllegalThrowsCheck	checkstyle	MAJOR	ACTIVE
Immutable Field	ImmutableField	pmd	MAJOR	ACTIVE
Import From Same Package	ImportFromSamePackage	pmd	MINOR	ACTIVE
Inefficient Empty String Check	InefficientEmptyStringCheck	pmd	MAJOR	ACTIVE
Inefficient String Buffering	InefficientStringBuffering	pmd	MAJOR	ACTIVE
Inner Assignment	com.puppycrawl.tools.checkstyle.checks.coding.InnerAssignmentCheck	checkstyle	MAJOR	ACTIVE
Instantiation To Get Class	InstantiationToGetClass	pmd	MAJOR	ACTIVE
Insufficient String Buffer Declaration	InsufficientStringBufferDeclaration	pmd	MAJOR	ACTIVE
Integer Instantiation	IntegerInstantiation	pmd	MAJOR	ACTIVE
Internationalization - Consider using Locale parameterized version of invoked method	DM_CONVERT_CASE	findbugs	INFO	ACTIVE
Java5 migration - Byte instantiation	ByteInstantiation	pmd	MAJOR	ACTIVE
Java5 migration - Long instantiation	LongInstantiation	pmd	MAJOR	ACTIVE
Java5 migration - Short instantiation	ShortInstantiation	pmd	MAJOR	ACTIVE
Jumbled Incrementer	JumbledIncrementer	pmd	MAJOR	ACTIVE
Local Final Variable Name	com.puppycrawl.tools.checkstyle.checks.naming.LocalFinalVariableNameCheck	checkstyle	MAJOR	ACTIVE
Local Home Naming Convention	LocalHomeNamingConvention	pmd	MAJOR	ACTIVE
Local Interface Session Naming Convention	LocalInterfaceSessionNamingConvention	pmd	MAJOR	ACTIVE
Local Variable Name	com.puppycrawl.tools.checkstyle.checks.naming.LocalVariableNameCheck	checkstyle	MINOR	ACTIVE
Logger Is Not Static Final	LoggerIsNotStaticFinal	pmd	MAJOR	ACTIVE
Long Variable	LongVariable	pmd	MAJOR	ACTIVE
Loose coupling	LooseCoupling	pmd	MAJOR	ACTIVE

title	Key	plugin	priority	status
Magic Number	com.puppycrawl.tools.checkstyle.checks.coding.MagicNumberCheck	checkstyle	MINOR	ACTIVE
Malicious code vulnerability - Field is a mutable array	MS_MUTABLE_ARRAY	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - Field is a mutable Hashtable	MS_MUTABLE_HASHTABLE	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - Field isn't final and can't be protected from malicious code	MS_CANNOT_BE_FINAL	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - Field isn't final but should be	MS_SHOULD_BE_FINAL	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - Field should be both final and package protected	MS_FINAL_PKGPROTECT	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - Field should be moved out of an interface and made package protected	MS_OOI_PKGPROTECT	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - Field should be package protected	MS_PKGPROTECT	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - Finalizer should be protected, not public	FI_PUBLIC_SHOULD_BE_PROTECTED	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - May expose internal representation by incorporating reference to mutable object	EI_EXPOSE_REP2	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - May expose internal representation by returning reference to mutable object	EI_EXPOSE_REP	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - May expose internal static state by storing a mutable object into a static field	EI_EXPOSE_STATIC_REP2	findbugs	MAJOR	ACTIVE
Malicious code vulnerability - Public static method may expose internal representation by returning	MS_EXPOSE_REP	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
array				
Member name	com.puppycrawl.tools.checkstyle.checks.naming.MemberNameCheck	checkstyle	MAJOR	ACTIVE
Message Driven Bean And Session Bean Naming Convention	MDBAndSessionBeanNamingConvention	pmd	MAJOR	ACTIVE
Misplaced Null Check	MisplacedNullCheck	pmd	CRITICAL	ACTIVE
Missing Break In Switch	MissingBreakInSwitch	pmd	CRITICAL	ACTIVE
Missing Serial Version UID	MissingSerialVersionUID	pmd	MINOR	ACTIVE
Missing Static Method In Non Instantiatable Class	MissingStaticMethodInNonInstantiatableClass	pmd	MAJOR	ACTIVE
Modifier Order	com.puppycrawl.tools.checkstyle.checks.modifier.ModifierOrderCheck	checkstyle	MINOR	ACTIVE
More Than One Logger	MoreThanOneLogger	pmd	MAJOR	ACTIVE
Multithreaded correctness - A thread was created using the default empty run method	DM_USELESS_THREAD	findbugs	MAJOR	ACTIVE
Multithreaded correctness - A volatile reference to an array doesn't treat the array elements as volatile	VO_VOLATILE_REFERENCE_TO_ARRAY	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Call to static Calendar	STCAL_INVOKE_ON_STATIC_CALENDAR_INSTANCE	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Call to static DateFormat	STCAL_INVOKE_ON_STATIC_DATE_FORMAT_INSTANCE	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Class's readObject() method is synchronized	RS_READOBJECT_SYNC	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Class's writeObject() method is synchronized but nothing else is	WS_WRITEOBJECT_SYNC	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Condition.await() not in loop	WA_AWAIT_NOT_IN_LOOP	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Constructor invokes Thread.start()	SC_START_IN_CTOR	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Multithreaded correctness - Field not guarded against concurrent access	IS_FIELD_NOT_GUARDED	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Inconsistent synchronization	IS_INCONSISTENT_SYNC	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Inconsistent synchronization	IS2_INCONSISTENT_SYNC	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Incorrect lazy initialization and update of static field	LI_LAZY_INIT_UPDATE_STATIC	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Incorrect lazy initialization of static field	LI_LAZY_INIT_STATIC	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Invokes run on a thread (did you mean to start it instead?)	RU_INVOKE_RUN	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Method calls Thread.sleep() with a lock held	SWL_SLEEP_WITH_LOCK_HEL D	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Method does not release lock on all exception paths	UL_UNRELEASED_LOCK_EXCE PTION_PATH	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Method does not release lock on all paths	UL_UNRELEASED_LOCK	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Method spins on field	SP_SPIN_ON_FIELD	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Method synchronizes on an updated field	ML_SYNC_ON_UPDATED_FIEL D	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Mismatched notify()	MWN_MISMATCHED_NOTIFY	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Mismatched wait()	MWN_MISMATCHED_WAIT	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Monitor wait() called on Condition	DM_MONITOR_WAIT_ON_CO NDITION	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Mutable servlet field	MSF_MUTABLE_SERVLET_FIEL D	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Naked notify	NN_NAKED_NOTIFY	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Multithreaded correctness - Static Calendar	STCAL_STATIC_CALENDAR_INSTANCE	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Static DateFormat	STCAL_STATIC_SIMPLE_DATE_FORMAT_INSTANCE	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Sychronization on getClass rather than class literal	WL_USING_GETCLASS_RATHER_THAN_CLASS_LITERAL	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Synchronization on Boolean could lead to deadlock	DL_SYNCHRONIZATION_ON_BOOLEAN	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Synchronization on boxed primitive could lead to deadlock	DL_SYNCHRONIZATION_ON_BOXED_PRIMITIVE	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Synchronization on boxed primitive values	DL_SYNCHRONIZATION_ON_UNSHARED_BOXED_PRIMITIVE	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Synchronization on field in futile attempt to guard that field	ML_SYNC_ON_FIELD_TO_GUARD_CHANGING_THAT_FIELD	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Synchronization on interned String could lead to deadlock	DL_SYNCHRONIZATION_ON_SHARED_CONSTANT	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Synchronization performed on java.util.concurrent Lock	JLM_JSR166_LOCK_MONITORENTER	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Synchronize and null check on the same field.	NP_SYNC_AND_NULL_CHECK_FIELD	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Unconditional wait	UW_UNCOND_WAIT	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Unsynchronized get method, synchronized set method	UG_SYNC_SET_UNSYNC_GET	findbugs	MAJOR	ACTIVE
Multithreaded correctness - Using notify() rather than notifyAll()	NO_NOTIFY_NOT_NOTIFYALL	findbugs	CRITICAL	ACTIVE
Multithreaded correctness - Wait not in loop	WA_NOT_IN_LOOP	findbugs	CRITICAL	ACTIVE

title	Key	plugin	priority	status
Multithreaded correctness - Wait with two locks held	TLW_TWO_LOCK_WAIT	findbugs	MAJOR	ACTIVE
Naming - Avoid dollar signs	AvoidDollarSigns	pmd	MINOR	ACTIVE
Naming - Avoid field name matching method name	AvoidFieldNameMatchingMethodName	pmd	MAJOR	ACTIVE
Naming - Avoid field name matching type name	AvoidFieldNameMatchingTypeName	pmd	MAJOR	ACTIVE
Naming - Class naming conventions	ClassNamingConventions	pmd	MAJOR	ACTIVE
Naming - Method naming conventions	MethodNamingConventions	pmd	MAJOR	ACTIVE
Naming - Method with same name as enclosing class	MethodWithSameNameAsEnclosingClass	pmd	MAJOR	ACTIVE
Naming - Misleading variable name	MisleadingVariableName	pmd	MAJOR	ACTIVE
Naming - Short method name	ShortMethodName	pmd	MAJOR	ACTIVE
Naming - Suspicious constant field name	SuspiciousConstantFieldName	pmd	MAJOR	ACTIVE
Naming - Suspicious equals method name	SuspiciousEqualsMethodName	pmd	CRITICAL	ACTIVE
Naming - Suspicious Hashcode method name	SuspiciousHashcodeMethodName	pmd	MAJOR	ACTIVE
Ncss Constructor Count	NcssConstructorCount	pmd	MAJOR	ACTIVE
Ncss Method Count	NcssMethodCount	pmd	MAJOR	ACTIVE
Ncss Type Count	NcssTypeCount	pmd	MAJOR	ACTIVE
No package	NoPackage	pmd	MAJOR	ACTIVE
Non Case Label In Switch Statement	NonCaseLabelInSwitchStatement	pmd	MAJOR	ACTIVE
Non Static Initializer	NonStaticInitializer	pmd	MAJOR	ACTIVE
Non Thread Safe Singleton	NonThreadSafeSingleton	pmd	MAJOR	ACTIVE
NPath complexity	NPathComplexity	pmd	MAJOR	ACTIVE
Null Assignment	NullAssignment	pmd	MAJOR	ACTIVE
Only One Return	OnlyOneReturn	pmd	MINOR	ACTIVE
Optimizable To Array Call	OptimizableToArrayCall	pmd	MAJOR	ACTIVE
Override both equals and hashcode	OverrideBothEqualsAndHashcode	pmd	CRITICAL	ACTIVE
Package name	com.puppycrawl.tools.checkstyle.checks.naming.PackageNameCheck	checkstyle	MAJOR	ACTIVE

