SHARING AND PRESERVING CODING BEST PRACTICES THROUGH PROGRAMMER DATA ANALYTICS

Samiththa Bashani

(168210M)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

June 2018

SHARING AND PRESERVING CODING BEST PRACTICES THROUGH PROGRAMMER DATA ANALYTICS

Jasing Pathiranage Samiththa Bashani

(168210M)

Thesis submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science and Engineering

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

June 2018

DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books)

Signature: ..... Date: ...….....

J.P. Samiththa Bashani

The above candidate has carried out research for the Masters thesis under my supervision.

Name of the supervisor: Dr. Indika Perera

Signature of the supervisor: ..... Date: .....

Abstract

Sharing and preserving coding best practices among the developers are becoming an important objective of software development life cycle. Because violations on coding best practices may lead to catastrophic events which are costly and time consuming. There has been numerous researches done in order to mitigate the issues related to bad coding practices. One of the most challenging tasks towards mitigating this is to identify the skill level of the developers, coding patterns and likelihood for bad coding practices. The widely used methods for this are conducting one on one interview with the developers and review developers work.

This particular research tried to contribute to the field of software architecture by analyzing the feasibility of using machine data to identify the developer coding patterns and related data and provide a mechanism to enhance the skills of a developer. By doing that it makes sure an organization can share and preserve the coding best practices within an organization.

This research focused on developing a parsing mechanism to collect those data from various file formats and types. For this research scope it focused on the static code analysis tool called FindBugs and log data. A successful parser of logs formatted in XML has been developed. A central data storage architecture has been developed in order to capture data from various sources which are different from each other.

Collected data analyzed to generate information about the developers' pattern in doing mistakes and coding styles. To prove that analyzing programmer data for a significant period can predict their abilities and weaknesses an evaluation has been carried out. The evaluation compare data from developer spot interviews with developers' log analyzed data. With those comparisons it identified log data results can match the interview results in an 80% success rate.

ACKNOWLEDGEMENT

I would like to express my profound gratitude and deep regards to my supervisor Dr. Indika Perera for his exemplary guidance, valuable feedback and constant encouragement throughout the duration of the project. His valuable suggestions were of immense help throughout this work. Also I am grateful for the support and advice given by Dr. Malaka Walpola, by encouraging this research.

Finally I would like to thank the academic and nonacademic staff of Department of Computer Science and Engineering and collogue of MSC'16 for the support and encouragement given.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
SCA	Source Code Analyzer
SIG	Software Improvement Group
SDLC	Software Development Life Cycle
SDK	Software Development Kit
IDE	Integrated Development Environment

Chapter 1
INTRODUCTION

1.1 Background

The evolving trends and complex requirements have made software development a much harder task than the earlier days. For the past few decades many software development methodologies, programming paradigms, comprehensive tools and many languages have been emerged in the field of software development to ease these complexities. Unlike the earlier days programmers now have to be up to date with most of these to deliver a quality product. Although the surrounding environment is much more supportive to develop really complex software it is still mostly depends on the level of expertise of the developers.

Many organizations have identified that preserving coding best practices can be a key to develop high complex software system with less negative impacts. Organizations believe apart from having a vast knowledge and hands on experience related to cutting edge technologies and methodologies developers should persist good practices towards software development. By preserving coding best practices and applying them continuously will,

- Reduce the number of unnecessary code churn
- Reduce obvious defects in the code
- Make the code easy to maintain
- Make the code easy to understand
- Reduce the time to market the software
- And many more ...

Many researchers have been carried out to investigate the potential methodologies that would increase the skill level of the developers and hence mitigate the bad coding practices from them. As identifies from those researches few ways that would create a good developer who writes good code is as follows.

- With the time developers gain experience and sharpen their skill sets
- Developers tend to learn by mistakes and after few attempts they tend to correct them by their own. This might not be true for most of the times
- In a collaborative environment developers will receive feedback in order to correct their mistakes. This is depends on the level of the group itself
- Developers might use tools which help them to write a better and a clean code

Writing a clean and stable code is also important as writing a just working piece of code.

Any fool can write code that a computer can understand. Good programmers write code that humans can understand - Martin Fowler, 2008

Though there are numerous tools to support developers to write a better code the impact from them is still questionable. As identified violations on coding best

practices may lead to catastrophic events which are costly and time consuming. This is the main reason that organizations continuously try to enhance their developers by taking necessary actions. One of the most challenging tasks towards this is to identify the skill level of the developers, coding patterns and likelihood for bad coding practices. The most effective known methodology to capture those information is to review the work done by the developer and having a one on one meeting or an interview with the developer.

1.2 Problem Statement

As identified coding best practices play a vital role in today's software development industry. Organizations pay a huge attention to develop the standards and general guidelines and best practices towards successful software development. Yet sharing and preserving coding best practices within an organization is still a challenging task. Most of the time the fact that the developer is the key resource, is unnoticed. Organizations now identified that developing the skill set of the developer itself can make a huge impact on preserving coding best practices.

One of the major steps towards sharing and preserving coding best practices within an organization is to identify the skill level of the developers, coding patterns and likelihood for bad coding practices. The method known to be effective is to analyze the developer and his development tasks.

This might become less feasible due to some factors,

- Organization has a large number of developers and conducting interviews and analyzing each one of them separately is a time and resource consuming task
- The analysis done by the organization is depends on the methodology and the resources used
- The results might not be accurate enough to make decisions

With those and many other factors organizations tend to take generic solutions aiming to mitigate most of the issues related to coding best practices and standards. Few of them are conducting workshops regarding new technologies and best practices, providing tools that help the development and hiring good developers. Those might not be effective since the developers coding patterns; skill levels are different from each other. As stated earlier the best possible solution to identify those would be analyzing each and every one of them separately.

A very first step towards sharing and preserving coding best practices is identifying the skill level of the developers, coding patterns and likelihood for bad coding practices. Current methods like analyzing the developer and his work towards collecting that information becoming less effective. With these barriers and

limitations it has been a harder task to share and preserve the coding best practices effectively within an organization.

