Project Number: GFP1204

Software Processes at PayPal

A Major Qualifying Project Report:
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

Sponsoring Agency: PayPal

Submitted by:

____

Jared Andrews

____

Glen Lovett

Date: March 11, 2013

Approved:

____

Professor Gary Pollice, Advisor

Abstract

Static Code Analysis (SCA) is the process of analyzing software source code for potential defects. This project implements a plugin for and analyzes one such SCA tool, Sonar, with the goal of determining its effectiveness in correctly identifying problem areas in code. Real world data which was collected at PayPal after setting up a continuous integration system using Jenkins CI and Sonar was used as the basis for this project.

Acknowledgements

We would like to extend our thanks to all of those who made this project possible. We would like to thank Professor Gary Pollice, who provided us with this wonderful opportunity and continued support and guidance throughout the course of this project. We would also like to thank our sponsor, PayPal for making this project possible. Additionally, we would like to thank our manager at PayPal, Serkan Ozel for setting up this project and supporting us throughout. Furthermore, thanks to Scott MacDonald at PayPal for teaching us the processes of release engineering and Continuous Integration.

List of Figures

Table of Contents

1 Introduction

Completing the entire implementation of a software project prior to testing leads to defect detection long after defect injection. (Royce 1970) To address the flaws in this approach, commonly referred to as waterfall development, many software development teams have adopted iterative development procedures. Iterative development supports the delivery of a product at the conclusion of each iteration. The practice of Continuous Integration (CI) facilitates an iterative approach and has been adopted by many software development teams.

CI is the practice of committing small, focused changes to a software project and continuously merging (integrating) these changes with a central source that all developers access. The goal of this practice is to always have an up to date, working product that can be developed and tested in parallel. CI facilitates more frequent feedback on committed code, which in turn can reduce technical debt. (Fowler 2006)

Technical debt describes the inevitable loss in value that a product accrues over time. No agreed upon definition of technical debt exists, but factors such as defects introduced, lack of test suites, parallel development requiring merges, and excessive refactoring are seen as common indicators of technical debt. For the purpose of this report, we measure technical debt as the amount of work required to fix defects.

1.1 Problem Statement

Many best practices and technologies contribute to a working CI system. These include small atomic commits, automated builds, automated tests, and static code analysis. Static code analysis (SCA) is a method of scanning source code for potential weak spots and violations. The first SCA tools appeared in 1979 (Johnson 1977), so they are by no means a new technology. SCA proponents claim that SCA reduces technical debt of a project when used effectively. (Gaudin, Evaluate Your Technicla Debt with Sonar 2009)

Many software teams use SCA to guide refactoring efforts and manage the technical debt of their projects. (Parasoft 2012) Introducing SCA introduces overhead in the form of setup and maintenance of an SCA server, in addition to the cost of actions taken based on its results. Therefore, in order to be a worthy investment, measurable benefits of tool adoption must exist.

This project aims to make an existing SCA tool, Sonar, more accessible to developers through the development of a custom plugin. Additionally, the relationship between Sonar analysis and real defects will be measured using data provided by the PayPal Here Android team.

1.2 Hypothesis

We seek to determine if SCA tools accurately measure which parts of a software project are accruing the most technical debt, which would correspond with real world loss of value. If this can be shown to be the case, then it helps validate the usage of SCA tools.

2 Background

2.1 Sponsor Information

PayPal™ is a global company that provides solutions for transferring money via the Internet. Founded in 1998, PayPal became a subsidiary of eBay™ in 2002 and employs more than 10,000 people throughout the world. PayPal receives funds, converts them to foreign currencies if needed, and distributes them to the involved parties. PayPal places a heavy emphasis on security and has become highly trusted as a result. In addition to online transactions, PayPal has been breaking into real world transactions with smart phone enabled 'wallet' applications that allow a customer to use their PayPal account at brick and mortar stores.

2.2 The Android PayPal Here Team

This report was done in conjunction with PayPal, specifically the Android PayPal Here™ Team. PayPal Here is a smart phone application for iPhone and Android devices that enables businesses and individuals to accept credit, debit and PayPal payments via their smart phone. This eliminates the need for costly credit card processing equipment and it transfers funds into the seller's account shortly after the transaction is completed. In order to use PayPal Here, credit cards are scanned either through a smartphone camera or a free dongle provided by PayPal. These transactions provide the core of PayPal Here's functionality.

The subject of this report is the Android PayPal Here team. This team has been developing PayPal Here since January 2012 and the lead developer, Serkan Ozel, felt that it would be beneficial for the team to have CI technologies at their disposal. Serkan felt that introducing CI into his teams existing work flow would have many benefits such as early detection of bugs, violations, and suspicious code. If caught sooner rather than later, CI could save time and money.

The authors of this report interned at PayPal and set up a CI system for the Android PayPal Here team over the summer of 2012. Further research outlined in the methodology and analysis sections of this report was completed from August 2012 to March 2013.

2.3 Tools Used

The following tools were used to build the continuous integration system for the PayPal Here Android Team. They work together to automate several areas of the release process.

2.3.1 Jenkins

Jenkins is an open source CI server. It monitors source repositories and builds artifacts based on parameters set by the user. (Jenkins CI Community 2011) Jenkins is extensible via plugins and scripting. Jenkins manages the entire build process from compilation to testing to artifact publication.

A Jenkins instance is separated into several projects and each project has its own set of jobs. A job is a set of instructions for completing a certain task. Jenkins can be configured to alert developers of any problems encountered during the build process. For example if the code fails to compile Jenkins can be configured to send an alert email to a manager. Using Jenkins to automate the release process removes the potential for human error and allows for easily repeatable builds.

In a typical Jenkins setup a job is tied to one particular task. For example, a job associated with a development branch could compile with the debug features on whereas the release branch job turns debugging features off. This allows each job to be tailored to a particular goal.

2.3.2 Sonar

Sonar is an open source SCA tool that tracks metrics of source code quality over time. These metrics are determined by matching Java™ source code to patterns that are known to lead to or cause unintended behavior within a program. (SonarSource 2012) Sonar classifies violations into five categories: info, minor, major, critical and blocker, with blocker being the most severe of violations and info the least. Violations are tracked over time and on a per file basis allowing users to explore their source code and see where potential problems lie.

Sonar is also capable of tracking other metrics such as the amount of duplicated code, code coverage, and unit test success and code complexity. Sonar is easily extensible and there are many plugins available that add new metrics, language support and analysis. A Sonar analysis can take a fair amount of time to complete for large projects so it advisable to create a standalone Jenkins job for triggering a Sonar analysis.

Sonar's effectiveness has never been formally studied. (Gaudin, Personal Communication 2012) Sonar is the preferred tool of the PayPal release engineering team so it would be beneficial to verify its usefulness.

3 Methodology

3.1 Sonar Trends Plugin

Though the Android PayPal Here team now had a system for CI in place, the features afforded by this system were not made apparent to the team, and thus adoption was limited to occasional checks on the status of the builds. Sonar saw little use during the course of the internship because when a team member would look at the Sonar dashboard they were never given any clear actions to take. The first stage of the MQP was the implementation of a Sonar plugin in order to provide actionable data which could help the team determine which parts of the source were potential sources of technical debt.

The current state of the project as displayed by Sonar would have improved context if a plugin was designed that displayed historical Sonar data. In addition, we desired to easily detect the packages with the highest and lowest number of certain Sonar metrics. A plugin that displayed a timeline or historical Sonar data represented as a line graph was determined to be the best way to meet this goal.