title	Key	plugin	priority	status
Parameter Name	com.puppycrawl.tools.checkstyle.checks.naming.ParameterNameCheck	checkstyle	MAJOR	ACTIVE
Performance - Could be refactored into a named static inner class	SIC_INNER_SHOULD_BE_STATIC_ANON	findbugs	MAJOR	ACTIVE
Performance - Could be refactored into a static inner class	SIC_INNER_SHOULD_BE_STATIC_NEEDS_THIS	findbugs	MAJOR	ACTIVE
Performance - Explicit garbage collection; extremely dubious except in benchmarking code	DM_GC	findbugs	MAJOR	ACTIVE
Performance - Huge string constants is duplicated across multiple class files	HSC_HUGE_SHARED_STRING_CONSTANT	findbugs	CRITICAL	ACTIVE
Performance - Inefficient use of keySet iterator instead of entrySet iterator	WMI_WRONG_MAP_ITERATOR	findbugs	CRITICAL	ACTIVE
Performance - Maps and sets of URLs can be performance hogs	DMI_COLLECTION_OF_URLS	findbugs	BLOCKER	ACTIVE
Performance - Method allocates a boxed primitive just to call toString	DM_BOXED_PRIMITIVE_TOSTRING	findbugs	MAJOR	ACTIVE
Performance - Method allocates an object, only to get the class object	DM_NEW_FOR_GETCLASS	findbugs	MAJOR	ACTIVE
Performance - Method calls static Math class method on a constant value	UM_UNNECESSARY_MATH	findbugs	CRITICAL	ACTIVE
Performance - Method concatenates strings using + in a loop	SBSC_USE_STRINGBUFFER_CONCATENATION	findbugs	CRITICAL	ACTIVE
Performance - Method invokes inefficient floating-point Number constructor; use static valueOf instead	DM_FP_NUMBER_CTOR	findbugs	MAJOR	ACTIVE
Performance - Method invokes inefficient new String(String) constructor	DM_STRING_CTOR	findbugs	MAJOR	ACTIVE
Performance - Method invokes toString() method on a String	DM_STRING_TOSTRING	findbugs	INFO	ACTIVE

title	Key	plugin	priority	status
Performance - Method uses toArray() with zero-length array argument	ITA_INEFFICIENT_TO_ARRAY	findbugs	CRITICAL	ACTIVE
Performance - Primitive value is boxed and then immediately unboxed	BX_BOXING_IMMEDIATELY_UNBOXED	findbugs	MAJOR	ACTIVE
Performance - Primitive value is boxed then unboxed to perform primitive coercion	BX_BOXING_IMMEDIATELY_UNBOXED_TO_PERFORM_COERCION	findbugs	MAJOR	ACTIVE
Performance - Should be a static inner class	SIC_INNER_SHOULD_BE_STATIC	findbugs	MAJOR	ACTIVE
Performance - The equals and hashCode methods of URL are blocking	DMI_BLOCKING_METHODS_ON_URL	findbugs	BLOCKER	ACTIVE
Performance - Unread field	URF_UNREAD_FIELD	findbugs	MAJOR	ACTIVE
Performance - Unread field: should this field be static?	SS_SHOULD_BE_STATIC	findbugs	MAJOR	ACTIVE
Performance - Unused field	UUF_UNUSED_FIELD	findbugs	MAJOR	ACTIVE
Performance - Use the nextInt method of Random rather than nextDouble to generate a random integer	DM_NEXTINT_VIA_NEXTDOUBLE	findbugs	MAJOR	ACTIVE
Position Literals First In Comparisons	PositionLiteralsFirstInComparisons	pmd	MAJOR	ACTIVE
Preserve Stack Trace	PreserveStackTrace	pmd	MAJOR	ACTIVE
Proper clone implementation	ProperCloneImplementation	pmd	CRITICAL	ACTIVE
Proper Logger	ProperLogger	pmd	MAJOR	ACTIVE
Redundant Modifier	com.puppycrawl.tools.checkstyle.checks.modifier.RedundantModifierCheck	checkstyle	MINOR	ACTIVE
Redundant Throws	com.puppycrawl.tools.checkstyle.checks.coding.RedundantThrowsCheck	checkstyle	MINOR	ACTIVE
Remote Interface Naming Convention	RemoteInterfaceNamingConvention	pmd	MAJOR	ACTIVE
Remote Session Interface Naming Convention	RemoteSessionInterfaceNamingConvention	pmd	MAJOR	ACTIVE
Replace Enumeration With Iterator	ReplaceEnumerationWithIterator	pmd	MAJOR	ACTIVE
Replace Hashtable With Map	ReplaceHashtableWithMap	pmd	MAJOR	ACTIVE
Replace Vector With List	ReplaceVectorWithList	pmd	MAJOR	ACTIVE
Return empty array rather	ReturnEmptyArrayRatherThan	pmd	MINOR	ACTIVE

title	Key	plugin	priority	status
than null	Null			
Return From Finally Block	ReturnFromFinallyBlock	pmd	MAJOR	ACTIVE
Security - A prepared statement is generated from a nonconstant String	SQL_PREPARED_STATEMENT_GENERATED_FROM_NONCONSTANT_STRING	findbugs	CRITICAL	ACTIVE
Security - Empty database password	DMI_EMPTY_DB_PASSWORD	findbugs	CRITICAL	ACTIVE
Security - Hardcoded constant database password	DMI_CONSTANT_DB_PASSWORD	findbugs	BLOCKER	ACTIVE
Security - HTTP cookie formed from untrusted input	HRS_REQUEST_PARAMETER_TO_COOKIE	findbugs	MAJOR	ACTIVE
Security - HTTP Response splitting vulnerability	HRS_REQUEST_PARAMETER_TO_HTTP_HEADER	findbugs	MAJOR	ACTIVE
Security - JSP reflected cross site scripting vulnerability	XSS_REQUEST_PARAMETER_TO_JSP_WRITER	findbugs	CRITICAL	ACTIVE
Security - Nonconstant string passed to execute method on an SQL statement	SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE	findbugs	CRITICAL	ACTIVE
Security - Servlet reflected cross site scripting vulnerability	XSS_REQUEST_PARAMETER_TO_SEND_ERROR	findbugs	CRITICAL	ACTIVE
Security - Servlet reflected cross site scripting vulnerability	XSS_REQUEST_PARAMETER_TO_SERVLET_WRITER	findbugs	CRITICAL	ACTIVE
Signature Declare Throws Exception	SignatureDeclareThrowsException	pmd	MAJOR	ACTIVE
Simple Date Format Needs Locale	SimpleDateFormatNeedsLocale	pmd	MAJOR	ACTIVE
Simplify Boolean Expression	com.puppycrawl.tools.checkstyle.checks.coding.SimplifyBooleanExpressionCheck	checkstyle	MAJOR	ACTIVE
Simplify Boolean Return	com.puppycrawl.tools.checkstyle.checks.coding.SimplifyBooleanReturnCheck	checkstyle	MAJOR	ACTIVE
Simplify boolean returns	SimplifyBooleanReturns	pmd	MINOR	ACTIVE
Simplify Conditional	SimplifyConditional	pmd	MAJOR	ACTIVE
Simplify Starts With	SimplifyStartsWith	pmd	MINOR	ACTIVE
Singular Field	SingularField	pmd	MINOR	ACTIVE
Static EJB Field Should Be Final	StaticEJBFieldShouldBeFinal	pmd	MAJOR	ACTIVE
Static Variable Name	com.puppycrawl.tools.checkstyle.checks.naming.StaticVaria	checkstyle	MAJOR	ACTIVE

title	Key	plugin	priority	status
	bleNameCheck			
Strict Exception - Do not throw exception in finally	DoNotThrowExceptionInFinally	pmd	MAJOR	ACTIVE
String Buffer Instantiation With Char	StringBufferInstantiationWithChar	pmd	MAJOR	ACTIVE
String Instantiation	StringInstantiation	pmd	MAJOR	ACTIVE
String Literal Equality	com.puppycrawl.tools.checkstyle.checks.coding.StringLiteralEqualityCheck	checkstyle	MAJOR	ACTIVE
String To String	StringToString	pmd	MAJOR	ACTIVE
Suspicious Octal Escape	SuspiciousOctalEscape	pmd	MAJOR	ACTIVE
Switch Density	SwitchDensity	pmd	MAJOR	ACTIVE
Switch statements should have default	SwitchStmtsShouldHaveDefault	pmd	MAJOR	ACTIVE
System Println	SystemPrintln	pmd	MAJOR	ACTIVE
Too few branches for a switch statement	TooFewBranchesForASwitchStatement	pmd	MINOR	ACTIVE
Too Many Fields	TooManyFields	pmd	MAJOR	ACTIVE
Too many methods	TooManyMethods	pmd	MAJOR	ACTIVE
Too Many Static Imports	TooManyStaticImports	pmd	MAJOR	ACTIVE
Type Name	com.puppycrawl.tools.checkstyle.checks.naming.TypeNameCheck	checkstyle	MAJOR	ACTIVE
Typecast Paren Pad	com.puppycrawl.tools.checkstyle.checks.whitespace.TypecastParenPadCheck	checkstyle	MAJOR	ACTIVE
Uncommented Empty Constructor	UncommentedEmptyConstructor	pmd	MAJOR	ACTIVE
Uncommented Empty Method	UncommentedEmptyMethod	pmd	MAJOR	ACTIVE
Unconditional If Statement	UnconditionalIfStatement	pmd	CRITICAL	ACTIVE
Unnecessary Case Change	UnnecessaryCaseChange	pmd	MINOR	ACTIVE
Unnecessary constructor	UnnecessaryConstructor	pmd	MAJOR	ACTIVE
Unnecessary Conversion Temporary	UnnecessaryConversionTemporary	pmd	MAJOR	ACTIVE
Unnecessary Final Modifier	UnnecessaryFinalModifier	pmd	INFO	ACTIVE
Unnecessary Local Before Return	UnnecessaryLocalBeforeReturn	pmd	MAJOR	ACTIVE
Unnecessary parentheses	UnnecessaryParentheses	pmd	MINOR	ACTIVE
Unnecessary Return	UnnecessaryReturn	pmd	MINOR	ACTIVE