1.3 Motivations to Solve the Problem

The main motivation for conducting this research is to find a viable solution to the above mentioned problem. The benefits that can be gifted to the software development industry and the field of software architecture and software engineering are another motivation towards this.

Another factor that motivated this research is that the potential research areas available for this research. Although there has been numerous researches done in the area of code quality, best practices and standards, there are areas where it got less attention. One such area has become the main focus of this research. That is analyzing the feasibility of using machine data left by static code analysis tools to identify the developer coding patterns and related data. Static code analyzing tools has become much popular among developers since those tools support to write better code with less negative effects. These tools generate log file with the code analysis results. Almost all the time these files are left unnoticed since there is no value of them to the developer. Those log files contain valuable data such as the mistakes done by the developer and the distribution of the mistakes with the time. If properly treated these data can turn into valuable information which in turn useful for various aspects.

Though there are several researches conducted focused on static code analysis and create, analysis and compare code analysis tools it seems that there are no researches for monitoring, collecting this data continuously and do analyze this data to provide overall decision about the quality of the programmer. Creating a separate tool/IDE to improve developer's best practices while integrating static code analysis tools to the environments currently used for the implementations is also an emerging area of the industry researches.

1.4 Proposed Solution

This research proposed a solution to mitigate the limitations of the traditional data collecting methodologies used to collect the data related to skill level of the developers, coding patterns and likelihood for bad coding practices. The stated problem is a main barrier for sharing and preserving coding best practices throughout an organization.

The main idea is to analyze the feasibility of using machine data left by static code analysis tools such as FindBugs. Solution includes a parsing mechanism for

collecting data from those log files. A separate storage architecture is also disused to store the collected data from various sources.

The solution also followed by a comprehensive evaluation to prove that it is an effective methodology to replace or augment the traditional data collecting methods.

1.5 Research Objectives

The main objective of this research is to analyze the feasibility of using machine data left by static code analysis tools to identify the developer coding patterns and related data. Towards that goal there are several sub objectives has been set.

As for the first sub objective a comprehensive literature survey was conducted to collect information about the existing methodologies and tools related to coding best practices and preserving them. By doing that few key areas have identified to focus on this research.

Developing a research methodology was another major objective in this research. There it identified the scope, focus areas for the research and suggested an approach towards completing them. This was followed by implementing the methodology for proof of concept. That involved developing an IDE supported tool and data parsing mechanism.

As the final objective an empirical and theoretical evaluation has been carried out to verify this suggested concept and the methodologies can actually provide a viable solution to the considered research problem.

1.6 Overview of the Document

This document contains six chapters which state the major objectives of this research and the overview. The first chapter starts with a description about the background and an introduction to the research. A statement to the underline research problem and the identified solution to that problem is stated along with the motivations.

The second chapter dedicated to the literature survey. This chapter includes the related literature about the coding best practices and preserving them, the existing methodologies, tools along with the evaluation details of them. The literature survey finding was used to propose a solution towards the identified research problem. This chapter is followed by the chapter dedicated to the methodology, which identified the relevant solutions architecture to solve this problem. The third chapter contains an overview of the proposed solution. The methodologies and tools to be used and the

proposed architectures for the proof of concept systems are also included in this chapter.

The solution implementation details are included in the next chapter. This chapter comprised of comprehensive information about implementing a proof of concept solution system to the underline research problem. Chapter also contains details about the considered methodologies, tools and resources. The evaluation of the implemented solution was done in the next chapter. This chapter comprised of the details related to the theoretical and empirical evaluation of the suggested solution architecture. This research document concludes with the chapter dedicated to the conclusions. The conclusion chapter discuss about the research contribution and the limitations. The findings of the research and the results of the evaluations are discussed. Finally it states the opportunities to enhance this research idea as its future work.

Chapter 2
LITERATURE REVIEW

2.1 Code Quality

Quality is one of the key attributes of ensuring high standards of code. There are so many ways to implement a one code in software development. There are many opinions about how to makes good high quality code. What one developer thinks about code quality may be very different than what another developer thinks.[1]

Generally Code quality is a loose approximation of how long-term useful and long-term maintainable the code is. So the High quality can be recognized as the code that is being carried over from product to product, developed further, maybe even open sourced after establishing its value.

So quality code consist of

- Clear and understandable design and implementation.
- follows a coding standard, uses linting
- Well defined interfaces.
- Ease of build and use.
- Ease of extensibility.
- Minimum extra dependencies.
- Tests and examples.
- self-explaining code.
- Up to date means to contact the developer.

If an organization doesn't consider of code quality the associated cost and risk of the code will prove to be difficult to manage. [2]

Quality code provides so many benefits for an organization. Some of them are as follows.

- improves Faster times to market
- Decreases technical debt
- Elimination of Risks and failures
- Regains control of legacy systems
- Reusability
- Improve issue handling performance [3]
- Improve maintainability of the system
- Reduce time to understand to other developer
- Faster defect resolution

As well quality of the code has been used in many researches for evaluate the quality of the whole system.

2.1.1 Code Quality to Measure Quality of the System

As maintainability of the software also depend on the quality of the code the research. The Software Improvement Group, or SIG who is an Amsterdam-based consultancy firm specialized in quantitative assessments of software portfolios has developed a model for assessing the maintainability of software by Conceiving maintainability as a function of code quality.[4] The SIG quality model (SIG QM) defines source code metrics and maps these metrics to the quality characteristics of ISO/IEC 9126 that are related to maintainability.

In a first step, source code metrics are used to collect facts about a software system. Following six source code properties are used as key metrics for the quality assessments.

- Volume
  If the code is large, the more effort need maintain since there is more information to be taken into account
- Redundancy
  Duplicated codes has to be maintained in all places where they exist
- Unit size
  Units as the lowest level piece of functionality should be kept small. It make easier to focused and understand
- Complexity
  Simple codes are easier to comprehend, maintain and test than complex ones
- Unit interface
  Size units with many parameters can be a symptom of bad encapsulation
- Coupling
  Tightly coupled components are more resistant to change

The measured values are combined and aggregated to provide information on properties at the level of the entire system. These properties are then mapped onto the ISO/IEC 9126 standard quality characteristics. Figure 2-1 depicts that mapping process.