Packages were chosen as the software component of interest because they are the smallest component of a project that Sonar stores historical data for. Since the plugin makes use of historical data, package level analysis was the obvious choice. During our time working directly with the PayPal team in the summer of 2012, we found that Sonar's project oriented dashboard did not directly address what developers were most interested in. The requirements of the Trends plugin were:

- To display metric trends of specific packages
- To allow users to interact with the plugin and retrieve information based on selected date range, Sonar metric and the number of packages to be shown.

3.1.1 Technology Requirements

Upon determining the requirements of the plugin we researched the ways in which a Sonar plugin could be created. The Sonar website provides some documentation on plugin development but for the most part we relied on the source code of existing plugins to get the full picture. Some Sonar plugins focus on adding new information to Sonar, such as definitions for a new programming language, and others focus on reformatting existing data to change and extend the way it is presented to the Sonar user. Our plugin falls is in the latter category.

Sonar plugins are implemented using Java, Ruby, JavaScript, HTML and the Sonar web services API. The Trends plugin primarily uses JavaScript in conjunction with the web services API. Boilerplate Ruby and Java is required to hook into the Sonar instance. The plugin interacts with Sonar using the web services API which facilitates querying of the Sonar database. The Protovis JavaScript library is used to generate the chart.

3.1.2 Plugin Design

The plugin was designed to allow for users to easily query the Sonar database and retrieve information on packages based on the parameters shown in Figure 1.

Metric	The metric of interest. Included were the default Sonar metrics for violations, size and duplication. Other metrics could easily be added if needed.
Calculate Metric	How the metric information is used: • Per Line: The metric is calculated as a ratio of a package's total lines of code. • Total: The raw total is used.
Number of Packages	The number of packages to display on the chart.
View Based On	• Highest: Graph the packages with the highest values for the selected metric. • Lowest: Graph the packages with the lowest values for the selected metric.
From Date	The lower limit of the date range.
To Date	The upper limit of the date range.

Figure 1 – Parameters of the Sonar Trends plugin

By adjusting these parameters a user can find many points of interest within the source code that might need refactoring. Additionally, the legend beneath the chart provides links to a breakdown of each package's components allowing for deeper exploration of each package if the user desires to do so.

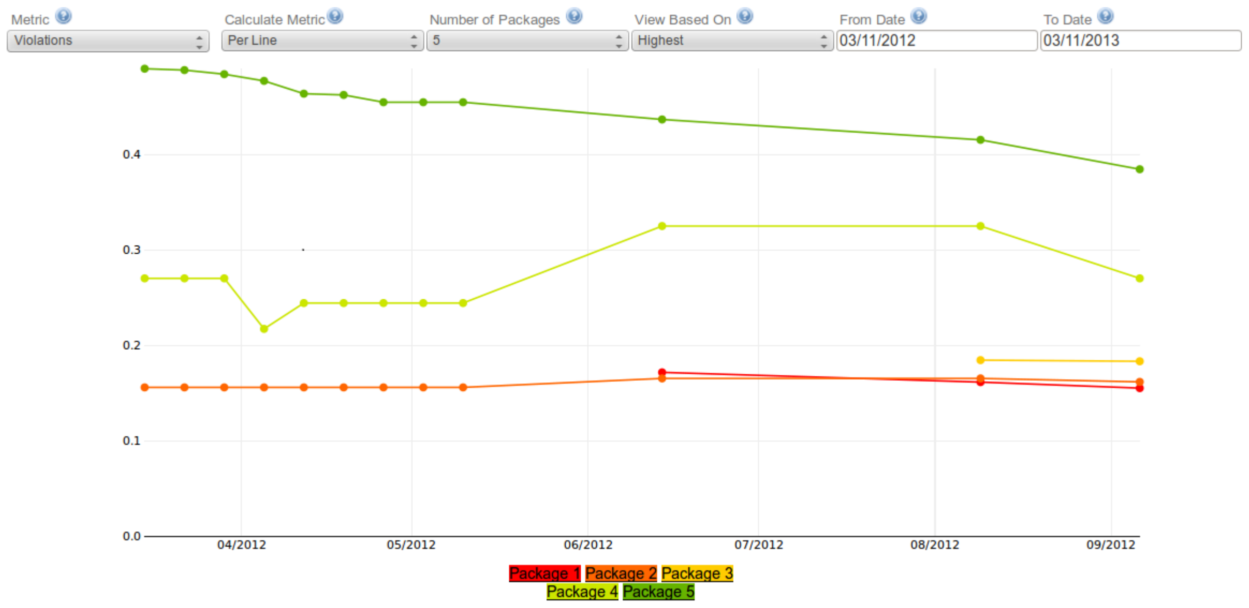

Metric: Violations
Calculate Metric: Per Line
Number of Packages: 5
View Based On: Highest
From Date: 03/11/2012
To Date: 03/11/2013

Figure 2 – Sonar Trends plugin as seen on a Sonar dashboard

3.1.3 Deployment

The Trends plugin was deployed at PayPal in December of 2012. We created a script that regenerated Sonar history. This allowed the team to query historical data using the plugin. The Sonar instance and plugin are hosted on a machine which scans the source repository each night. A more detailed overview of the Trends plugin can be found in Appendix A and a sample use case can be found in Appendix B.

3.2 Analysis

The primary motive of this project was to determine if there is a correlation between Sonar violations and the bugs found in live code bases. For the second half of our project the JIRA and Git logs of the PayPal Here project were studied to determine if such a correlation exists. Historical data was provided by PayPal from the entirety of Android PayPal Here's existence and was used as the basis for this analysis. Ideally each defect would be weighted based on the units of work required fix the defect. This information was not available, so, for the purposes of our analysis, all defects are considered to require equal work to resolve. Thus all defects are treated as though they incur the same amount of technical debt.

3.2.1 Data Sources

Our analysis relied on three primary data sources: Sonar, Git and JIRA. Git is a revision control system and JIRA is an issue tracker; both are used by the Android PayPal Here team. Between Git logs and JIRA logs, a detailed summary of the PayPal Here code base was constructed. The Git logs provide a complete history of the code while JIRA contains information on the defects that were discovered in the code base as it evolved. In order to conduct a proper analysis we connected these two sources of information.

The PayPal Here Android team uses JIRA ticket numbers in the messages of some commits relating to those tickets. We were able to use this convention to connect the Git logs to the bugs reported in JIRA. By using this information we were able to identify the packages that belonged to 129 defect reports from December 15, 2011 to January 17, 2013. These data points represent the entire lifespan of the PayPal Here Android Application. Their distribution can be seen in Figure 3.