title	Key	plugin	priority	status
Unnecessary Wrapper Object Creation	UnnecessaryWrapperObjectCreation	pmd	MAJOR	ACTIVE
Unsynchronized Static Date Formatter	UnsynchronizedStaticDateFormatter	pmd	MAJOR	ACTIVE
Unused formal parameter	UnusedFormalParameter	pmd	MAJOR	ACTIVE
Unused Imports	com.puppycrawl.tools.checkstyle.checks.imports.UnusedImportsCheck	checkstyle	INFO	ACTIVE
Unused local variable	UnusedLocalVariable	pmd	MAJOR	ACTIVE
Unused Modifier	UnusedModifier	pmd	INFO	ACTIVE
Unused Null Check In Equals	UnusedNullCheckInEquals	pmd	MAJOR	ACTIVE
Unused Private Field	UnusedPrivateField	pmd	MAJOR	ACTIVE
Unused private method	UnusedPrivateMethod	squid	MAJOR	ACTIVE
Unused protected method	UnusedProtectedMethod	squid	MAJOR	ACTIVE
Use Array List Instead Of Vector	UseArrayListInsteadOfVector	pmd	MAJOR	ACTIVE
Use Arrays As List	UseArraysAsList	pmd	MAJOR	ACTIVE
Use Collection Is Empty	UseCollectionIsEmpty	pmd	MINOR	ACTIVE
Use Correct Exception Logging	UseCorrectExceptionLogging	pmd	MAJOR	ACTIVE
Use Equals To Compare Strings	UseEqualsToCompareStrings	pmd	MAJOR	ACTIVE
Use Index Of Char	UseIndexOfChar	pmd	MAJOR	ACTIVE
Use Locale With Case Conversions	UseLocaleWithCaseConversions	pmd	MAJOR	ACTIVE
Use Notify All Instead Of Notify	UseNotifyAllInsteadOfNotify	pmd	MAJOR	ACTIVE
Use Proper Class Loader	UseProperClassLoader	pmd	CRITICAL	ACTIVE
Use Singleton	UseSingleton	pmd	MAJOR	ACTIVE
Use String Buffer For String Appends	UseStringBufferForStringAppends	pmd	MAJOR	ACTIVE
Use String Buffer Length	UseStringBufferLength	pmd	MINOR	ACTIVE
Useless Operation On Immutable	UselessOperationOnImmutable	pmd	CRITICAL	ACTIVE
Useless Overriding Method	UselessOverridingMethod	pmd	MAJOR	ACTIVE
Useless String Value Of	UselessStringValueOf	pmd	MINOR	ACTIVE
Visibility Modifier	com.puppycrawl.tools.checkstyle.checks.design.VisibilityModifierCheck	checkstyle	MAJOR	ACTIVE
While Loops Must Use Braces	WhileLoopsMustUseBraces	pmd	MAJOR	ACTIVE

APPENDIX B

TECHNICAL DEBT SCORES REPORTED BY SONARQUBE

PEAG Technical Debt Scores

Injected	DP	Version 1 Run	1	2	3	4	5
10	DP	Run	1	2	3	4	5
	clean_decorator		3.2	3.1	3.2	3.2	3.1
	clean_factory		3.6	3.6	3.6	3.6	3.6
	clean_observer		3.9	3.9	3.9	3.9	3.9
50	DP	Run	1	2	3	4	5
	clean_decorator		3.2	3.2	3.5	3.2	3.7
	clean_factory		3.6	3.6	3.6	3.6	3.6
	clean_observer		4.5	4.2	4.5	4.2	4.2
100	DP	Run	1	2	3	4	5
	clean_decorator		3.8	4.1	3.8	4.1	3.8
	clean_factory		3.7	3.7	3.7	3.7	3.7
	clean_observer		5.1	5	5.1	5.1	4.8

Injected	DP	Version 2 Run	1	2	3	4	5
20	DP	Run	1	2	3	4	5
	clean_decorator		3.2	3.2	3.2	3.2	3.2
	clean_factory		3.6	3.5	3.5	3.5	3.5
	clean_observer		3.9	3.9	3.9	3.9	3.9
100	DP	Run	1	2	3	4	5
	clean_decorator		3.8	4.1	3.8	3.8	4.1
	clean_factory		3.7	3.7	3.7	3.7	3.7
	clean_observer		4.6	5.1	4.8	5	5.1
200	DP	Run	1	2	3	4	5
	clean_decorator		4.3	4.3	4.3	4.3	4.3
	clean_factory		3.9	3.9	3.9	3.9	3.9
	clean_observer		5.2	5.2	5.2	5.2	5.2

Injected	DP	Version 3 Run	1	2	3	4	5
30	DP	Run	1	2	3	4	5
	clean_decorator		3.2	3.2	3.2	3.2	3.2
	clean_factory		3.6	3.6	3.6	3.6	3.6
	clean_observer		3.9	3.9	3.9	4.2	3.9
150	DP	Run	1	2	3	4	5
	clean_decorator		4.2	4.2	4.2	4.2	4.2
	clean_factory		3.8	3.8	3.8	3.8	3.8
	clean_observer		5.1	5.1	5.2	5.1	5.1
300	DP	Run	1	2	3	4	5
	clean_decorator		4.5	4.5	4.5	4.5	4.5
	clean_factory		4.1	4.1	4.1	4.1	4.1
	clean_observer		5.4	5.4	5.4	5.4	5.4

PEEG Technical Debt Scores

	Version 1						
Injected							
10	DP	Run	1	2	3	4	5
	clean_decorator		3.1	3.2	3.1	3.1	3.2
	clean_factory		3.5	3.5	3.5	3.5	3.6
	clean_observer		4	3.9	4	3.9	3.9
50	DP	Run	1	2	3	4	5
	clean_decorator		3.2	3.5	3.2	3.5	3.2
	clean_factory		3.6	3.6	3.6	3.6	3.6
	clean_observer		4.2	4.2	4.2	4.2	4.2
100	DP	Run	1	2	3	4	5
	clean_decorator		3.6	3.8	3.8	4.1	3.8
	clean_factory		3.7	3.7	3.7	3.7	3.7
	clean_observer		5.1	4.8	4.8	4.8	5

	Version 2						
Injected							
20	DP	Run	1	2	3	4	5
	clean_decorator		3.2	3.2	3.2	3.2	3.2
	clean_factory		3.5	3.5	3.5	3.5	3.6
	clean_observer		4	3.9	3.9	3.9	3.9
100	DP	Run	1	2	3	4	5
	clean_decorator		3.8	3.8	4.1	3.8	4.1
	clean_factory		3.7	3.7	3.7	3.7	3.7
	clean_observer		4.8	4.8	5	5	5.1
200	DP	Run	1	2	3	4	5
	clean_decorator		4.3	4.3	4.3	4.3	4.3
	clean_factory		3.9	3.9	3.9	3.9	3.9
	clean_observer		5.2	5.2	5.2	5.2	5.2

	Version 3						
Injected							
30	DP	Run	1	2	3	4	5
	clean_decorator		3.2	3.2	3.2	3.2	3.2
	clean_factory		3.6	3.6	3.6	3.6	3.6
	clean_observer		3.9	3.9	3.9	3.9	3.9
150	DP	Run	1	2	3	4	5
	clean_decorator		4.2	4.2	4.2	4.2	4.2
	clean_factory		3.8	3.8	3.8	3.8	3.8
	clean_observer		5.1	5.1	5.1	5.1	5.1
300	DP	Run	1	2	3	4	5
	clean_decorator		4.5	4.5	4.5	4.5	4.5
	clean_factory		4.1	4.1	4.1	4.1	4.1
	clean_observer		5.4	5.4	5.4	5.4	5.4

PIG Technical Debt Scores

	Version 1							Injected	Version 2							Injected	Version 3						
Injected																							
10	**DP**	**Run**		1	2	3	4	5	20	**DP**	**Run**	1	2	3	4	5	30	**DP**	**Run**	1	2	3	4	5
	clean_decorator		3.1	3.1	3.1	3.1	3.1		**clean_decorator**		3.2	3.2	3.2	3.2	3.2		**clean_decorator**		3.2	3.2	3.2	3.2	3.2
	clean_factory		3.6	3.5	3.5	3.5	3.6		**clean_factory**		3.6	3.5	3.5	3.5	3.5		**clean_factory**		3.6	3.6	3.6	3.6	3.6
	clean_observer		3.9	3.9	3.9	3.9	3.9		**clean_observer**		3.9	3.9	3.9	3.9	3.9		**clean_observer**		3.9	3.9	3.9	3.9	4.2
50	**DP**	**Run**	1	2	3	4	5	100	**DP**	**Run**	1	2	3	4	5	150	**DP**	**Run**	1	2	3	4	5
	clean_decorator		3.5	3.2	3.2	3.5	3.2		**clean_decorator**		3.8	4.1	4.1	3.8	3.8		**clean_decorator**		4.2	4.2	4.2	4.2	4.2
	clean_factory		3.6	3.6	3.6	3.6	3.6		**clean_factory**		3.7	3.7	3.7	3.7	3.7		**clean_factory**		3.8	3.8	3.8	3.8	3.8
	clean_observer		4.2	4.2	4.2	4	4.5		**clean_observer**		5.1	5.1	5.1	4.8	5.1		**clean_observer**		5.1	5.1	5.1	5.1	5.2
100	**DP**	**Run**	1	2	3	4	5	200	**DP**	**Run**	1	2	3	4	5	300	**DP**	**Run**	1	2	3	4	5
	clean_decorator		4.1	4.1	4.1	4.1	4.1		**clean_decorator**		4.3	4.3	4.3	4.3	4.3		**clean_decorator**		4.5	4.5	4.5	4.5	4.5
	clean_factory		3.7	3.7	3.7	3.7	3.7		**clean_factory**		3.9	3.9	3.9	3.9	3.9		**clean_factory**		4.1	4.1	4.1	4.1	4.1
	clean_observer		5.1	4.8	5	5.1	4.8		**clean_observer**		5.2	5.3	5.2	5.2	5.2		**clean_observer**		5.4	5.4	5.4	5.4	5.4

TEAG Technical Debt Scores

Version 1						
Injected						
10	DP / Run	1	2	3	4	5
	clean_decorator	3.3	3.3	3.4	3.3	3.3
	clean_factory	3.8	3.8	3.7	3.7	3.8
	clean_observer	4.1	4.1	4.1	4.1	4.1
50	DP / Run	1	2	3	4	5
	clean_decorator	4.3	4.3	4.5	4.3	4.3
	clean_factory	4.6	4.6	4.6	4.6	4.6
	clean_observer	5.4	5.3	5.5	5.5	5.7
100	DP / Run	1	2	3	4	5
	clean_decorator	6.5	6.2	5.9	5.7	6.2
	clean_factory	5.8	5.8	5.8	5.8	5.8
	clean_observer	7.6	7.8	7.7	7.7	7.6