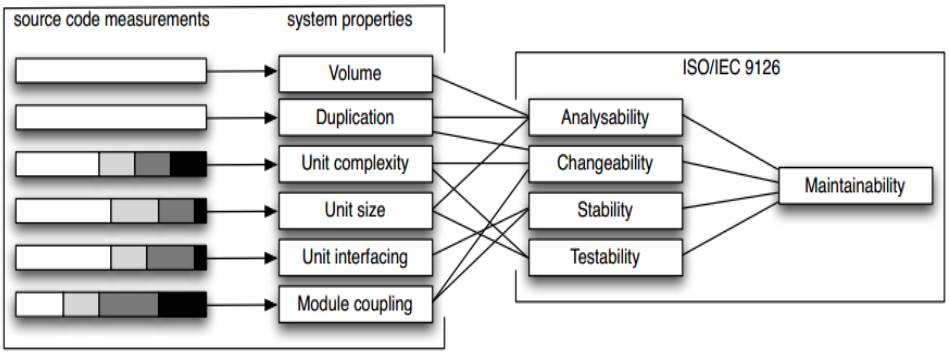

Figure 2-1: The SIG Quality Model maps source code measurements onto ISO/IEC 9126 quality characteristics

In the second step, after the measurements are obtained from the source code, the system-level properties are converted from metric-values into ratings. This conversion is performed through benchmarking and relies on the database that SIG possesses and curates. Using this database of systems, the SIGQM model is calibrated so that the metric values are converted into star ratings that reflect the system's performance in comparison with the benchmark. This process results in a system getting attributed 1 to 5 stars.

2.1.2 Improving Code Quality

As the code quality is so important in software developments there are so many researches to improve the quality of code in various ways. Some of them are focused on improving code quality by identifying the code as good or bad code while some others focused on improving code quality by improving the attributes that affect to the quality of the code [28]. As well some of researches use the real time programming environment while others implemented separate environment to evaluate code i.e the developer has participate to this application on purpose. There are so many researchers have been conducted to improve the code quality in open source software development [35]. The different types, techniques and methodologies are used in those researches.

2.1.3 Improving Code Quality Using Gamification

Games have been exist and improved since thousands of years as the main purpose is to entertaining people and create pleasure. People nowadays are dedicating a great amount of their time to video games [6].

Salen & Zimmerman define a game as “a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome” though unfortunately most common games are considered to be unproductive, with no valuable outcomes. With gamification these gaming concepts and elements can be used to get a more valuable and productive outcome. It is possible to engage users in solving real-world problems, which activities and outcomes are value-adding and not just a waste of time with gamification. Many researches show that when people use gamification they are likely to be more productive, resulting in higher quality outcomes. It can be used to influence people’s behavior and improve their motivation and engagement.[7]

Gamification has been applied in many different domains in the last years. One domain is education and training domain. There game elements are used to increase the motivation, engagement and performance of the students. Gamification has also been central part of the design of many mobile applications for smart phones and tablets. It helps to achieve stronger user engagement and diffusion of the mobile applications. Corporate websites oriented toward customers have also been the object of gamification as they seek to improve the customer experience on the website [9].

Zichermann &Cunningham [8] define gamification as “the use of game thinking and game mechanics in non-game contexts”. There is a two dimensional model to distinguish games and gamification. The main difference is that games are just for fun and entertaining for the players while gamification has a certain purpose, without any game play. . Figure 2-2 shows the different types of game thinking and the place where the gamification is classified.

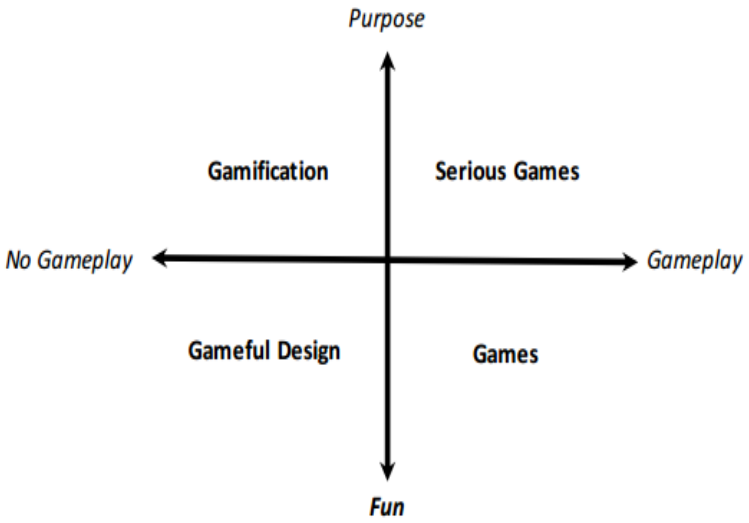

Figure 2-2: Types of Game thinking

In the paper "Improving the Quality of the Software Development Lifecycle with gamification" Philipp Lombriser and Roald van der Valk describe how glamification use in software engineering to create A successful system which increase software engineers' work performance and motivation, resulting in higher software quality, with less changes and defects.

This paper focused on gamifying all the process of the SDLC which are 'requirements definition', 'system and software design', 'implementation and unit testing', 'integration and system testing' and 'operations and maintenance'.