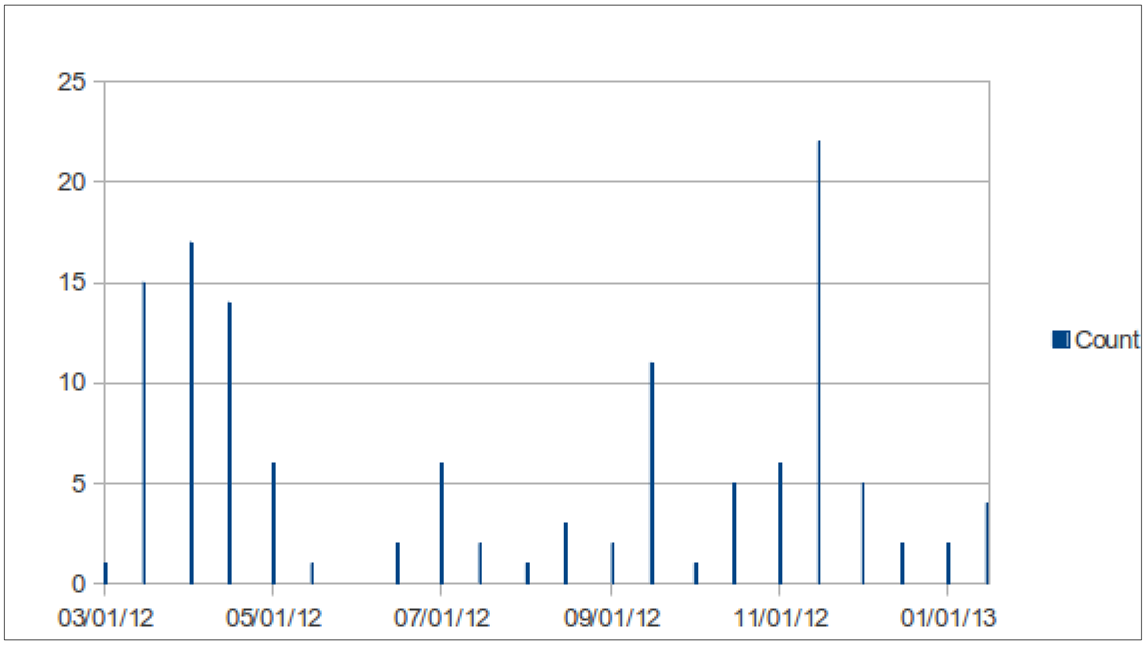
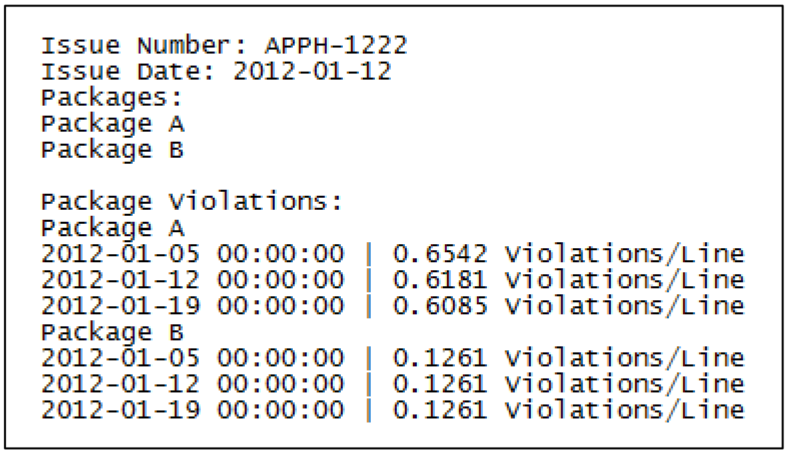


Figure 3 – JIRA ticket frequency plotted over time

Once this association was made, the Sonar time machine web API was used to retrieve information on the packages involved in each bug. This process was automated with a Python script that searched JIRA for all bugs reported, compared them to Git logs and then sent the packages involved to Sonar. Once completed the script provided relevant information on each defect.



```
Issue Number: APPH-1222
Issue Date: 2012-01-12
Packages:
Package A
Package B

Package Violations:
Package A
2012-01-05 00:00:00 | 0.6542 Violations/Line
2012-01-12 00:00:00 | 0.6181 Violations/Line
2012-01-19 00:00:00 | 0.6085 Violations/Line
Package B
2012-01-05 00:00:00 | 0.1261 Violations/Line
2012-01-12 00:00:00 | 0.1261 Violations/Line
2012-01-19 00:00:00 | 0.1261 Violations/Line
```

Figure 4 – Example of Sonar violation information associated with a bug ticket

Figure 4 shows sample Sonar violation information before the bug fix, on the day the bug fix occurred and a week after the bug fix. This data set was used to determine the validity of Sonar violations in the following analyses.

3.2.2 Goal, Question, Metric Approach

The Goal, Question, Metric approach (GQM) is a method developed by Victor Basili for guiding software metric research. A GQM approach first determines the goals of a study, each with questions that address those goals. Metrics are then selected that answer those questions. (Victor Basili 1994) GQM was used to guide decisions and research. Specifically many goals were proposed and removed as our interests narrowed. For our analysis of Sonar we used GQM to determine what metrics needed to be measured.

3.2.3 Analysis Process

Once information was retrieved on as many defects as possible, statistical analysis was performed to determine if violations could be used to predict whether or not a package would be present in defects. Violations were chosen as the Sonar metric to consider because of their prominence within Sonar over other metrics. By using the GQM model that had been refined through the course of the project, a set of measurements was determined and programmatically extracted from the data described in the preceding paragraphs.

Goal	Determine if static code analysis techniques accurately measure which parts of a project are accruing the most technical debt and that these measurements correspond with real defects in the source code.
Purpose	Determine
Issue	Accuracy of
Object	Sonar SCA
Focus	Code Quality, Defects
Viewpoint	PayPal, Android PayPal Here Team
Questions	
1	If a package has a violations per line ratio higher than the overall project average is it more likely to appear in a JIRA bug ticket?
2	What are the average violations per line for all packages involved in JIRA bug tickets?
3	What are the average violations per line for the overall project?
4	What is the relationship between the results of 2 and 3?

Figure 5 – Final GQM model

Using the GQM model shown in Figure 5 as a guide, historical data was analyzed to determine if a relationship could be found between JIRA bug tickets and Sonar violations. If question 1 is shown to be true then a case can be made for using Sonar as a guide in predicting and refactoring packages in a software project. Data extracted from the Git and JIRA logs answered questions 1, 2 and 3. Question 4 was answered by comparing the results of questions 2 and 3.

4 Results

Questions found in the GQM table guided our analysis. The average violations per line of the project as a whole were calculated, as was the average violations per line of every package spanning the Android PayPal Here projects lifetime. The number of times each package appeared in a defect report was counted and a comparison was performed to determine if packages that had a higher number of violations over time appeared in more defects. Our sample looks at 65 individual packages appearing in 101 defect reports. On average each appears in a defect report 3.36 times. A complete statistical description can be found in Figure 6. A complete listing of the data used for our analysis can be found in Appendix C: Package Data

Minimum	1
Maximum	15
Range	14
Mean	3.36923077

Figure 6 - Descriptive statistics for package defect frequency

This analysis was done for all violations per line, blocker violations per line, major violations per line, minor violations per line and info violations per line. Violations per line represent an aggregation of all 5 types of violations. Blocker violations never appeared in the packages analyzed so the results were omitted.

Figure 7 shows the percent change in package appearance in defects (frequency change) when they have violations per line greater than the project average. In many cases, a package's violations per line ratio was only slightly higher than the project average. Frequency change was also calculated for packages that had a violations per line ratio greater than 110% the project average and 120% the project average. This was done to eliminate packages that were only slightly higher than the project average and put focus on those that were substantially higher than the project average. The individual categories of violations were also calculated to see if certain violation types tend to influence a packages appearance in defects more than others.

	Project Average	110% Project Average	120% Project Average
Violations / Line Frequency Change	**+18.3%**	**+27.30%**	**+56.25%**
Major Violations / Line Frequency Change	-11.56%	+1.19 %	+44.15%
Minor Violations / Line Frequency Change	+12.28 %	+26.49%	+26.49%
Info Violations / Line Frequency Change	-8.59 %	-1.35 %	-3.42%

Figure 7 – Change in frequency for packages that have a higher level of violations than the project average

In general, when a package has violations per line metric that is higher than the project it more like to be involved with a defect. This is consistent with the notion that Sonar can be used to predict what packages are more likely to be involved with defects. Because the total population is unknown a t-test was used to calculate the test statistic. At most these values are 15% likely to be the result of random chance.