Version 2						
Injected						
20	DP / Run	1	2	3	4	5
	clean_decorator	3.6	3.6	3.6	3.6	3.6
	clean_factory	4	3.9	3.9	3.9	4
	clean_observer	4.5	4.5	4.5	4.3	4.4
100	DP / Run	1	2	3	4	5
	clean_decorator	5.7	5.9	6.2	5.7	5.9
	clean_factory	5.8	5.8	5.8	5.8	5.8
	clean_observer	7.7	7.7	7.6	7.6	7.6
200	DP / Run	1	2	3	4	5
	clean_decorator	8.4	8.4	8.4	8.4	8.4
	clean_factory	8.1	8.1	8.1	8.1	8.1
	clean_observer	10.4	10.6	10.4	10.7	10.5

Version 3						
Injected						
30	DP / Run	1	2	3	4	5
	clean_decorator	3.8	3.8	3.8	3.8	3.8
	clean_factory	4.2	4.2	4.2	4.2	4.2
	clean_observer	4.8	4.7	5.1	4.8	4.5
150	DP / Run	1	2	3	4	5
	clean_decorator	7.3	7.3	7.3	7.3	7.3
	clean_factory	6.9	6.9	6.9	6.9	6.9
	clean_observer	9.2	9.1	8.8	9	9
300	DP / Run	1	2	3	4	5
	clean_decorator	10.7	10.7	10.7	10.7	10.7
	clean_factory	10.4	10.4	10.4	10.4	10.4
	clean_observer	13.2	13.5	13.3	13.4	13.4

TEEG Technical Debt Scores

Version 1							
Injected							
10	DP	Run	1	2	3	4	5
	clean_decorator		3.4	3.4	3.3	3.4	3.3
	clean_factory		3.7	3.8	3.8	3.7	3.8
	clean_observer		4.3	4.2	4.1	4.1	4.1
50	DP	Run	1	2	3	4	5
	clean_decorator		4.5	4.3	4.3	4.5	4.3
	clean_factory		4.6	4.6	4.6	4.6	4.6
	clean_observer		5.5	5.6	5.5	5.8	5.8
100	DP	Run	1	2	3	4	5
	clean_decorator		5.9	5.9	6.2	5.9	6.2
	clean_factory		5.8	5.8	5.8	5.8	5.8
	clean_observer		7.4	7.6	7.9	7.6	7.9

Version 2							
Injected							
20	DP	Run	1	2	3	4	5
	clean_decorator		3.6	3.6	3.6	3.6	3.6
	clean_factory		3.9	3.9	3.9	3.9	4
	clean_observer		4.6	4.6	4.5	4.5	4.4
100	DP	Run	1	2	3	4	5
	clean_decorator		5.7	6.2	5.9	6.2	6.2
	clean_factory		5.8	5.8	5.8	5.8	5.8
	clean_observer		8	7.8	7.8	7.6	7.8
200	DP	Run	1	2	3	4	5
	clean_decorator		8.4	8.4	8.4	8.4	8.4
	clean_factory		8.1	8.1	8.1	8.1	8.1
	clean_observer		10.5	10.6	10.6	10.6	10.5

Version 3							
Injected							
30	DP	Run	1	2	3	4	5
	clean_decorator		3.8	3.8	3.8	3.8	3.8
	clean_factory		4.2	4.2	4.2	4.2	4.2
	clean_observer		4.9	4.8	4.8	4.8	4.6
150	DP	Run	1	2	3	4	5
	clean_decorator		7.3	7.3	7.3	7.3	4.3
	clean_factory		6.9	6.9	6.9	6.9	6.9
	clean_observer		9.7	9.1	9.2	9.3	9.1
300	DP	Run	1	2	3	4	5
	clean_decorator		10.7	10.7	10.7	10.7	10.7
	clean_factory		10.4	10.4	10.4	10.4	10.4
	clean_observer		13.3	13.3	13.6	13.5	13.4

TIG Technical Debt Scores

Injected	DP	Run	1	2	3	4	5
		Version 1					
10	**DP**	**Run**	1	2	3	4	5
	clean_decorator		3.3	3.3	3.4	3.3	3.4
	clean_factory		3.7	3.8	3.8	3.8	3.8
	clean_observer		4.1	4.1	4.3	4.2	4.1
50	**DP**	**Run**	1	2	3	4	5
	clean_decorator		4.3	4.3	4.3	4.3	4.3
	clean_factory		4.6	4.6	4.6	4.6	4.6
	clean_observer		5.5	5.6	5.9	5.4	5.4
100	**DP**	**Run**	1	2	3	4	5
	clean_decorator		5.7	6.2	6.2	5.9	6.2
	clean_factory		5.8	5.8	5.8	5.8	5.8
	clean_observer		7.7	7.9	7.7	7.8	8

Injected	DP	Run	1	2	3	4	5
		Version 2					
20	**DP**	**Run**	1	2	3	4	5
	clean_decorator		3.6	3.6	3.6	3.6	3.6
	clean_factory		3.9	4	4	3.9	4
	clean_observer		4.5	4.4	4.6	4.5	4.5
100	**DP**	**Run**	1	2	3	4	5
	clean_decorator		5.9	5.9	5.9	5.9	5.7
	clean_factory		5.8	5.8	5.8	5.8	5.8
	clean_observer		7.6	7.8	7.5	7.8	7.8
200	**DP**	**Run**	1	2	3	4	5
	clean_decorator		8.4	8.4	8.4	8.4	8.4
	clean_factory		8.1	8.1	8.1	8.1	8.1
	clean_observer		10.4	10.8	10.6	10.8	11

Injected	DP	Run	1	2	3	4	5
		Version 3					
30	**DP**	**Run**	1	2	3	4	5
	clean_decorator		3.8	3.8	3.9	3.8	3.8
	clean_factory		4.2	4.2	4.2	4.2	4.2
	clean_observer		4.7	4.8	4.9	4.7	4.8
150	**DP**	**Run**	1	2	3	4	5
	clean_decorator		7.3	7.1	7.3	7.3	7.3
	clean_factory		6.9	6.9	6.9	6.9	6.9
	clean_observer		9.1	9.4	9.2	9.3	9.3
300	**DP**	**Run**	1	2	3	4	5
	clean_decorator		10.7	10.7	10.7	10.7	10.7
	clean_factory		10.4	10.4	10.4	10.4	10.4
	clean_observer		13.3	13.7	13.3	13.7	13.7

APPENDIX C

SAS RESULTS

10 instances of Modular Grime

The GLM Procedure

Class Level Information		
Class	**Levels**	**Values**
GrimeType	6	PEAG PEEG PIG TEAG TEEG TIG
DPattern	3	Deco Fact Obse

Number of Observations Read	90
Number of Observations Used	90

10 instances of Modular Grime
The GLM Procedure

Dependent Variable: TehnicalDebt

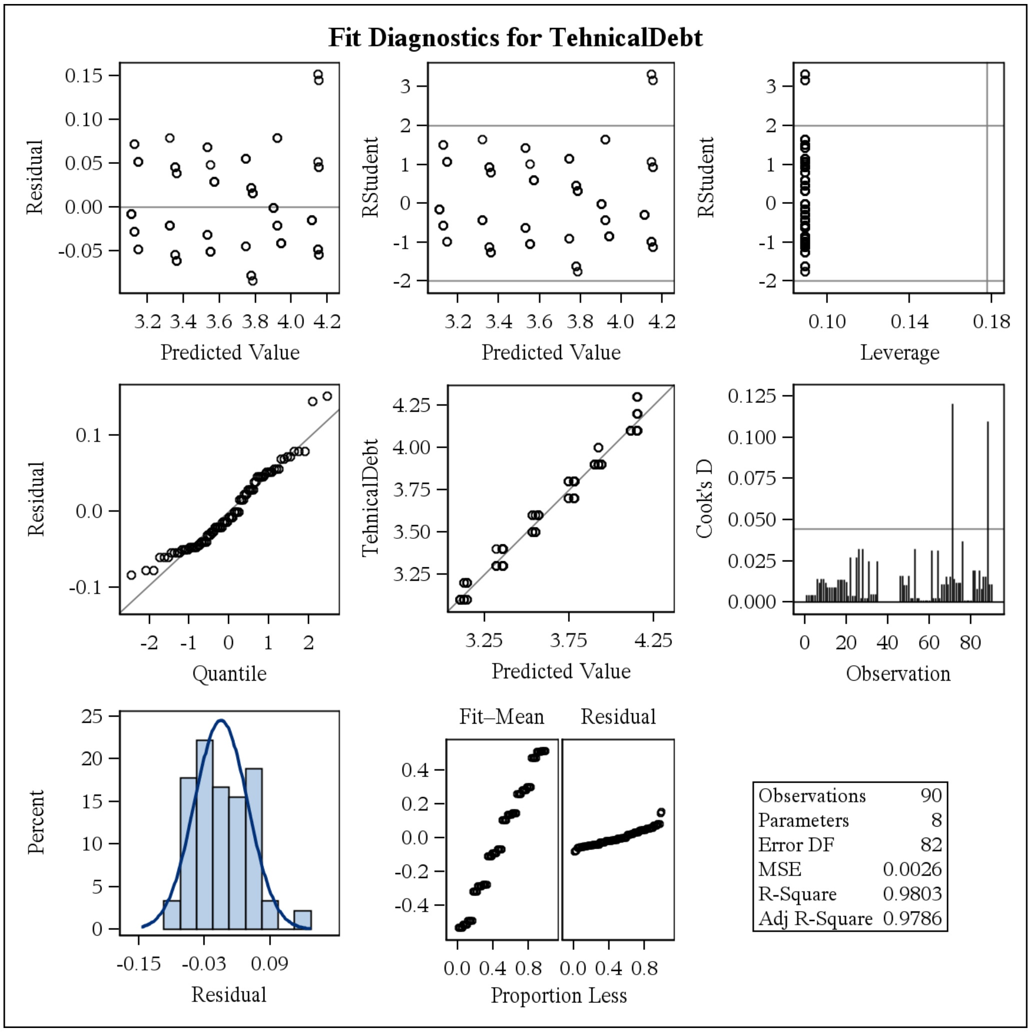
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	10.54777778	1.50682540	583.44	<.0001
Error	82	0.21177778	0.00258266		
Corrected Total	89	10.75955556			