- Requirements Definition

  This is a good example for using gamification in requirement engineering. The authors analyzed several requirement elicitation practices and then developed a system with the appropriate game elements. This too developed based on the theory of the six thinking hats[11] which stimulate parallel thinking in teams in order to be more effective and avoid team conflicts. In ithink these methods are used to new requirements and discus existing ones. The main game elements used for ithink are points and progress bars. A user of this tool can earn points by submitting a new requirement, rating a requirement, commenting on a requirement and completing a discussion of a requirement.
- Testing

  The expectation of organizations from the testers is to find as many bugs and defects as possible and reports them properly. But for lot of people this task might seem boring. However, game elements can make testing more entertaining, just as with other applications. Much of what testers do can be thought of in gaming terms. Microsoft has developed a 'Language Quality Game' (LQG) to improve the quality of their products and increase productivity at the same time. The standard business process at Microsoft uses professional language translators for the textual content of the Windows operating system. In a second step a quality assurance team is responsible to verify the translation. In some countries it was very difficult to find enough people to translate and review the dialog boxes. To overcome this problem, LQG was established for the review process step. Native speaking citizens around the world can voluntary participate in the game and review Windows' dialog boxes. In this game, users are awarded points for every mistranslation found and then ranked on a public leader board.
- Software development
  - In the coding domain, there are many educational web-based platforms exist which are targeting students and teachers to learn and

advance their programming skills. Code hunt is such platform which gamification to encourage players to learn how to code. As well gamification can also be used to solve real business cases in software development. For example In 2012 visual studio introduced a new plug-in, bringing achievement badges to the development phase using gamification.

Although the most of the phases of the software development process have already been gamified these phases are still separated and not yet cross-integrated into one gamified environment. Fecher did a first step in this direction, by creating a conceptual proposal of an interconnected gamified process using a multi-layer architecture containing different game elements. The author investigated several game elements and analyzed how they can be utilized to make software building more productive and fun. Figure 2-3 depicts the conceptual multilevel architecture proposed by Fetcher.

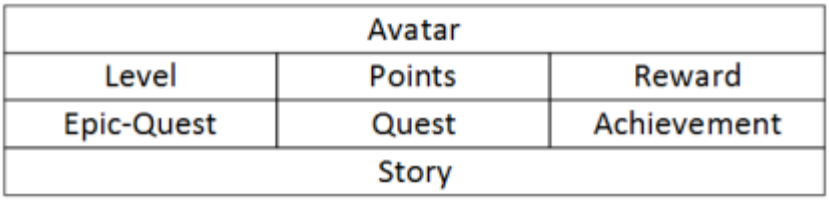

Figure 2-3: Conceptual architecture for gamification in SDLC

As described above in the software development sector there are some researchers conducted to improve code quality using gamification.

In Code Hunt tool provide different platform to developers to improve their coding quality and ability by providing gaming platform to solve puzzles by coding.[12] As the code develops, the game engine gives custom progress feedback to the player. It is part of the game play that the player learns more about the nature of the goal algorithm from the progress feedbacks. So with this feedback the player can learn coding very effectively. Every puzzle must be solved with a piece of written code that is verified by the system. If the code is wrong, the system returns an error. If the code is correct, the player wins the quest and continues to the next level. As the player keeps playing, his skill rating goes up and the quests get more difficult, thereby maintaining 'flow'. This example of puzzle solving for educational purpose indirectly improve the coding ability and coding quality of a developer.

Gamification is also be used to real time coding. In 2012 visual studio introduced a new plug-in, bringing achievement badges to the development phase [13]. With

Visual Studio Achievements, developers can unlock badges (figure 2-4) based on their activity doing in the development in visual studio. Figure 2-5 shows the categories of these badges. Unlocking achievements is done in the background during the compilation of the program. When certain criteria or actions are detected, the plug-in triggers a pop-up alert and award a new badge, which is then displayed on the public leader board and the developer's Channel 9 profile. New obtained badges pop up in the lower right corner of the development environment to visualize the achievements as shown in Figure 2- 4. Furthermore, the personal player profile is updated and the player's position in the leader board is re-ranked. Badges can also be shared via Facebook so developer can impress others (leads, management) about how well at the job.

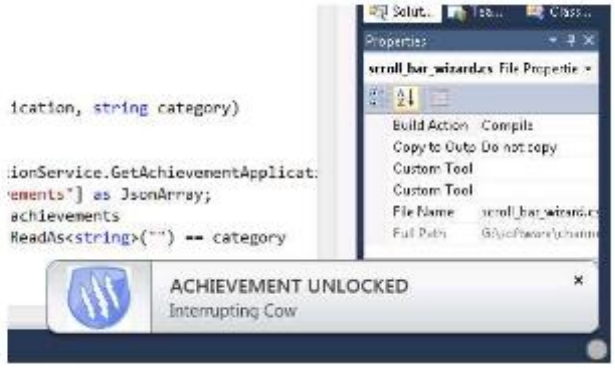

Figure 2-4: Badge pop up

Figure 2-5: Six categories of achievements

2.1.4 Improving Code Quality Using Code Reviews and Analysis

Code reviews and analysis is very common and widely used technique in checking quality of the code. This technique is widely used in many researchers in varies ways

in improving code quality. This area is important to be discussed details. So this topic will be covered in the section 2.3 of this report.

2.2 Coding Best Practices

Coding best practices are set of rules and standards that the software development community has established to improve and maintain the quality of software by using their experience and the things have been learned over the time.

Most of the software remains in use for longer time. As well some software are developed longer time adding and improving functionalities. Most of the time original developers do not work in the same project this much longer time. So rules and standard are needed to facilitate both initial development and subsequent maintenance and enhancement by people other than the original authors. As well these rules are very important in projects where the projects involve more than one programmer.

For example it is easier programmer to read code written by some other programmer if the code follows the coding standard and best practices.

So most of the organizations and software communities establish coding standards which includes programming guidelines, best practices, programming styles and conventions. Some of the standards are specialized to an organization and some standards are specialized to a language. There are different conventions for different programming languages.

2.2.1 Importance of Best Coding Practices

Coding best practices play a vital role in code quality. Following coding standards and best practices not only helps to improve the quality of the code but also the quality of the overall software system.

The good coding standard should preserve the consistency. So the rules and guidelines of the standard should not be contradicted. So completed source code which is implemented by following a coding standard should reflect harmonized style as if a single developer wrote the code in one session.

Using coding standards leads to a more readable source code. The more readable source code is, the easier it is for someone to maintain. So a new programmer can go into code and figure out what's going on easily to help with maintenance or new development.

As well using coding standards and best practices it becomes easier to find and correct bugs.

It will give a better view of how that code fits within the larger application. This clear view helps to improve potential for reuse the codes. It leads to do a significant affect on a cost.