4.1 Weka Data Mining

The data shown in Figure 4 lends itself to being mined for patterns and associations. Weka, a data mining tool developed by the Machine Learning Group at the University of Waikato, can be used to perform such data mining techniques. (Mark Hall 2009)

By performing an attribute evaluation with Weka on the data in Figure 4, we can determine which attributes of a package are the best indicators of the frequency of that package in JIRA bug tickets, out of the following attributes:

- Average violations/line
- Average blocker violations/line
- Average major violations/line
- Average minor violations/line
- Average info violations/line

Weka uses an algorithm known as Cfs Subset Evaluation to "evaluate the worth of a subset of attributes by considering the individual predictive ability of each." (Hall 1998) This evaluation selected the following three attributes as the most helpful in predicting the frequency of a package in JIRA bug tickets:

- Average violations/line
- Average major violations/line
- Average info violations/line

This fact that this subset of attributes was chosen is not surprising, as average violations per line and average major violations per line percentages are highest in Figure 7. This shows that they are the most indicative of the number of times a package is likely to appear in a bug ticket. To have Weka validate this using a different approach however, helps confirm the associations in Figure 7. The fact that info violations per line was also selected was not expected however, as Figure 7 does not show as high of a relationship with info violations, so there must be something unexplained by the values Figure 7 that the Cfs Subset Evaluation detected.

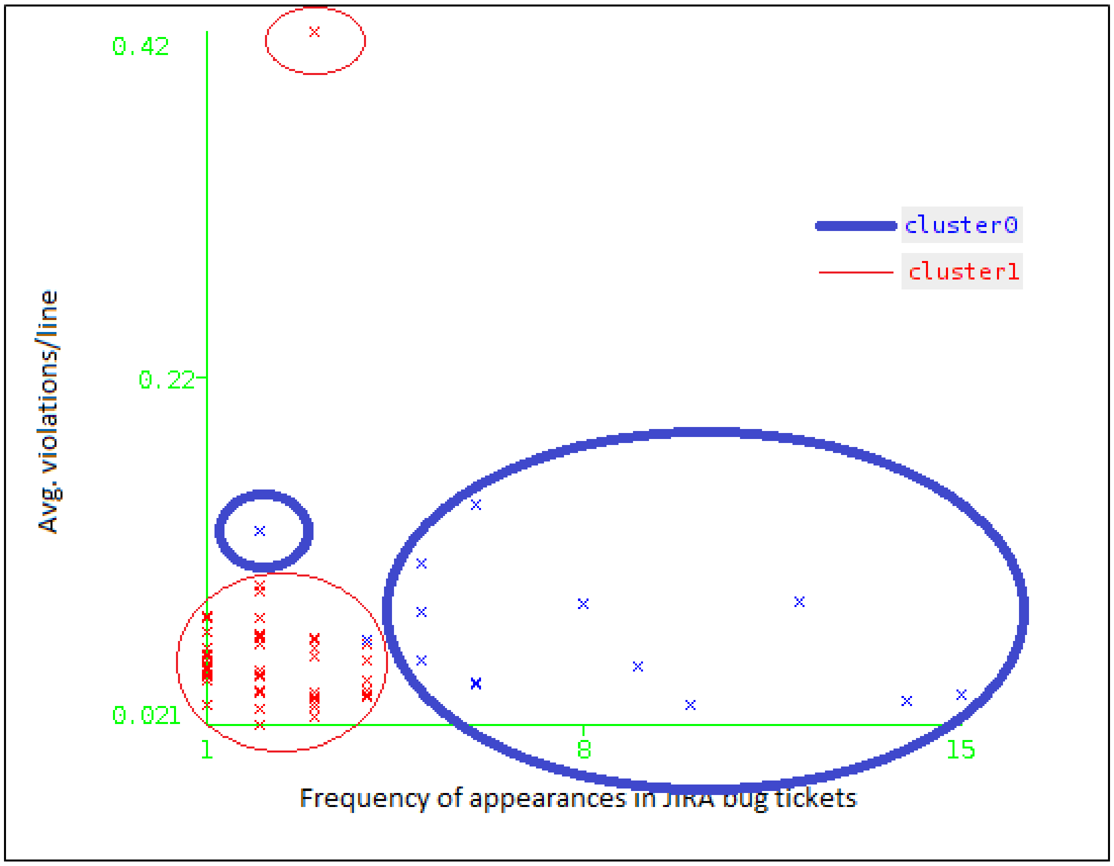


Figure 8 – Average violations/line plotted against the frequency of appearances in JIRA bug tickets, with clusters

Figure 8 displays the result of performing a *k-means* density based clustering in Weka, which creates k clusters (in this case 2) of data instances by creating k centroids with which to associate the nearest instances based on the mean of their values, and iteratively move those centroids to the mean location of the instances closest to them in N dimensions where N is the number of attributes of each instance. The algorithm then re-assigns any instances which are closer to a different centroid. (Vassilvitskii 2007) When this method is performed on our data, the two clusters seen above are generated. The red cluster contains mostly data instances with low violations per line, and low appearances in JIRA bug tickets, and could be classified as low risk packages. The blue instances however, contain high violations per line, high appearances in JIRA tickets, or both so should be considered high risk. There is one red cluster instance with very high (greater than 0.4) violations per line. This is likely due to a shortcoming in the clustering method and indicates that this instance is an anomaly. This clustering method provides an example of a way in which future work may help users of Sonar select which packages to focus on.

5 Conclusions and Future Work

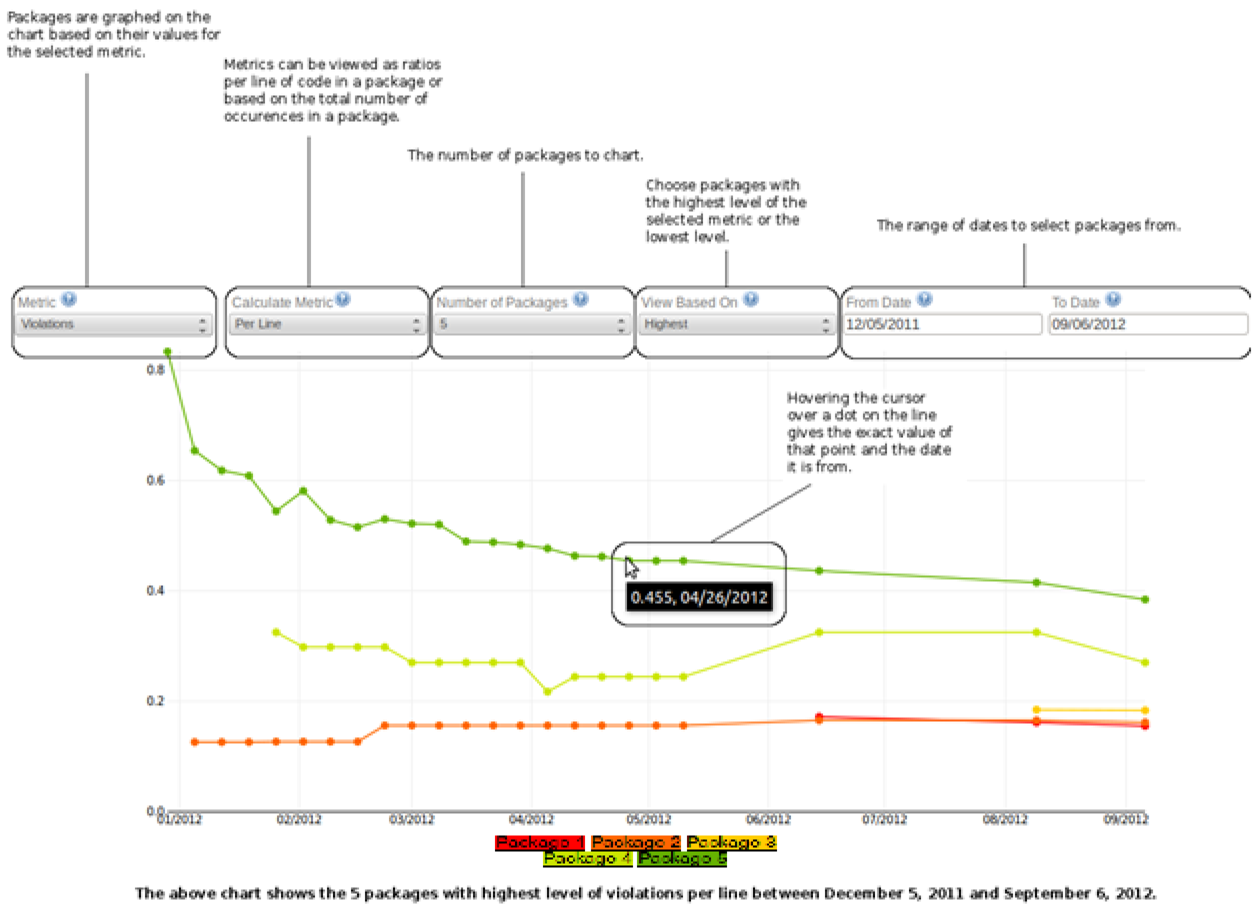
Sonar is a powerful tool for evaluating the state of a software project according to various SCA violations and software quality metrics. The problem with Sonar is that the abundance of the information presented makes it difficult for a team to find the actionable information amongst the less impactful information. Using the newly developed Trends plugin however, a user may identify the components of a project that are potentially accruing the most technical debt in order to be informed as to which components could be causing defects and should be considered dangerous to build on without first checking for quality.