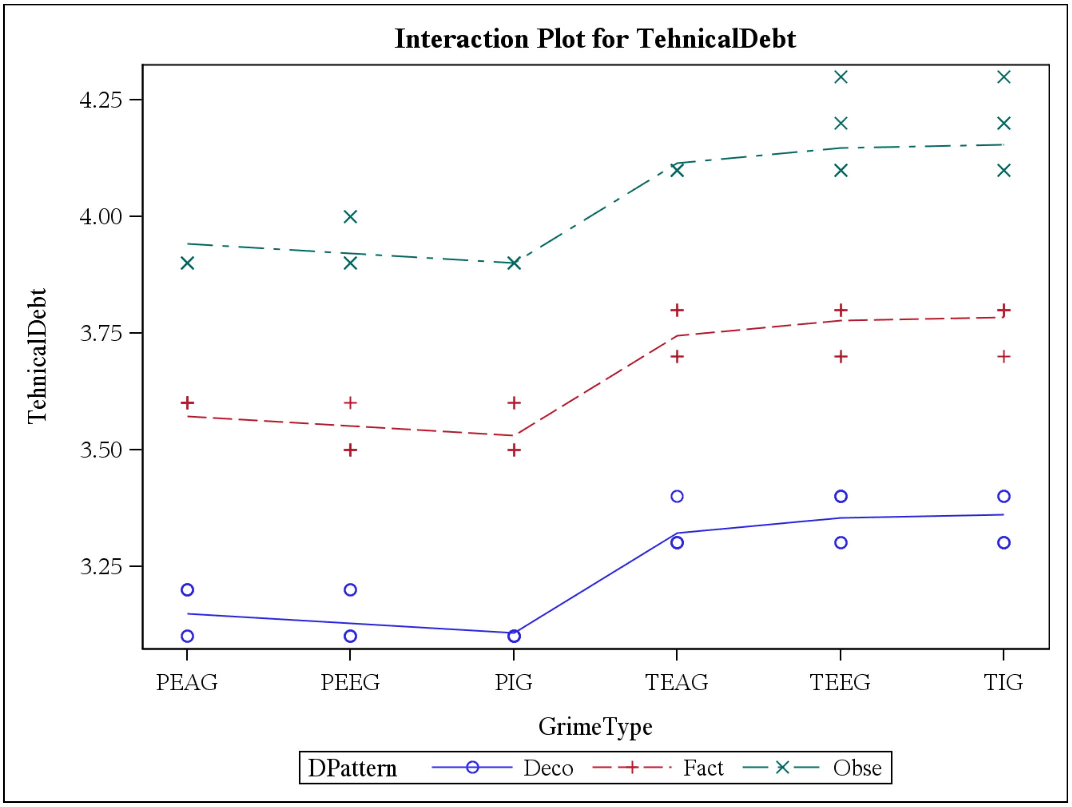
R-Square	Coeff Var	Root MSE	TehnicalDebt Mean
0.980317	1.395298	0.050820	3.642222

Source	DF	Type III SS	Mean Square	F Value	Pr > F
GrimeType	5	1.09288889	0.21857778	84.63	<.0001
DPattern	2	9.45488889	4.72744444	1830.46	<.0001

Parameter	Estimate		Standard Error	t Value	Pr > |t|
Intercept	4.15444	B	0.01515155	274.19	<.0001
GrimeType PEAG	-0.21333	B	0.01855678	-11.50	<.0001
GrimeType PEEG	-0.23333	B	0.01855678	-12.57	<.0001
GrimeType PIG	-0.25333	B	0.01855678	-13.65	<.0001
GrimeType TEAG	-0.04000	B	0.01855678	-2.16	0.0341
GrimeType TEEG	-0.00666	B	0.01855678	-0.36	0.7203
GrimeType TIG	0.00000	B	.	.	.
DPattern Deco	-0.79333	B	0.01312163	-60.46	<.0001
DPattern Fact	-0.37000	B	0.01312163	-28.20	<.0001
DPattern Obse	0.00000	B	.	.	.

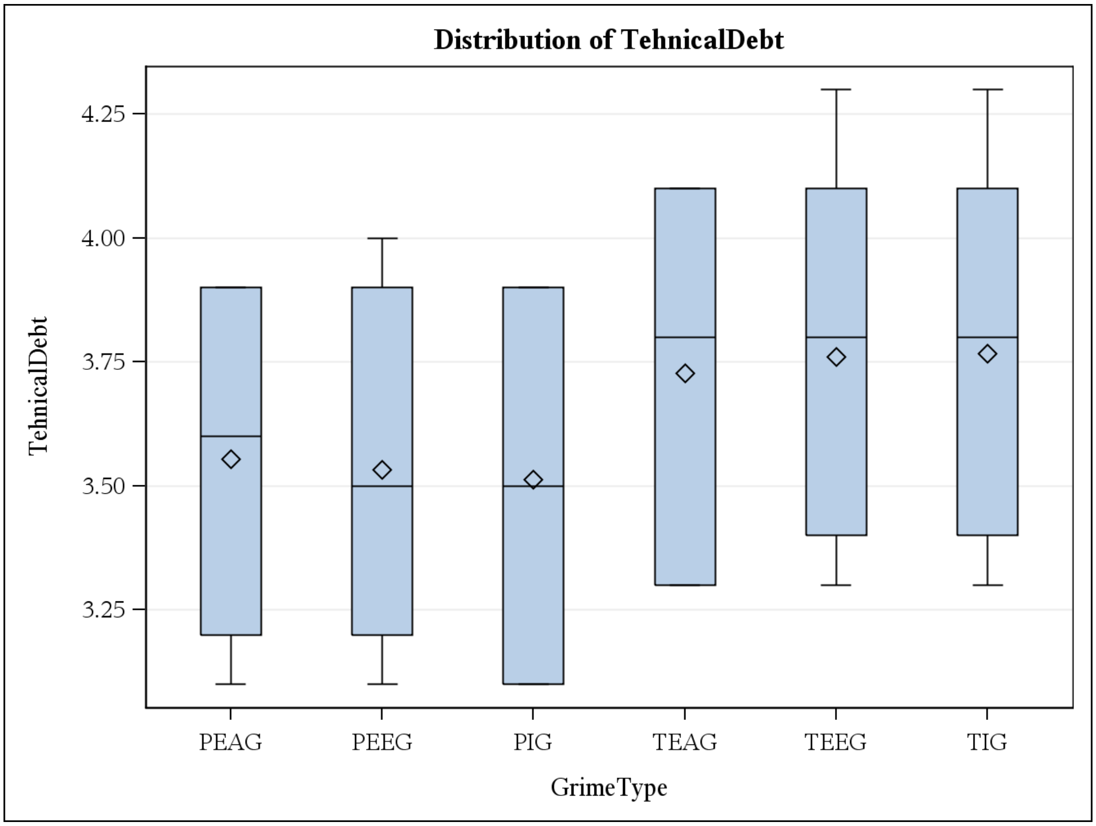
Note: The X'X matrix has been found to be singular, and a generalized inverse was used to solve the normal equations. Terms whose estimates are followed by the letter 'B' are not uniquely estimable.


Fit Diagnostics for TehnicalDebt
Residual
Predicted Value
RStudent
Predicted Value
RStudent
Leverage
Residual
Quantile
TehnicalDebt
Predicted Value
Cook's D
Observation
Percent
Residual
Fit–Mean
Residual
Proportion Less
Observations 90
Parameters 8
Error DF 82
MSE 0.0026
R-Square 0.9803
Adj R-Square 0.9786


Interaction Plot for TehnicalDebt
4.25
4.00
3.75
3.50
3.25
TehnicalDebt
PEAG
PEEG
PIG
TEAG
TEEG
TIG
GrimeType
DPattern
Deco
Fact
Obse

10 instances of Modular Grime

The GLM Procedure



10 instances of Modular Grime

The GLM Procedure

Tukey's Studentized Range (HSD) Test for TehnicalDebt

Note: This test controls the Type I experimentwise error rate.

Alpha	0.05
Error Degrees of Freedom	82
Error Mean Square	0.002583
Critical Value of Studentized Range	4.12696
Minimum Significant Difference	0.0542

Comparisons significant at the 0.05 level are indicated by ***.				
GrimeType Comparison	**Difference Between Means**	**Simultaneous 95% Confidence Limits**		
TIG - TEEG	0.00667	-0.04749	0.06082	
TIG - TEAG	0.04000	-0.01415	0.09415	
TIG - PEAG	0.21333	0.15918	0.26749	***
TIG - PEEG	0.23333	0.17918	0.28749	***
TIG - PIG	0.25333	0.19918	0.30749	***
TEEG - TIG	-0.00667	-0.06082	0.04749	
TEEG - TEAG	0.03333	-0.02082	0.08749	
TEEG - PEAG	0.20667	0.15251	0.26082	***
TEEG - PEEG	0.22667	0.17251	0.28082	***
TEEG - PIG	0.24667	0.19251	0.30082	***
TEAG - TIG	-0.04000	-0.09415	0.01415	
TEAG - TEEG	-0.03333	-0.08749	0.02082	
TEAG - PEAG	0.17333	0.11918	0.22749	***
TEAG - PEEG	0.19333	0.13918	0.24749	***
TEAG - PIG	0.21333	0.15918	0.26749	***
PEAG - TIG	-0.21333	-0.26749	-0.15918	***

Comparisons significant at the 0.05 level are indicated by ***.				
GrimeType Comparison	**Difference Between Means**	**Simultaneous 95% Confidence Limits**		
PEAG - TEEG	-0.20667	-0.26082	-0.15251	***
PEAG - TEAG	-0.17333	-0.22749	-0.11918	***
PEAG - PEEG	0.02000	-0.03415	0.07415	
PEAG - PIG	0.04000	-0.01415	0.09415	
PEEG - TIG	-0.23333	-0.28749	-0.17918	***
PEEG - TEEG	-0.22667	-0.28082	-0.17251	***
PEEG - TEAG	-0.19333	-0.24749	-0.13918	***
PEEG - PEAG	-0.02000	-0.07415	0.03415	
PEEG - PIG	0.02000	-0.03415	0.07415	
PIG - TIG	-0.25333	-0.30749	-0.19918	***
PIG - TEEG	-0.24667	-0.30082	-0.19251	***
PIG - TEAG	-0.21333	-0.26749	-0.15918	***
PIG - PEAG	-0.04000	-0.09415	0.01415	
PIG - PEEG	-0.02000	-0.07415	0.03415	

10 instances of Modular Grime

The GLM Procedure

Tukey's Studentized Range (HSD) Test for TehnicalDebt

Note: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	82
Error Mean Square	0.002583
Critical Value of Studentized Range	4.12696
Minimum Significant Difference	0.0542

Means with the same letter are not significantly different.			
Tukey Grouping	**Mean**	**N**	**GrimeType**
A	3.76667	15	TIG
A			
A	3.76000	15	TEEG
A			
A	3.72667	15	TEAG
B	3.55333	15	PEAG
B			
B	3.53333	15	PEEG
B			
B	3.51333	15	PIG