The sense of ownership increases as the application becomes more stable and the code becomes easier to maintain with coding standards. The higher the sense of ownership leads to make better feeling in developer. The better the developer feels about his skills, the better the code becomes.

As well code quality and quality software process very important in Software Acquisition for a software company to winning bit as proposed development activities from contractors commonly list includes the activities which measures the quality of the code such as the number of and type of reviews, The use of automated syntax analysis tools and adherence to the rules incorporated by them, The use of automated unit testing including test coverage requirements and etc. [14]

Areas typically covered in coding standards.

- Program Design
- Naming Conventions
- Formatting Conventions
- Documentation
- Possibly Even Licensing

2.3 Code Review and Code Analysis

Code quality is becoming more important with the increasing reliance on software systems. There are different ways to ensure quality in code. The use of analytical methods to review source code in order to correct implementation bugs has become one of main approach to ensure quality in code.

In the past there was no conscience on how necessary and effective a review. In 1970's, formal review and inspections were recognized as important to productivity and product quality, and thus were adopted by development projects. This new approach removes the defects of the software at the early stages of the development process which lead to produce more reliable and efficient programs.

Among analytics method code review, static code analysis and dynamic code analysis is the very common method used in most of the software company. Code review is when a colleague/mentor/professor/friend goes over your code and gives you constructive criticism while Static analysis is an automated process in which a machine analyzing the program with what it knows about the language and tries to pick out things that could be incorrect, inefficient, poor style, or otherwise suboptimal. The static analysis approach is meant to review the source code, checking the compliance of specific rules, usage of arguments and so forth; the dynamic approach is essentially executing the code, running the program and dynamically checking for inconsistencies of the given results.

2.3.1 Static Code Analysis

Static code analysis (source code analysis) is a software verification technique refers to the process of examining the code without executing it in order to capture the defects in the code early avoiding costly later fixations.[15]. Static code analysis gives an opportunity to check the quality of source code without executing the program. Static code analysis has two main approaches: manual and automated. Manual approaches involve human subjects performing the process of reviewing the code and capturing defects, while in automated approach; computer-based tools are used to detect defects.

2.3.2 Importance and Benefits of Static Code Analysis

Source code analysis is very important in many ways [34]. From decades source code analysis has grown [16]. The paper V M. Harman, "Why Source Code Analysis and Manipulation Will Always Be Important,"[16] describes the importance of source code analysis descriptively.

- Lead to Reliable, readable Code

  One of the advantages of the static analysis approach during development is that the code is forcefully directed in a way as to be reliable, readable and lees prone to errors on future tests. This also influences the verification of the code after it is ready, reducing the number of problems found in further implementations that code
- Reduced development costs

  Multiple studies have shown that using static analysis products enables development teams to quickly and easily find code defects early in the development lifecycle, when the cost of fixing defects is lowest. Using Rational Software Analyzer, developers can analyze code and discover issues related to general code quality problems such as calling standard functions in the wrong order as the code is being developed, before the entire system is created. When defects are discovered at this early phase in the development lifecycle, the number of defects in the testing and debugging phase is reduced helping to save money and time. [17]
- Greater control over the quality of outsourced code

  It is very important to check and validate a outsourced code before it is introduced into a larger application or product as there can be many problem occurred if it is not suit with the standards of the code of the existence system
- Increased speed to market

  Static analysis allows developers to test for code quality before applications get to your QA team and decreases the amount of time spent identifying and fixing bugs. By identifying and correcting defects early in the lifecycle, we can potentially reduce your time to market and sharpen the competitive edge and the large number of benefits they expect to achieve from that suite of tools they mentioned.

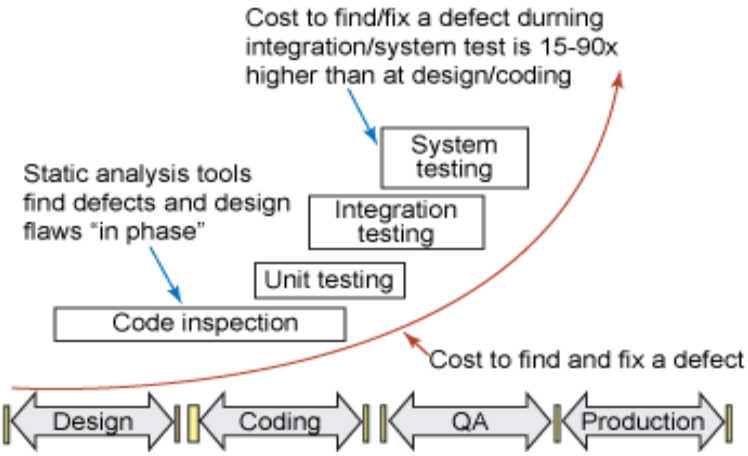

Figure 2-6: Comparison of costs to fix defects found at different stages of the development lifecycle

2.3.3 Manual Review

This form of static analysis is mostly used in software organizations. It is very time consuming and to perform it effectively the reviewers must have a good knowledge about the coding standards and best practices as well they must know what type of errors there are supposed to find before they can rigorously examine the code.

Basically static code analysis conducted by humans can be divided in to two categories. They are self reviews and 3rd party reviews.

Manual approaches are conducted in both formal and informal manners. The formal reviews follow a formal process that is well defined, structured and regulated. The informal reviews refer to examine software artifacts to detect defects without a prescribed process.