Not only was this plugin developed, but its usefulness was validated through analysis of real world data from the Android PayPal Here team. The historical data which the team generated was cross referenced with Sonar analysis data in order to determine what correlations exist between their bug tickets and Sonar metrics. Our analysis found an average 18.35% increase in the number of times a package appears in a bug report when the package has a violations per line ratio greater than the project average. This correlation, however, cannot prove that resolving Sonar violations causes fewer instances of bugs in that code. Rather, we can only determine that Sonar violation and bugs are positively correlated.

As defects accumulate in a software project so does technical debt. In order to decrease technical debt a team must focus on fixing problematic areas before adding new features otherwise small problems can slowly compound into bigger ones over time. By correlating Sonar violations with real world defects it is shown that violations are indicative of increased technical debt because fixing defects requires a development team to do more work.

Although this research has not shown that acting on Sonar violations will decrease technical debt it has shown that violations indicate technical debt. A future study could be conducted in which a team fully adopts Sonar along with the Trends plugin in order to identify and resolve Sonar violations. In such a study, the data generated could be used to prove or disprove the claim that by resolving Sonar violations, fewer defects will appear in the resulting code.

6 Appendix A: Sonar Trends Plugin


Packages are graphed on the chart based on their values for the selected metric.
Metrics can be viewed as ratios per line of code in a package or based on the total number of occurences in a package.
The number of packages to chart.
Choose packages with the highest level of the selected metric or the lowest level.
The range of dates to select packages from.
Hovering the cursor over a dot on the line gives the exact value of that point and the date it is from.
0.8
0.6
0.4
0.2
0.0
01/2012
02/2012
03/2012
04/2012
05/2012
06/2012
07/2012
08/2012
09/2012
0.455, 04/26/2012
Package 1
Package 2
Package 3
Package 4
Package 5

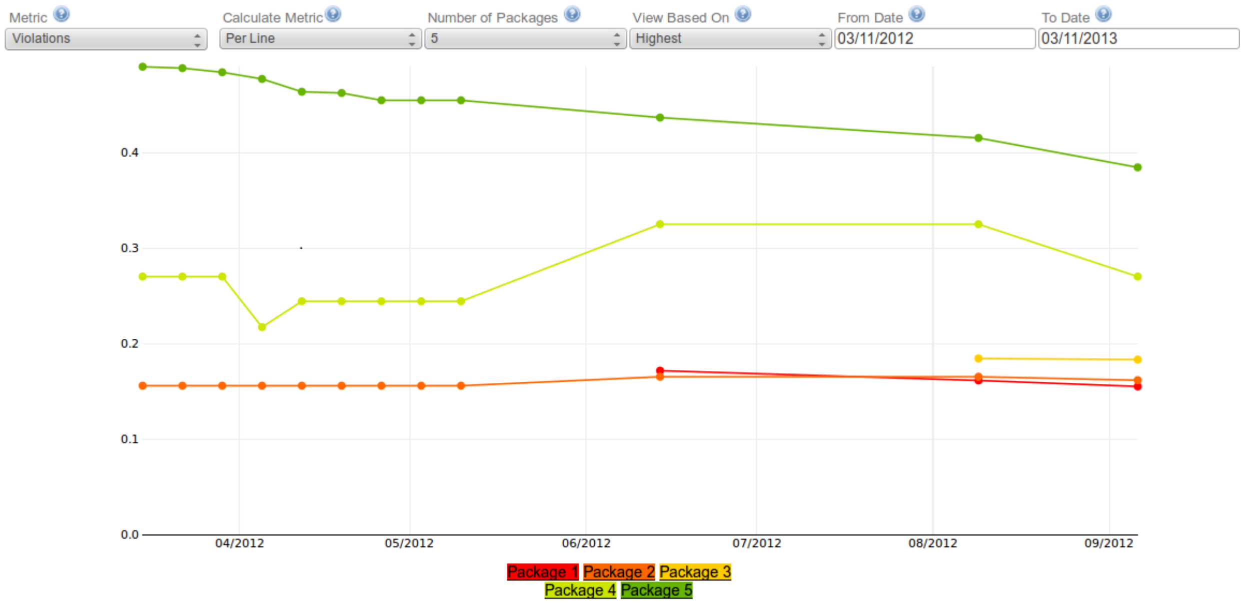
Metric: Violations
Calculate Metric: Per Line
Number of Packages: 5
View Based On: Highest
From Date: 12/05/2011
To Date: 09/06/2012

The above chart shows the 5 packages with highest level of violations per line between December 5, 2011 and September 6, 2012.

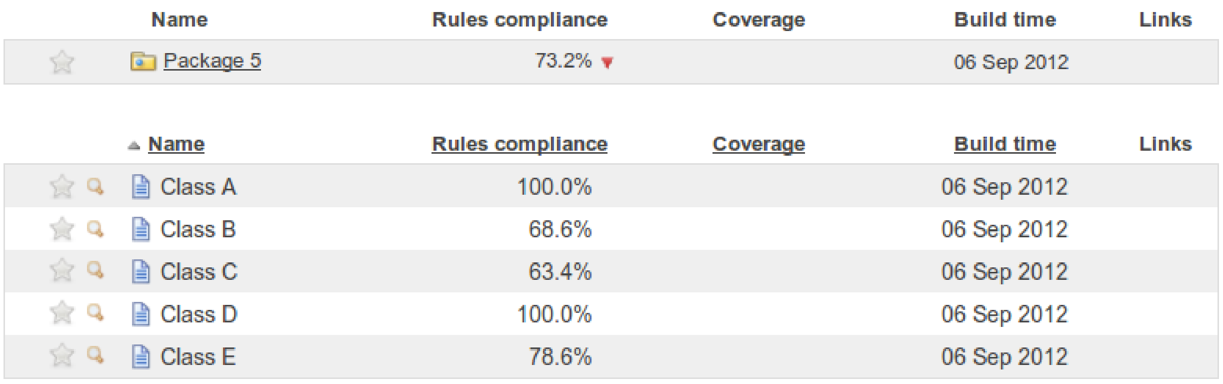
Figure 9 – An overview of the options provided by the Trends plugin

7 Appendix B: Trends Plugin Run Through

The following is a sample usage scenario of the trends plugin.