50 instances of Modular Grime

The GLM Procedure

Class Level Information		
Class	**Levels**	**Values**
GrimeType	6	PEAG PEEG PIG TEAG TEEG TIG
DPattern	3	Deco Fact Obse

Number of Observations Read	90
Number of Observations Used	90

50 instances of Modular Grime
The GLM Procedure
Dependent Variable: TehnicalDebt

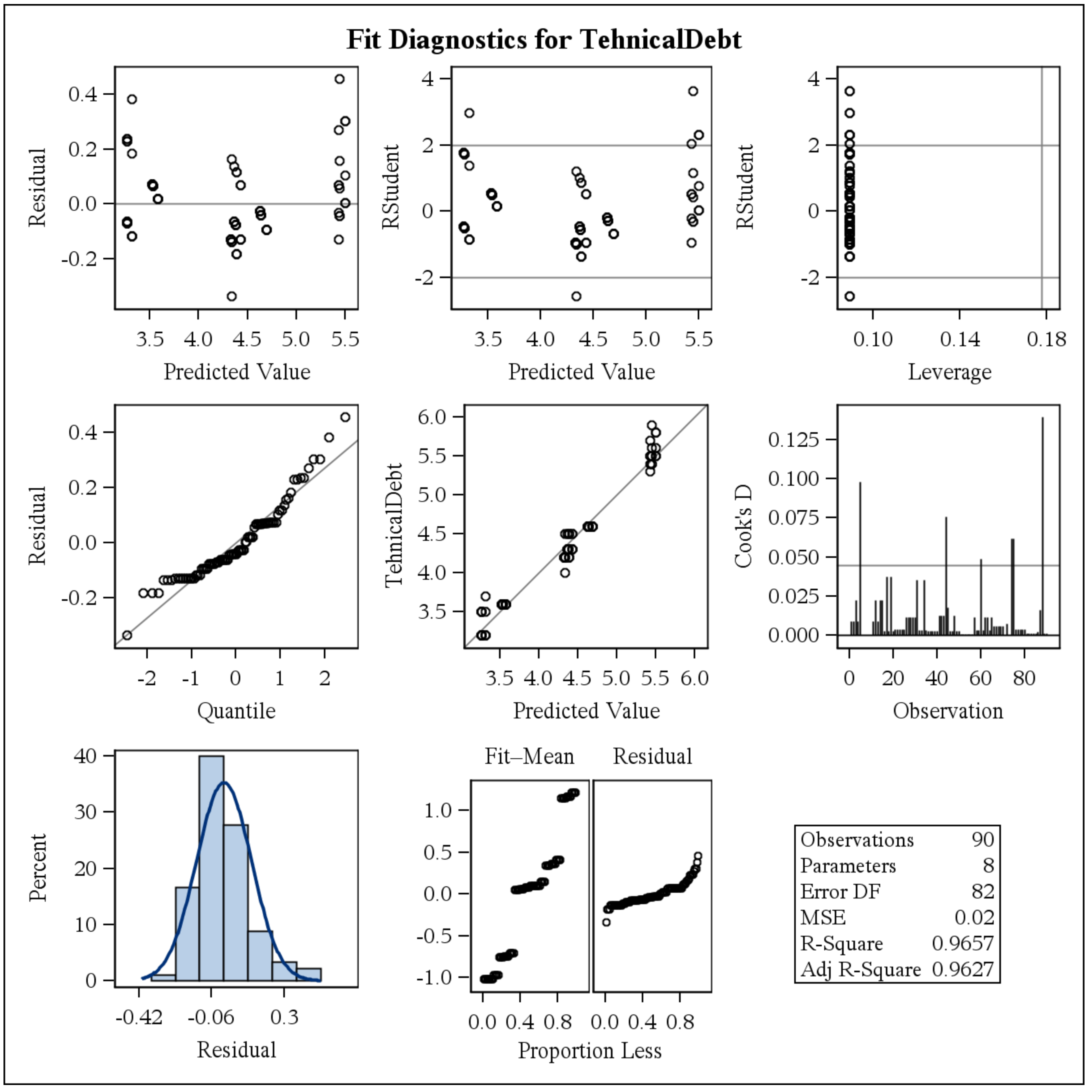
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	46.14333333	6.59190476	329.46	<.0001
Error	82	1.64066667	0.02000813		
Corrected Total	89	47.78400000			

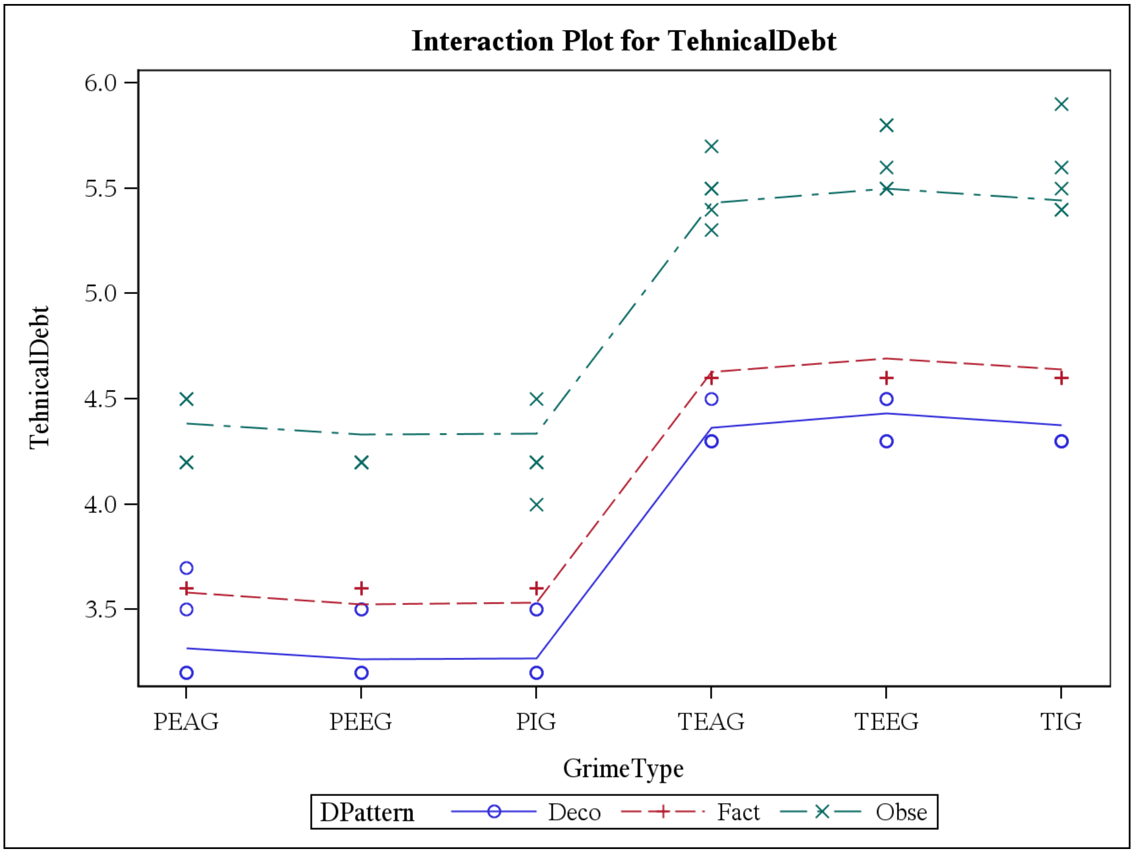
R-Square	Coeff Var	Root MSE	TehnicalDebt Mean
0.965665	3.304909	0.141450	4.280000

Source	DF	Type III SS	Mean Square	F Value	Pr > F
GrimeType	5	27.61866667	5.52373333	276.07	<.0001
DPattern	2	18.52466667	9.26233333	462.93	<.0001

Parameter	Estimate		Standard Error	t Value	Pr > |t|
Intercept	5.44333	B	0.04217227	129.07	<.0001
GrimeType PEAG	-1.06000	B	0.05165027	-20.52	<.0001
GrimeType PEEG	-1.11333	B	0.05165027	-21.56	<.0001
GrimeType PIG	-1.10666	B	0.05165027	-21.43	<.0001
GrimeType TEAG	-0.01333	B	0.05165027	-0.26	0.7969
GrimeType TEEG	0.05333	B	0.05165027	1.03	0.3048
GrimeType TIG	0.00000	B	.	.	.
DPattern Deco	-1.06666	B	0.03652226	-29.21	<.0001
DPattern Fact	-0.80333	B	0.03652226	-22.00	<.0001
DPattern Obse	0.00000	B	.	.	.

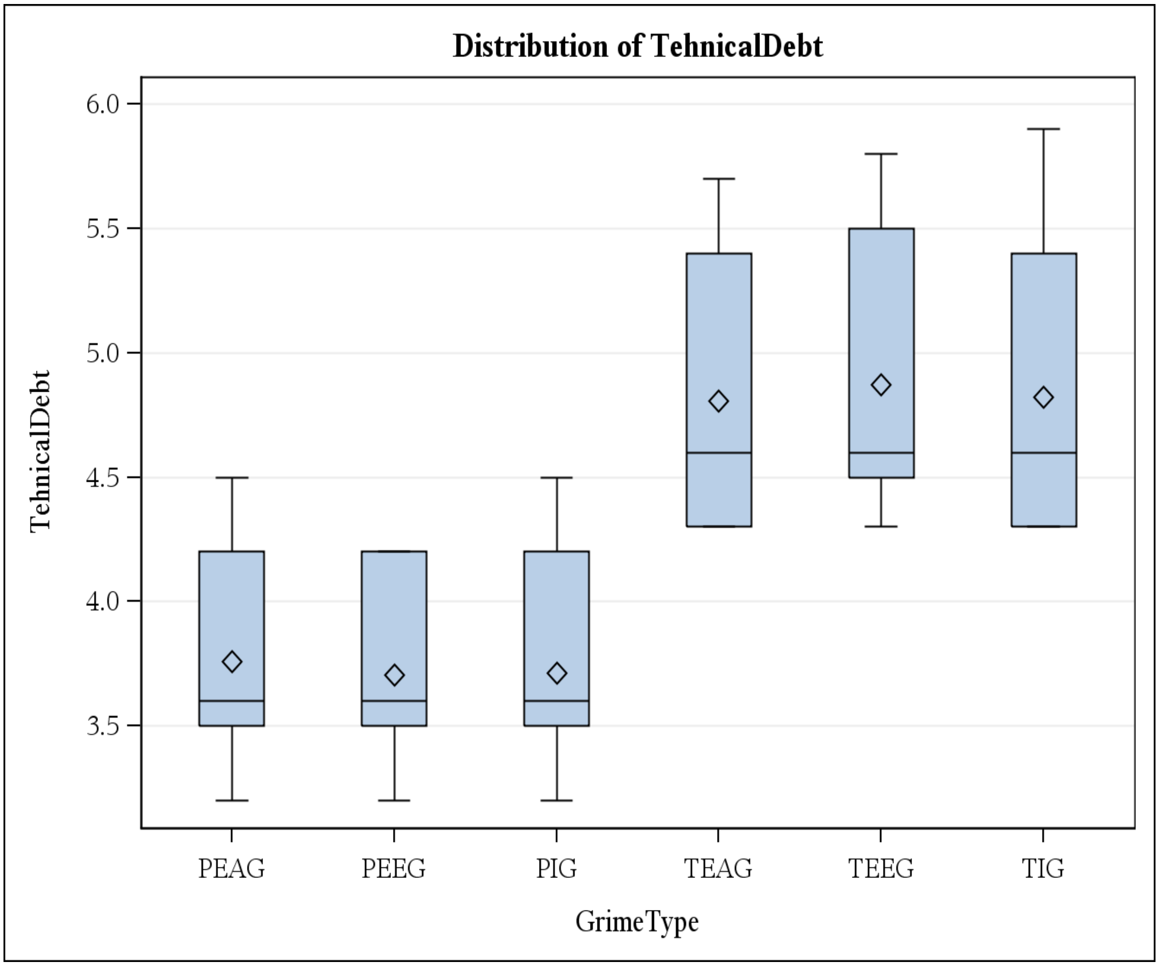
Note: The X'X matrix has been found to be singular, and a generalized inverse was used to solve the normal equations. Terms whose estimates are followed by the letter 'B' are not uniquely estimable.