Figure 2-7 shows the Flow of types of reviews such that increasing the formality. [18]

- Self Review
  The programmer tries to evaluate and correct by himself the code he implemented
- Walkthrough
  Focuses on the presentation to an audience of the code in question by its programmer
- Peer Review
  The programmer presents his code to a colleague to review
- Inspection and Audit

- Done by a third party of evaluators, the audit being the highest formal review

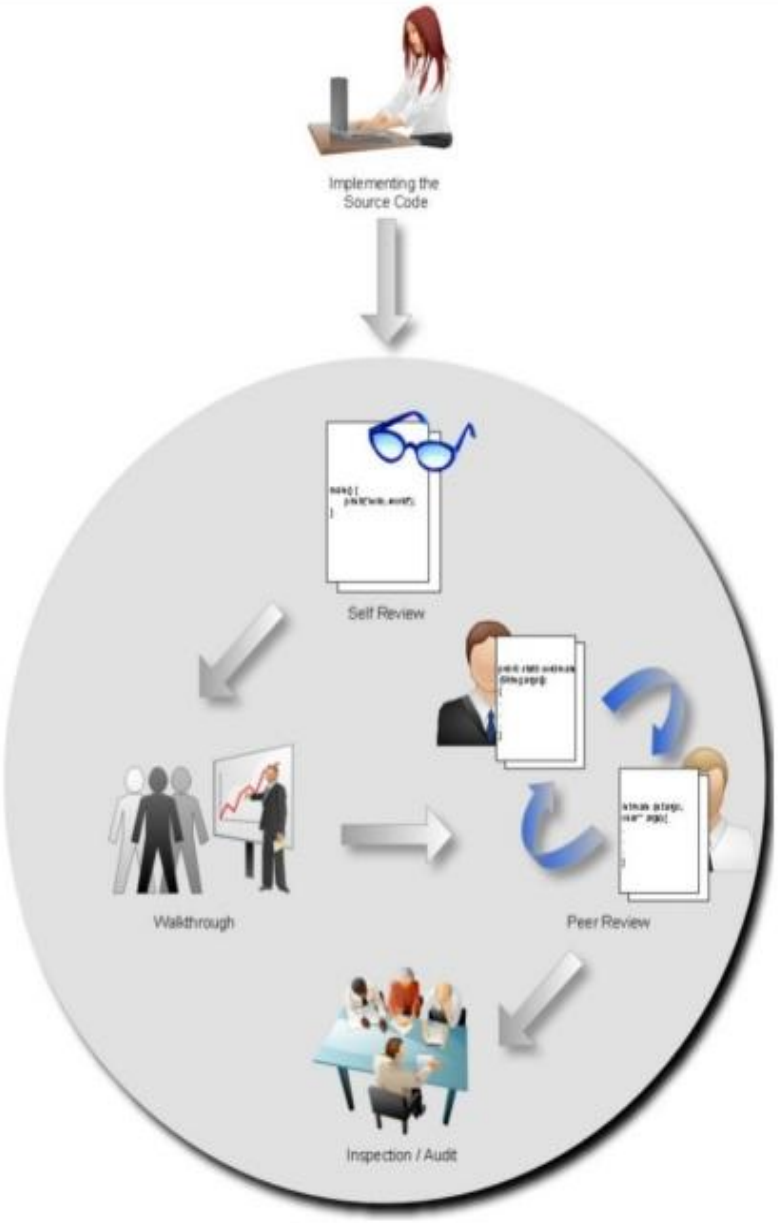

Figure 2-7: Flow of types of reviews that increase formality.

2.3.4 Automated Static Code Analysis

There are a variety of ways to perform automatic static analyses [18], including at the developer's request, continuously while creating the software in a development environment, and just before the software is committed to a version control system.

In the paper V M. Harman, "Why Source Code Analysis and Manipulation Will Always Be Important**,"[**16] it is mentioned that "In the future, we may hope that a software engineer would be provided with a suite of tools for analysis and manipulation of source code that encourages and facilitates this kind of exploratory investigation". The large number of benefits they expect to achieve from that suite of tools they mentioned. In Present the expectations becomes real with all the benefits with source code analysis tools.

2.3.5 Source Code Analysis Tools

Source code analysis tool created to analyze source code and compiled version of the code without or nearly without participation of the human being. These tools help to improve security and reliability of the code. Using static analysis tools for automating code inspections can be beneficial for software engineers. Static analysis tools for automating code inspections can be beneficial for software engineers. Such tools can make finding bugs, or software defects, faster and cheaper than manual inspections.

Now days there are so many source code analysis tools which are moved in to the IDE. They are design for the problems that can be detected during the software development phase itself, this is a powerful phase within the development life cycle to employ such tools, as it provides immediate feedback to the developer on issues they might be introducing into the code during code development itself. This immediate feedback is very useful, especially when compared to finding vulnerabilities much later in the development cycle.

Some tools are designed for a specific language some are for multi languages. Few of them which are commonly used are as follows [36].

Table 2-1: Static code analysis tools

Language	Tool	Description
.NET Tools	.NET Compiler Platform (Codename Roslyn)	Open-source compiler framework for C# and Visual Basic .NET developed by Microsoft .NET. Provides an API for analyzing and manipulating syntax.
	CodeIt.Right	Combines static code analysis and automatic refactoring to best practices which allows automatic correction of code errors and violations; supports C# and VB.NET.
	Resharper	a popular developer productivity extension for Microsoft Visual Studio. It automates most of what can be automated in your coding routines.
C, C++	Eclipse (software)	An open-source IDE that includes a static code analyzer (CODAN).
	Cppcheck	Open-source tool that checks for several types of errors, including use of STL.
Java	IntelliJ IDEA	Cross-platform IDE with own set of several hundred code inspections available for analyzing code on-the-fly in the editor and bulk analysis of the whole project.
	SourceMeter	A static analysis tool focused on finding concurrency bugs
	FindBug	Based on Jakarta BCEL from the University of Maryland.
Multi-language	SonarQube	A continuous inspection engine to manage the technical debt: unit tests, complexity, duplication, design, comments, coding standards and potential problems. Supports languages: ABAP, Android (Java), C, C++, CSS, Objective-C, COBOL, C#, Flex, Forms, Groovy, Java, JavaScript, Natural, PHP, PL/SQL, Swift, Visual Basic 6, Web, XML, Python.
	SourceMeter	A platform-independent, command-line static source code analyzer for Java, C, C++, RPG IV (AS/400) and Python.

As well there are so many research conducted to develop new code analysis tools, analyze code analysis tools as well there use.

In the paper "Improving Software Quality with Static Analysis" some new tools are introduces for different kind of purposes of finding bugs in different platforms.[5].These tools scan software for bug patterns or show that the software is free from a particular class of defects. Tools newly implemented in that research is as follows.