Metric: Violations
Calculate Metric: Per Line
Number of Packages: 5
View Based On: Highest
From Date: 03/11/2012
To Date: 03/11/2013

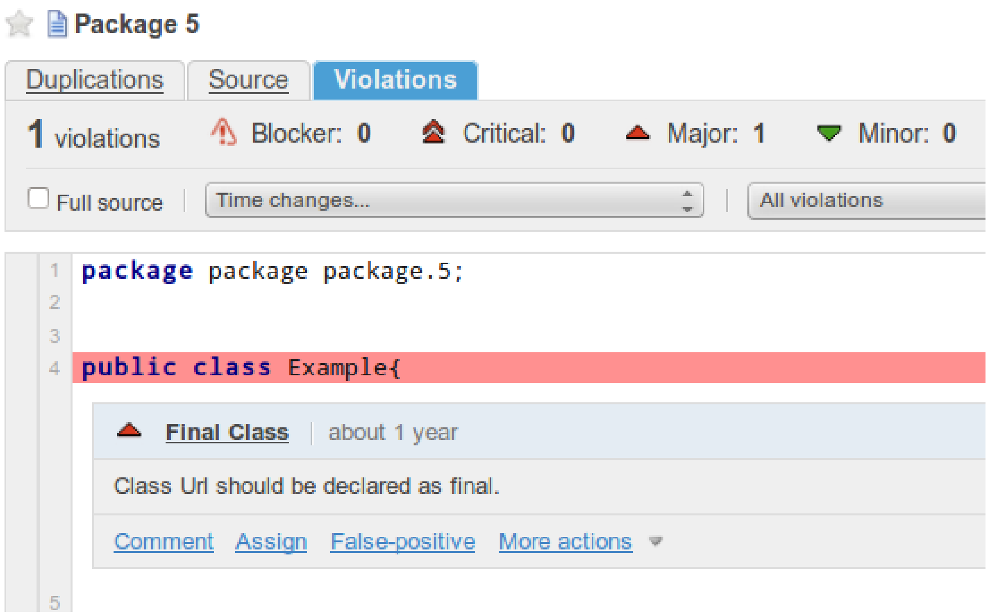
1) A query shows the five packages in the PayPal Here project with the most violations per line between December 10, 2011 and December 10, 2012. The green line which represents the package "Package 5" is the highest although it is showing a downward trend.
2) With this information we can determine that "Package 5" would be a good place to start refactoring efforts.



Name	Rules compliance	Coverage	Build time	Links
Package 5	73.2% ▼		06 Sep 2012	

Name	Rules compliance	Coverage	Build time	Links
Class A	100.0%		06 Sep 2012	
Class B	68.6%		06 Sep 2012	
Class C	63.4%		06 Sep 2012	
Class D	100.0%		06 Sep 2012	
Class E	78.6%		06 Sep 2012	

3) Clicking on "Package 5" in the legend open the packages Sonar components page. This page features a break, by file, of sonar violations and other metrics.


Package 5

1 violations Blocker: 0 Critical: 0 Major: 1 Minor: 0

```
package package package.5;


public class Example{
```

Final Class | about 1 year

Class Url should be declared as final.

4) Clicking on an individual file name opens anther windows which details exactly where in the source code a violation is located. In this case the file contains one major violation, a class that should be declared as final

5) By exploring violations and trends in the source code problematic areas can be easily uncovered and focused on during refactoring efforts.

8 Appendix C: Package Data

Data used for ours analysis of historical PayPal Here data. For sections 8.1 to 8.4 cells highlighted in red are higher than the project average. Package names were anonymized for this report.

8.1 Average Violations Per Line

Package	Avg. Violations / Line	10%	20%
Project	0.073144396	0.0804588	0.08777327
Package 1	0.048215203	0.0482152	0.0482152
Package 2	0.036818639	0.0368186	0.03681864
Package 3	0.045692527	0.0456925	0.04569253
Package 4	0.071081854	0.0710819	0.07108185
Package 5	0.050246481	0.0502465	0.05024648
Package 6	0.070581077	0.0705811	0.07058108
Package 7	0.060089686	0.0600897	0.06008969
Package 8	0.086346427	0.0863464	0.08634643
Package 9	0.036305474	0.0363055	0.03630547
Package 10	0.033616313	0.0336163	0.03361631
Package 11	0.032490228	0.0324902	0.03249023
Package 12	0.067582219	0.0675822	0.06758222
Package 13	0.039694342	0.0396943	0.03969434
Package 14	0.060872001	0.060872	0.060872
Package 15	0.148582762	0.1485828	0.14858276
Package 16	0.03616304	0.036163	0.03616304
Package 17	0.031098759	0.0310988	0.03109876
Package 18	0.039973547	0.0399735	0.03997355
Package 19	0.034489051	0.0344891	0.03448905
Package 20	0.091608373	0.0916084	0.09160837
Package 21	0.051705171	0.0517052	0.05170517
Package 22	0.050043335	0.0500433	0.05004333
Package 23	0.133231823	0.1332318	0.13323182
Package 24	0.020969245	0.0209692	0.02096925
Package 25	0.029342297	0.0293423	0.0293423
Package 26	0.097729337	0.0977293	0.09772934
Package 27	0.090510233	0.0905102	0.09051023
Package 28	0.1017047	0.1017047	0.1017047
Package 29	0.070716253	0.0707163	0.07071625
Package 30	0.040751197	0.0407512	0.0407512
Package 31	0.073082011	0.073082	0.07308201
Package 32	0.046121097	0.0461211	0.0461211
Package 33	0.038318882	0.0383189	0.03831888
Package 34	0.072317499	0.0723175	0.0723175

Package 35	0.070250232	0.0702502	0.07025023
Package 36	0.04596835	0.0459683	0.04596835
Package 37	0.031971878	0.0319719	0.03197188
Package 38	0.082682965	0.082683	0.08268297
Package 39	0.072899241	0.0728992	0.07289924
Package 40	0.069426871	0.0694269	0.06942687
Package 41	0.024930129	0.0249301	0.02493013
Package 42	0.060799282	0.0607993	0.06079928
Package 43	0.057876559	0.0578766	0.05787656
Package 44	0.08278995	0.08279	0.08278995
Package 45	0.061042041	0.061042	0.06104204
Package 46	0.048806492	0.0488065	0.04880649
Package 47	0.058572897	0.0585729	0.0585729
Package 48	0.03904064	0.0390406	0.03904064
Package 49	0.113865103	0.1138651	0.1138651
Package 50	0.074553151	0.0745532	0.07455315
Package 51	0.084223213	0.0842232	0.08422321
Package 52	0.065191467	0.0651915	0.06519147
Package 53	0.053138657	0.0531387	0.05313866
Package 54	0.044717091	0.0447171	0.04471709
Package 55	0.053939069	0.0539391	0.05393907
Package 56	0.065073716	0.0650737	0.06507372
Package 57	0.043445978	0.043446	0.04344598
Package 58	0.060187814	0.0601878	0.06018781
Package 59	0.054570384	0.0545704	0.05457038
Package 60	0.051615445	0.0516154	0.05161545
Package 61	0.4242147	0.4242147	0.4242147
Package 62	0.054560261	0.0545603	0.05456026
Package 63	0.037961312	0.0379613	0.03796131
Package 64	0.058502071	0.0585021	0.05850207
Package 65	0.06754061	0.0675406	0.06754061