Fit Diagnostics for TehnicalDebt
Residual
Predicted Value
RStudent
Predicted Value
RStudent
Leverage
Residual
Quantile
TehnicalDebt
Predicted Value
Cook's D
Observation
Percent
Residual
Fit–Mean
Residual
Proportion Less
Observations 90
Parameters 8
Error DF 82
MSE 0.02
R-Square 0.9657
Adj R-Square 0.9627


Interaction Plot for TehnicalDebt
TehnicalDebt
6.0
5.5
5.0
4.5
4.0
3.5
PEAG
PEEG
PIG
TEAG
TEEG
TIG
GrimeType
DPattern
Deco
Fact
Obse

50 instances of Modular Grime

The GLM Procedure



50 instances of Modular Grime

The GLM Procedure

Tukey's Studentized Range (HSD) Test for TehnicalDebt

Note: This test controls the Type I experimentwise error rate.

Alpha	0.05
Error Degrees of Freedom	82
Error Mean Square	0.020008
Critical Value of Studentized Range	4.12696
Minimum Significant Difference	0.1507

Comparisons significant at the 0.05 level are indicated by *.**				
GrimeType Comparison	**Difference Between Means**	**Simultaneous 95% Confidence Limits**		
TEEG - TIG	0.05333	-0.09739	0.20406	
TEEG - TEAG	0.06667	-0.08406	0.21739	
TEEG - PEAG	1.11333	0.96261	1.26406	***
TEEG - PIG	1.16000	1.00927	1.31073	***
TEEG - PEEG	1.16667	1.01594	1.31739	***
TIG - TEEG	-0.05333	-0.20406	0.09739	
TIG - TEAG	0.01333	-0.13739	0.16406	
TIG - PEAG	1.06000	0.90927	1.21073	***
TIG - PIG	1.10667	0.95594	1.25739	***
TIG - PEEG	1.11333	0.96261	1.26406	***
TEAG - TEEG	-0.06667	-0.21739	0.08406	
TEAG - TIG	-0.01333	-0.16406	0.13739	
TEAG - PEAG	1.04667	0.89594	1.19739	***
TEAG - PIG	1.09333	0.94261	1.24406	***
TEAG - PEEG	1.10000	0.94927	1.25073	***
PEAG - TEEG	-1.11333	-1.26406	-0.96261	***

Comparisons significant at the 0.05 level are indicated by ***.				
GrimeType Comparison	**Difference Between Means**	**Simultaneous 95% Confidence Limits**		
PEAG - TIG	-1.06000	-1.21073	-0.90927	***
PEAG - TEAG	-1.04667	-1.19739	-0.89594	***
PEAG - PIG	0.04667	-0.10406	0.19739	
PEAG - PEEG	0.05333	-0.09739	0.20406	
PIG - TEEG	-1.16000	-1.31073	-1.00927	***
PIG - TIG	-1.10667	-1.25739	-0.95594	***
PIG - TEAG	-1.09333	-1.24406	-0.94261	***
PIG - PEAG	-0.04667	-0.19739	0.10406	
PIG - PEEG	0.00667	-0.14406	0.15739	
PEEG - TEEG	-1.16667	-1.31739	-1.01594	***
PEEG - TIG	-1.11333	-1.26406	-0.96261	***
PEEG - TEAG	-1.10000	-1.25073	-0.94927	***
PEEG - PEAG	-0.05333	-0.20406	0.09739	
PEEG - PIG	-0.00667	-0.15739	0.14406	

50 instances of Modular Grime

The GLM Procedure

Tukey's Studentized Range (HSD) Test for TehnicalDebt

Note: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	82
Error Mean Square	0.020008
Critical Value of Studentized Range	4.12696
Minimum Significant Difference	0.1507

Means with the same letter are not significantly different.			
Tukey Grouping	**Mean**	**N**	**GrimeType**
A	4.87333	15	TEEG
A			
A	4.82000	15	TIG
A			
A	4.80667	15	TEAG
B	3.76000	15	PEAG
B			
B	3.71333	15	PIG
B			
B	3.70667	15	PEEG

100 instances of Modular Grime

The GLM Procedure

Class Level Information		
Class	**Levels**	**Values**
GrimeType	6	PEAG PEEG PIG TEAG TEEG TIG
DPattern	3	Deco Fact Obse

Number of Observations Read	90
Number of Observations Used	90

100 instances of Modular Grime
The GLM Procedure
Dependent Variable: TehnicalDebt

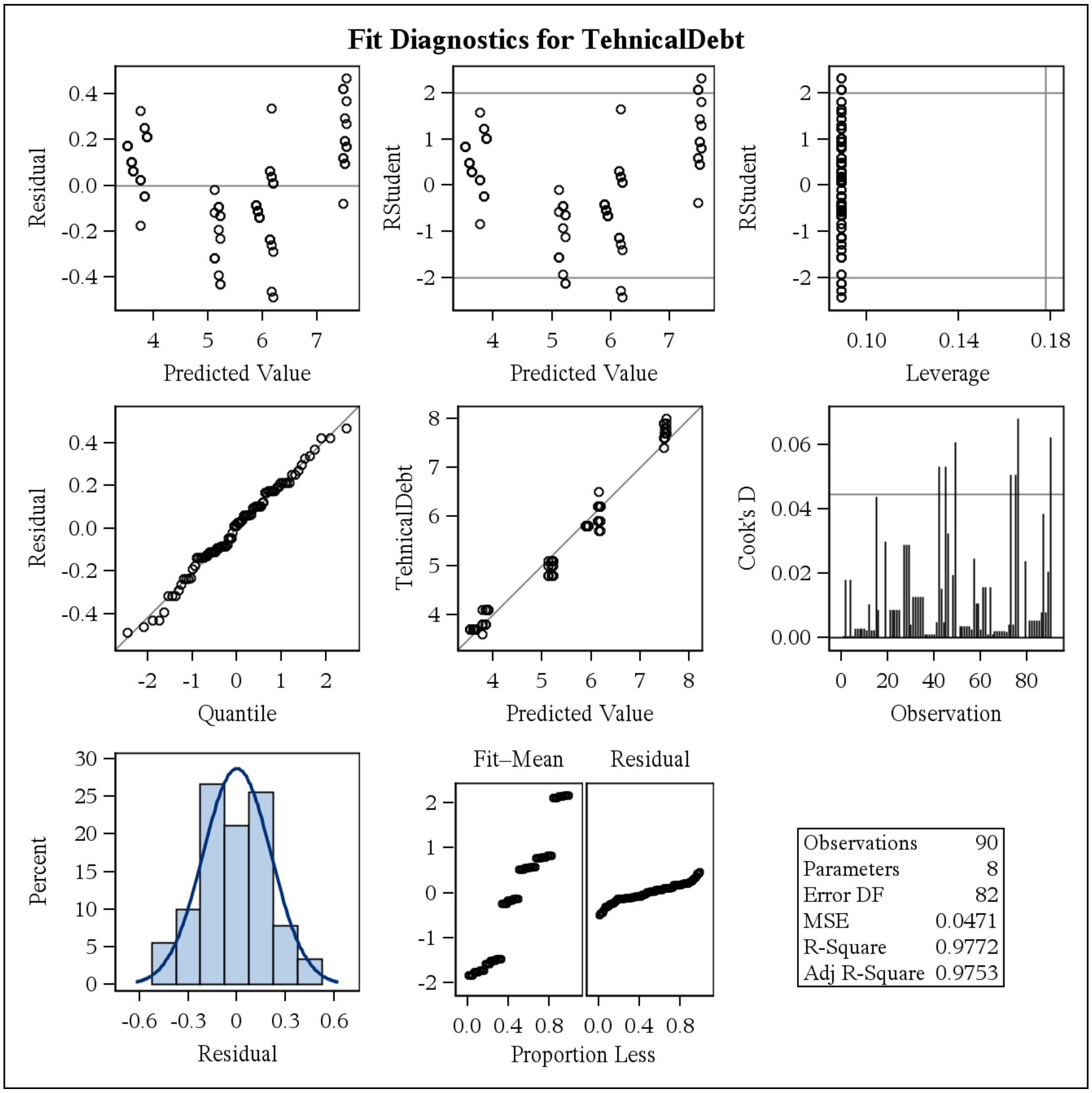
Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	7	165.7464444	23.6780635	503.03	<.0001
Error	82	3.8597778	0.0470705		
Corrected Total	89	169.6062222			