- Find bugs

  This is a static analysis tool that finds coding mistakes or defects in Java programs. The approach they conducted here is abstract bug patterns from the real source code and finds the simplest possible detector that can effectively identify that bug pattern using test cases for that bug pattern and using source codes which have large number of lines. Other important thing of find bugs is that it can run within multiple environments. As well the persistence of defects across successive builds of a software project can be captured as it supports for historical tracking. It is an open source tool which can be lead to allow new defect detection research to be started.

- Saffire

  Though there are many software written in multiple languages only there few research for detecting bugs in that software. Saffire is for such software. . Most multilanguage programs are built using foreign function interfaces (FFIs), which define what must be done to translate between native and foreign data representations and how the foreign code should safely interact with the native code. These requirements can lead to large number of mistakes and hard-to-find bugs. So saffire is implemented to check type safety across an FFI. Saffire supports for the OCaml-to-C FFI and the Java-to-C FFI. The analysis of this tool is very fast most of the time it takes one second with a low positive rate.

- Pistachio

  Now a day's most of the software systems communicate over internet using various protocols. The software that implements these protocols may still contain mistakes, and an incorrect implementation which can lead to vulnerabilities even in the well-understood protocol. So this tool helps to reduce it by checking that a C implementation of a protocol matches its description in an RFC or similar standards document. The first component of Pistachio is a rule-based specification language that is tuned to describing network protocols while the second part is a symbolic evaluation engine that simulates the execution of program source code.

- LockSmith

  This is implemented to prove that multi-threaded C programs are free from data races. It uses the well known “guarded-by” pattern in which each thread that accesses a memory location must do so while holding the lock that “guards” that location to analyze the racing conditions.
- CMod

  This is a tool that provides a sound, backward-compatible module system for C using a set of four rules that are based on principles of modular reasoning and on current programming practice. As well there are several researches which looked into using of the Static Analysis Tools among software developers. Though there are several analysis tools available some developers are companies are hesitate to use these tools and tend to use manual reviews. Tool Output, Collaboration, Customizability, Result Understandability are some of the facts users considered in to use or not to use a code analysis tool.

Source code analysis and collaborative tools are topical and efficient instruments for software developers. Better understanding of SCA tools might help to improve user’s code and increase his/her knowledge, which will lead to better project outcome. [19]

2.3.6 FindBugs

FindBugs is an open source static code analyzer which is implemented to detect possible bugs in java programs. It can be installed either as a plug-into Eclipse (from version 3.3 onwards) or as a standalone application with a Swing interface. The Eclipse installation is handled by the Eclipse plug-in manager. The standalone application contains a jar file with launchers both for Linux and Windows. The analysis engine reports nearly 300 different bug patterns. Findbugs has a plug-in architecture in which new bug detectors can be defined. They may report different bug patterns [33]. Find bugs detectors are simply written in java using a variety of techniques instead of using a pattern language for describing bugs. Many simple detectors are implemented using the visitor design pattern. In visitor design pattern each detector visits each class of the analyzed library or application. Each detector can be categorized to one or more of the following categories. [20]

- Single-threaded correctness issue
- Thread/synchronization correctness issue
- Performance issue
- Security and vulnerability to malicious un trusted code

The implementation strategies of the detectors can be divided into below rough categories [20]

- Class structure and inheritance hierarchy only

  Some of the detectors look at the structure the analyzed classes without looking at the code
- Linear code scan

  These detectors make a linear scan through the byte code for the methods of analyzed classes, using the visited instructions to drive a state machine. These detectors do not make use of complete control flow information. However, heuristics (such as identifying the targets of branch instructions) can be effective in approximating the effects of control flow
- Control sensitive

  These detectors make use of an accurate control flow graph for analyzed methods
- Dataflow

  The most complicated detectors use dataflow analysis to take both control and data flow into account. Eg: the null pointer dereference detector.

FindBugs does not perform inter procedural context sensitive analysis. But many detectors use global information such as subtype relationships and which fields are accessed across the entire application. Some few detectors use inters procedural summary information. For example which method parameters are always dereferencing.

Bug patterns are error-prone coding practices that arise from the use of erroneous design patterns, misunderstanding of language semantics, violating coding standards or simple and common mistakes.

In Find bugs each and every bug is categorized in to a category like correctness, bad practice, performance, internationalization and etc [29]. As well each and every report of a bug pattern is assigned a priority (high, medium or low). The priorities are determined by heuristics unique to each detector/pattern. It is not necessarily comparable across bug patterns. Those low priority warnings are not report in FindBugs in normal operations.

Reports can be exported to XML files, including classification of individual hits as “not a bug”, “I will fix” and etc.

2.3.7 Static Code Analysis for Other Researches

Though there are so many researchers have been conducted about source code analysis, its benefits, its importance, to create source code analysis tools and to analyze those tools in varies fields just some few researches used static code analysis collaborate with other techniques.

2.3.8 Static Code Analysis with Classification Methods

There are so many researchers conducted to investigate new paths and new frameworks to investigate new paths for researches with collaboration of code analysis tools and other techniques.

The paper Towards a Collaborative Code Review Plug-in [19] and Predicting Source Code Quality with Static Analysis and Machine Learning,[21] to use source code analysis with classification methods to identify source code quality.

Theses researches are for investigate the feasibility of a plug-in facilitating identification of code quality and for investigate what extent a tool facilitates identification of well written and badly written source code improves the code quality.

Though there are many tools for peer and machine code analysis most of them do not support for both. In the paper Towards a Collaborative Code Review Plug-in [19] they proposed a framework for identification of well written and badly written source code performed by a combination of human reviews, static analysis and classification methods. Naive Bayes classification is used as classification method here as it is relatively easy to use in comparison with other classification methods, but still gives a reliable result. A plug-in that facilitates identification of well written code and design for a framework that use static analysis and classification methods is given as outcome of this research.