8.2 Average Major Violation Per Line

Package	Avg. Major / Line	10%	20%
Project	0.049189349	0.054108284	0.064929941
Package 1	0.032935285	0.032935285	0.032935285
Package 2	0.025312815	0.025312815	0.025312815
Package 3	0.045692527	0.045692527	0.045692527
Package 4	0.049537868	0.049537868	0.049537868
Package 5	0.042584784	0.042584784	0.042584784
Package 6	0.039397627	0.039397627	0.039397627
Package 7	0.033781764	0.033781764	0.033781764
Package 8	0.065456327	0.065456327	0.065456327
Package 9	0.029044379	0.029044379	0.029044379
Package 10	0.022545966	0.022545966	0.022545966
Package 11	0.017827229	0.017827229	0.017827229
Package 12	0.040115649	0.040115649	0.040115649
Package 13	0.032274396	0.032274396	0.032274396
Package 14	0.050275418	0.050275418	0.050275418
Package 15	0.115940831	0.115940831	0.115940831
Package 16	0.026202881	0.026202881	0.026202881
Package 17	0.025156931	0.025156931	0.025156931
Package 18	0.034572709	0.034572709	0.034572709
Package 19	0.028688738	0.028688738	0.028688738
Package 20	0.039275759	0.039275759	0.039275759
Package 21	0.029152915	0.029152915	0.029152915
Package 22	0.035855295	0.035855295	0.035855295
Package 23	0.132775029	0.132775029	0.132775029
Package 24	0.015222119	0.015222119	0.015222119
Package 25	0.008448401	0.008448401	0.008448401
Package 26	0.059582198	0.059582198	0.059582198
Package 27	0.066263195	0.066263195	0.066263195
Package 28	0.053755773	0.053755773	0.053755773
Package 29	0.069311295	0.069311295	0.069311295
Package 30	0.034644654	0.034644654	0.034644654
Package 31	0.064153439	0.064153439	0.064153439
Package 32	0.039403974	0.039403974	0.039403974
Package 33	0.031220063	0.031220063	0.031220063
Package 34	0.057216573	0.057216573	0.057216573
Package 35	0.063299351	0.063299351	0.063299351
Package 36	0.032152725	0.032152725	0.032152725
Package 37	0.025443589	0.025443589	0.025443589
Package 38	0.038771352	0.038771352	0.038771352

Package 39	0.0548192	0.0548192	0.0548192
Package 40	0.055438399	0.055438399	0.055438399
Package 41	0.016741196	0.016741196	0.016741196
Package 42	0.036192187	0.036192187	0.036192187
Package 43	0.0332117	0.0332117	0.0332117
Package 44	0.061497574	0.061497574	0.061497574
Package 45	0.042809536	0.042809536	0.042809536
Package 46	0.036071788	0.036071788	0.036071788
Package 47	0.055740745	0.055740745	0.055740745
Package 48	0.032511659	0.032511659	0.032511659
Package 49	0.083736821	0.083736821	0.083736821
Package 50	0.048682973	0.048682973	0.048682973
Package 51	0.055494829	0.055494829	0.055494829
Package 52	0.053840378	0.053840378	0.053840378
Package 53	0.046781883	0.046781883	0.046781883
Package 54	0.033959677	0.033959677	0.033959677
Package 55	0.027709439	0.027709439	0.027709439
Package 56	0.043509575	0.043509575	0.043509575
Package 57	0.037457135	0.037457135	0.037457135
Package 58	0.045075344	0.045075344	0.045075344
Package 59	0.020018282	0.020018282	0.020018282
Package 60	0.036643026	0.036643026	0.036643026
Package 61	0.022669513	0.022669513	0.022669513
Package 62	0.045921258	0.045921258	0.045921258
Package 63	0.030814683	0.030814683	0.030814683
Package 64	0.052289462	0.052289462	0.052289462
Package 65	0.049871758	0.049871758	0.049871758

8.3 Average Minor Violations Per Line

Package	Avg. Minor / Line	10%	20%
Project	0.019343035	0.021277339	0.023405073
Package 1	0.01059322	0.01059322	0.01059322
Package 2	0.009276571	0.009276571	0.009276571
Package 3	0	0	0
Package 4	0.017255137	0.017255137	0.017255137
Package 5	0.007008374	0.007008374	0.007008374
Package 6	0.024794646	0.024794646	0.024794646
Package 7	0.019133034	0.019133034	0.019133034
Package 8	0.016899856	0.016899856	0.016899856
Package 9	0.00700721	0.00700721	0.00700721
Package 10	0.007060279	0.007060279	0.007060279
Package 11	0.011229913	0.011229913	0.011229913
Package 12	0.025659559	0.025659559	0.025659559
Package 13	0.007166814	0.007166814	0.007166814
Package 14	0.006862104	0.006862104	0.006862104
Package 15	0.027518387	0.027518387	0.027518387
Package 16	0.006486873	0.006486873	0.006486873
Package 17	0.005127222	0.005127222	0.005127222
Package 18	0.003931222	0.003931222	0.003931222
Package 19	0.004340459	0.004340459	0.004340459
Package 20	0.035396834	0.035396834	0.035396834
Package 21	0.001650165	0.001650165	0.001650165
Package 22	0.013674445	0.013674445	0.013674445
Package 23	0.000380662	0.000380662	0.000380662
Package 24	0.002640572	0.002640572	0.002640572
Package 25	0.010446948	0.010446948	0.010446948
Package 26	0.030154405	0.030154405	0.030154405
Package 27	0.021008078	0.021008078	0.021008078
Package 28	0.045877479	0.045877479	0.045877479
Package 29	0.001404959	0.001404959	0.001404959
Package 30	0.002731067	0.002731067	0.002731067
Package 31	0.008928571	0.008928571	0.008928571
Package 32	0.00359508	0.00359508	0.00359508
Package 33	0.005158311	0.005158311	0.005158311
Package 34	0.012293216	0.012293216	0.012293216
Package 35	0.005931418	0.005931418	0.005931418
Package 36	0.011680482	0.011680482	0.011680482
Package 37	0.005021761	0.005021761	0.005021761
Package 38	0.042281774	0.042281774	0.042281774

Package 39	0.015313344	0.015313344	0.015313344
Package 40	0.011036131	0.011036131	0.011036131
Package 41	0.005394075	0.005394075	0.005394075
Package 42	0.020206556	0.020206556	0.020206556
Package 43	0.013132476	0.013132476	0.013132476
Package 44	0.018541982	0.018541982	0.018541982
Package 45	0.017751858	0.017751858	0.017751858
Package 46	0.009891607	0.009891607	0.009891607
Package 47	0.001027595	0.001027595	0.001027595
Package 48	0.003597602	0.003597602	0.003597602
Package 49	0.025134103	0.025134103	0.025134103
Package 50	0.009524929	0.009524929	0.009524929
Package 51	0.017096892	0.017096892	0.017096892
Package 52	0.006183472	0.006183472	0.006183472
Package 53	0.004668256	0.004668256	0.004668256
Package 54	0.008314145	0.008314145	0.008314145
Package 55	0.016407392	0.016407392	0.016407392
Package 56	0.019572954	0.019572954	0.019572954
Package 57	0.002823451	0.002823451	0.002823451
Package 58	0.012229744	0.012229744	0.012229744
Package 59	0.019835466	0.019835466	0.019835466
Package 60	0.012608353	0.012608353	0.012608353
Package 61	0.39803802	0.39803802	0.39803802
Package 62	0.008639003	0.008639003	0.008639003
Package 63	0.005644682	0.005644682	0.005644682
Package 64	0.004199264	0.004199264	0.004199264
Package 65	0.017668852	0.017668852	0.017668852