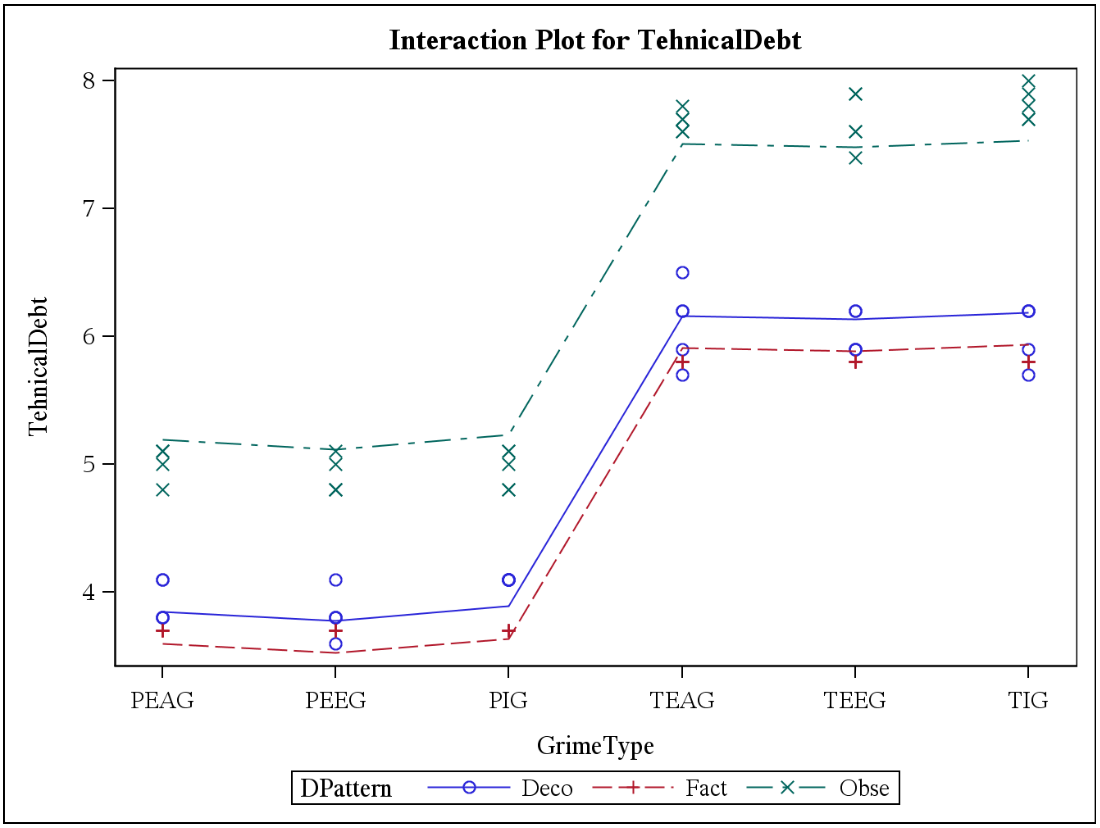
R-Square	Coeff Var	Root MSE	TehnicalDebt Mean
0.977243	4.044357	0.216957	5.364444

Source	DF	Type III SS	Mean Square	F Value	Pr > F
GrimeType	5	121.6888889	24.3377778	517.05	<.0001
DPattern	2	44.0575556	22.0287778	468.00	<.0001

Parameter	Estimate		Standard Error	t Value	Pr > |t|
Intercept	7.532222222	B	0.06468416	116.45	<.0001
GrimeType PEAG	-2.340000000	B	0.07922160	-29.54	<.0001
GrimeType PEEG	-2.413333333	B	0.07922160	-30.46	<.0001
GrimeType PIG	-2.300000000	B	0.07922160	-29.03	<.0001
GrimeType TEAG	-0.026666667	B	0.07922160	-0.34	0.7373
GrimeType TEEG	-0.053333333	B	0.07922160	-0.67	0.5027
GrimeType TIG	0.000000000	B	.	.	.
DPattern Deco	-1.343333333	B	0.05601813	-23.98	<.0001
DPattern Fact	-1.593333333	B	0.05601813	-28.44	<.0001
DPattern Obse	0.000000000	B	.	.	.

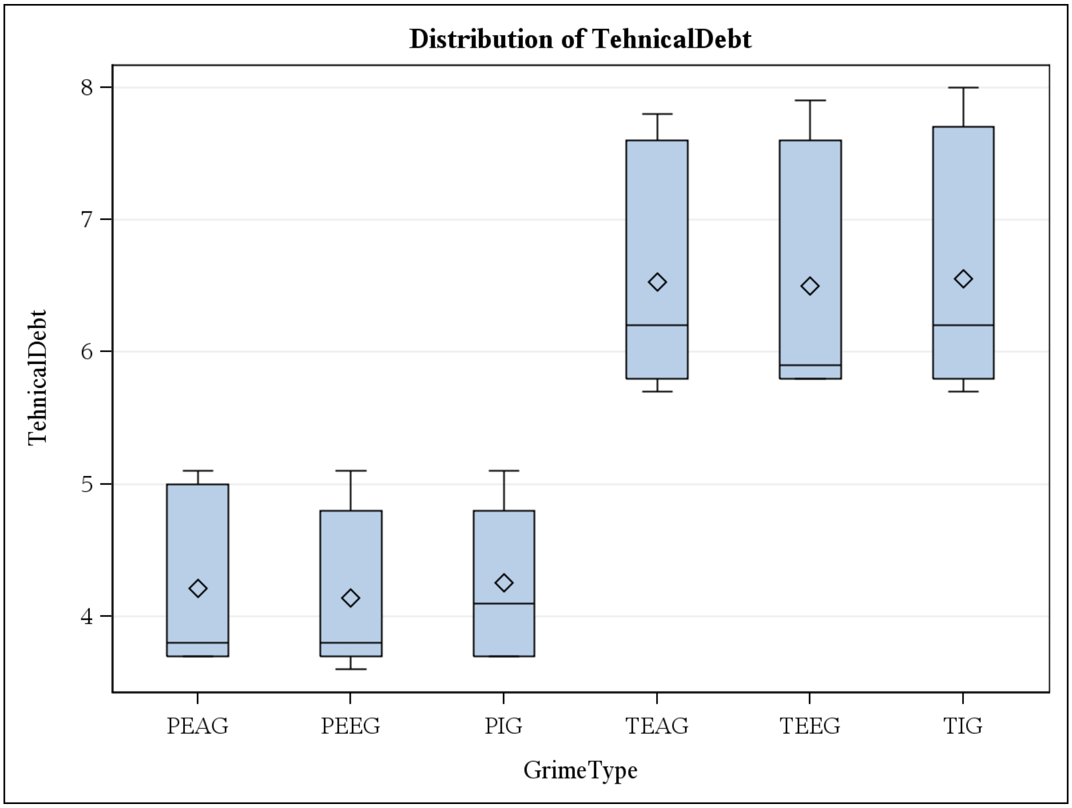
Note: The X'X matrix has been found to be singular, and a generalized inverse was used to solve the normal equations. Terms whose estimates are followed by the letter 'B' are not uniquely estimable.


Fit Diagnostics for TehnicalDebt
Residual
Predicted Value
RStudent
Predicted Value
RStudent
Leverage
Residual
Quantile
TehnicalDebt
Predicted Value
Cook's D
Observation
Percent
Residual
Fit–Mean
Residual
Proportion Less
Observations 90
Parameters 8
Error DF 82
MSE 0.0471
R-Square 0.9772
Adj R-Square 0.9753


Interaction Plot for TehnicalDebt
TehnicalDebt
8
7
6
5
4
PEAG
PEEG
PIG
TEAG
TEEG
TIG
GrimeType
DPattern
Deco
Fact
Obse

100 instances of Modular Grime

The GLM Procedure



100 instances of Modular Grime

The GLM Procedure

Tukey's Studentized Range (HSD) Test for TehnicalDebt

Note: This test controls the Type I experimentwise error rate.

Alpha	0.05
Error Degrees of Freedom	82
Error Mean Square	0.04707
Critical Value of Studentized Range	4.12696
Minimum Significant Difference	0.2312

Comparisons significant at the 0.05 level are indicated by ***.				
GrimeType Comparison	**Difference Between Means**	**Simultaneous 95% Confidence Limits**		
TIG - TEAG	0.02667	-0.20452	0.25785	
TIG - TEEG	0.05333	-0.17785	0.28452	
TIG - PIG	2.30000	2.06882	2.53118	***
TIG - PEAG	2.34000	2.10882	2.57118	***
TIG - PEEG	2.41333	2.18215	2.64452	***
TEAG - TIG	-0.02667	-0.25785	0.20452	
TEAG - TEEG	0.02667	-0.20452	0.25785	
TEAG - PIG	2.27333	2.04215	2.50452	***
TEAG - PEAG	2.31333	2.08215	2.54452	***
TEAG - PEEG	2.38667	2.15548	2.61785	***
TEEG - TIG	-0.05333	-0.28452	0.17785	
TEEG - TEAG	-0.02667	-0.25785	0.20452	
TEEG - PIG	2.24667	2.01548	2.47785	***
TEEG - PEAG	2.28667	2.05548	2.51785	***
TEEG - PEEG	2.36000	2.12882	2.59118	***
PIG - TIG	-2.30000	-2.53118	-2.06882	***
PIG - TEAG	-2.27333	-2.50452	-2.04215	***

Comparisons significant at the 0.05 level are indicated by ***.				
GrimeType Comparison	**Difference Between Means**	**Simultaneous 95% Confidence Limits**		
PIG - TEEG	-2.24667	-2.47785	-2.01548	***
PIG - PEAG	0.04000	-0.19118	0.27118	
PIG - PEEG	0.11333	-0.11785	0.34452	
PEAG - TIG	-2.34000	-2.57118	-2.10882	***
PEAG - TEAG	-2.31333	-2.54452	-2.08215	***
PEAG - TEEG	-2.28667	-2.51785	-2.05548	***
PEAG - PIG	-0.04000	-0.27118	0.19118	
PEAG - PEEG	0.07333	-0.15785	0.30452	
PEEG - TIG	-2.41333	-2.64452	-2.18215	***
PEEG - TEAG	-2.38667	-2.61785	-2.15548	***
PEEG - TEEG	-2.36000	-2.59118	-2.12882	***
PEEG - PIG	-0.11333	-0.34452	0.11785	
PEEG - PEAG	-0.07333	-0.30452	0.15785	

100 instances of Modular Grime

The GLM Procedure

Tukey's Studentized Range (HSD) Test for TehnicalDebt

Note: This test controls the Type I experimentwise error rate, but it generally has a higher Type II error rate than REGWQ.

Alpha	0.05
Error Degrees of Freedom	82
Error Mean Square	0.04707
Critical Value of Studentized Range	4.12696
Minimum Significant Difference	0.2312

Means with the same letter are not significantly different.			
Tukey Grouping	**Mean**	**N**	**GrimeType**
A	6.55333	15	TIG
A			
A	6.52667	15	TEAG
A			
A	6.50000	15	TEEG
B	4.25333	15	PIG
B			
B	4.21333	15	PEAG
B			
B	4.14000	15	PEEG