8.4 Average Info Violations Per Line

Package	Avg. Info / Line	10%	20%
Project	0.003916905	0.0043086	0.0051703
Package 1	0.004686697	0.0046867	0.0046867
Package 2	0.002229254	0.0022293	0.0022293
Package 3	0	0	0
Package 4	0.004155861	0.0041559	0.0041559
Package 5	0.000653323	0.0006533	0.0006533
Package 6	0.006388804	0.0063888	0.0063888
Package 7	0.006278027	0.006278	0.006278
Package 8	0.003051363	0.0030514	0.0030514
Package 9	0.000253884	0.0002539	0.0002539
Package 10	0.004010068	0.0040101	0.0040101
Package 11	0.003433086	0.0034331	0.0034331
Package 12	0.001807011	0.001807	0.001807
Package 13	0.000253133	0.0002531	0.0002531
Package 14	0.002660816	0.0026608	0.0026608
Package 15	0.003842658	0.0038427	0.0038427
Package 16	0.003473286	0.0034733	0.0034733
Package 17	0.000814605	0.0008146	0.0008146
Package 18	0.001396135	0.0013961	0.0013961
Package 19	0.000716893	0.0007169	0.0007169
Package 20	0.016357229	0.0163572	0.0163572
Package 21	0.02090209	0.0209021	0.0209021
Package 22	0.000513594	0.0005136	0.0005136
Package 23	0	0	0
Package 24	0.003106555	0.0031066	0.0031066
Package 25	0.010446948	0.0104469	0.0104469
Package 26	0.007266122	0.0072661	0.0072661
Package 27	0.001303327	0.0013033	0.0013033
Package 28	0.001188536	0.0011885	0.0011885
Package 29	0	0	0
Package 30	0.003375476	0.0033755	0.0033755
Package 31	0	0	0
Package 32	0.001229896	0.0012299	0.0012299
Package 33	0.001842254	0.0018423	0.0018423
Package 34	0.00280771	0.0028077	0.0028077
Package 35	0.001019462	0.0010195	0.0010195
Package 36	0.002135142	0.0021351	0.0021351
Package 37	0.001506528	0.0015065	0.0015065
Package 38	6.27E-05	6.27E-05	6.27E-05

Package 39	0.002766697	0.0027667	0.0027667
Package 40	0.001780777	0.0017808	0.0017808
Package 41	0.001621017	0.001621	0.001621
Package 42	0.004400539	0.0044005	0.0044005
Package 43	0.011532382	0.0115324	0.0115324
Package 44	0.001328842	0.0013288	0.0013288
Package 45	0.000480646	0.0004806	0.0004806
Package 46	0.002724634	0.0027246	0.0027246
Package 47	0.00175443	0.0017544	0.0017544
Package 48	0.000266489	0.0002665	0.0002665
Package 49	0.001033654	0.0010337	0.0010337
Package 50	0.016345249	0.0163452	0.0163452
Package 51	0.008968861	0.0089689	0.0089689
Package 52	0.005167616	0.0051676	0.0051676
Package 53	0.001688518	0.0016885	0.0016885
Package 54	0.001248001	0.001248	0.001248
Package 55	0.009674257	0.0096743	0.0096743
Package 56	0.001991188	0.0019912	0.0019912
Package 57	0.003165392	0.0031654	0.0031654
Package 58	0.001921817	0.0019218	0.0019218
Package 59	0.014716636	0.0147166	0.0147166
Package 60	0.002364066	0.0023641	0.0023641
Package 61	0.000965742	0.0009657	0.0009657
Package 62	0	0	0
Package 63	0.001138839	0.0011388	0.0011388
Package 64	0.001725725	0.0017257	0.0017257
Package 65	0	0	0

8.5 Package Defect Frequency

Package	Frequency
Project	n/a
Package 1	2
Package 2	3
Package 3	1
Package 4	3
Package 5	1
Package 6	2
Package 7	1
Package 8	5
Package 9	4
Package 10	3
Package 11	10
Package 12	4
Package 13	3
Package 14	3
Package 15	6
Package 16	3
Package 17	3
Package 18	4
Package 19	14
Package 20	12
Package 21	1
Package 22	2
Package 23	2
Package 24	2
Package 25	2
Package 26	2
Package 27	8
Package 28	2
Package 29	2
Package 30	2
Package 31	2
Package 32	4
Package 33	4
Package 34	2
Package 35	4
Package 36	4
Package 37	1
Package 38	1

Package 39	2
Package 40	3
Package 41	3
Package 42	1
Package 43	4
Package 44	2
Package 45	1
Package 46	1
Package 47	5
Package 48	2
Package 49	5
Package 50	1
Package 51	1
Package 52	3
Package 53	1
Package 54	6
Package 55	9
Package 56	1
Package 57	6
Package 58	3
Package 59	1
Package 60	2
Package 61	3
Package 62	1
Package 63	15
Package 64	1
Package 65	2

9 Glossary

Continuous Integration – The practice of committing small, focused changes to a software project and continuously merging (integrating) these changes with a central source that all developers access.

Defect – An error in source code that causes unexpected behavior in the software.

Git – A distributed revision control system.

Iterative Development – A software design philosophy that emphasizes short iterations of work completion. A single iteration consists of determining requirements of a feature, designing the feature, implementing the feature and testing the feature.

JIRA – An issue tracking system for software development used for bug tracking, feature tracking and project management.

Sonar – A static code analysis tool.

Static Code Analysis - Programmatically scanning source code for potential weak spots and violations.

Technical Debt – The accumulation of defects and other issues in a code base over time.

Violation – In the context static code analysis tools a violation refers to potentially problematic chunks of source code.

10 References

Fowler, Martin. "Continuous Integration." *Martin Fowler.* 2006. http://martinfowler.com/articles/continuousIntegration.html (accessed February 12, 2013).

Gaudin, Olivier. *Evaluate Your Technicla Debt with Sonar.* June 11, 2009. http://www.sonarsource.org/evaluate-your-technical-debt-with-sonar/ (accessed February 12, 2013).

Gaudin, Olivier.. "Personal Communication." September 24, 2012.

Hall, M. A. "Correlation-based Feature Subset Selection for Machine learning." Hamilton, NZ, 1998.

Jenkins CI Community. *About Jenkins CI.* February 12, 2011. http://jenkins-ci.org/content/about-jenkins-ci (accessed February 12, 2013).

Johnson, Stephen C. *Lint, a C Program Checker.* Bell Telephone Laboratories, 1977.

Mark Hall, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutermann. *The WEKA Data Mining Software.* SIGKDD Explorations, 2009.

Parasoft. *Embedded Software Quality Whitepaper.* 2012. http://alm.parasoft.com/embedded-software-vdc-report/ (accessed February 12, 2013).

Royce, Winston W. *Manageing the Development of Larege Software Systems.* IEEE, 1970.

Vassilvitskii, Arthur S. "K-means++: The Advantage of Careful." 2007.

Victor Basili, Dieter Rombach. *The Goal Question Metric Approach.* Encyclopedia of Software Engineering, 1994.