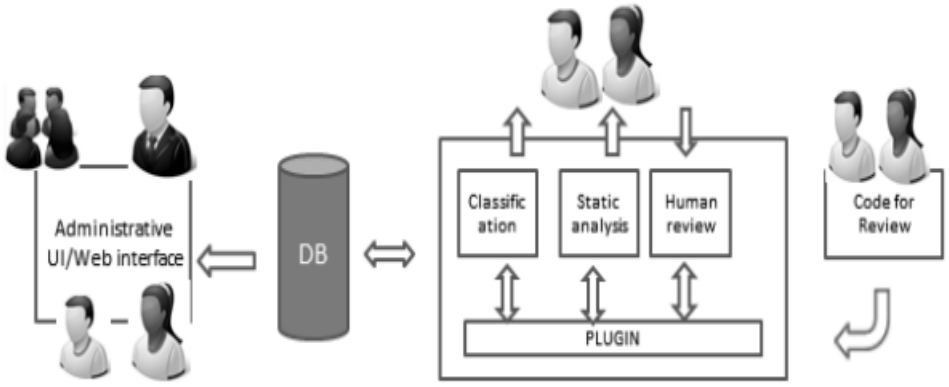

Figure 2-8: Architectural overview of the Collaborative Code Review Plug-in

In this research JHawk is used to generate the static analysis for the source code, and some metrics are generated by the plugin. The plug-in offers peer code review, and the implementation in Python the automated classification of source code quality. Figure 2-8 shows an overview of the architecture of the approach.

The approach uses a MySQL database for storing the collected training data and the results from static analysis. Some of the static analysis is implemented by theologian; the rest is performed using JHawk. The classification is implemented in Python, the selected methods are k nearest neighbor (KNN), na¨ıve Bayes (NB) and decision tree (DTree) in this research.

The plug-in extracts features based on static analysis. The source code to be analyzed is given as an input to JHawk by command line, and a text file containing the static metrics is generated. These metrics are stored in the MySQL database together with a textual representation of the source code to be analyzed. The classification is based on this metrics together with some metrics generated by the plug-in. Before the metrics are stored in the database, the metrics are converted to discrete values.

Figure 2-9 depicts the classification process used to classify as well written or badly written code.

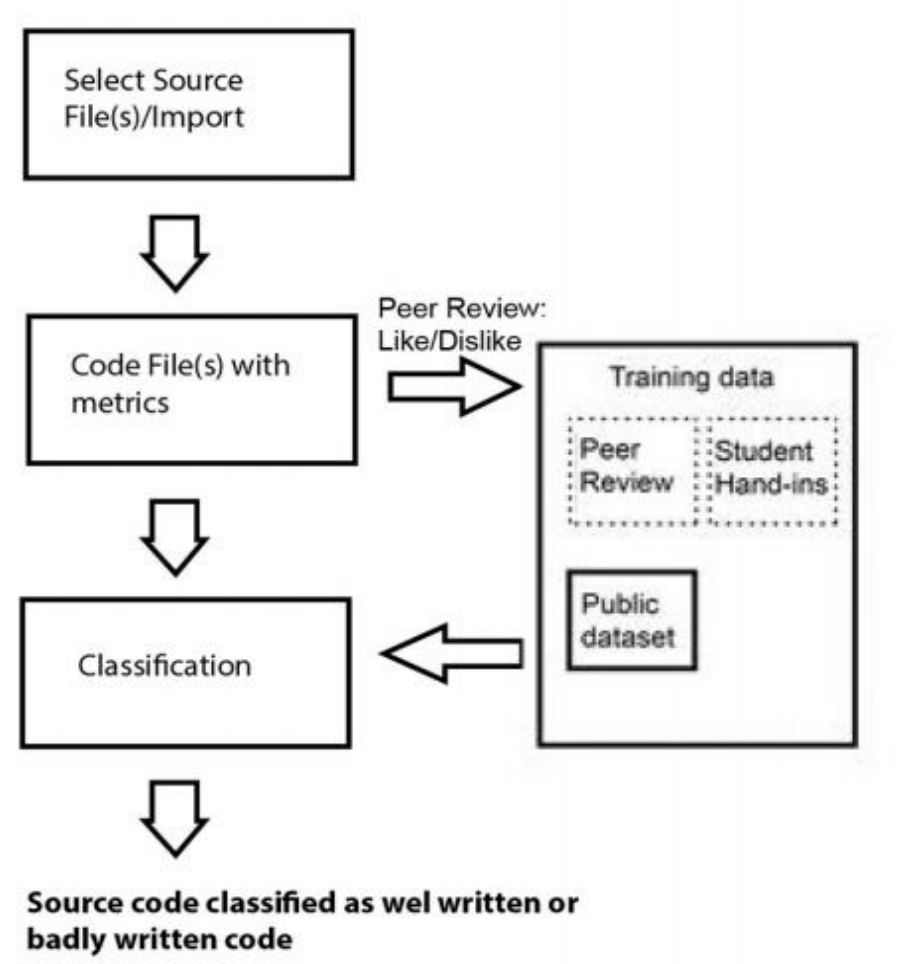

Figure 2-9: Classification Process

2.3.9 Source Code Analysis to Remove Security Vulnerabilities in Java Socket Programs

In the research "source code analysis to remove security vulnerabilities in java socket programs: A case study" the focus is testing for software security using source code analysis. Generally both static and dynamic analysis is used to test for software security. [22]While dynamic code analysis is mainly used to test for logical errors and stress test the software by running it in an environment with limited resources, static or source code analysis has been the principal means to evaluate the software with respect to functional, semantic and structural issues including, but not limited to, type checking, style checking, program verification, property checking and bug finding. Static code analysis helps to identify the potential threats and vulnerabilities associated with the software, analyze the complexity involved and the impact on user experiences in fixing these vulnerabilities through appropriate security controls. Static code analysis also facilitates evaluating the risks involved in only mitigating or

just leaving these vulnerabilities unattended – thus, leading to an attack, the consequences of such attacks and the cost of developing security controls and integrating them to the software after the attack has occurred. In addition, static code analysis is also used to analyze the impact of the design and the use of the underlying platform and technologies on the security of the software.

For this research they used Fortify SCA to analysis the source code. The Fortify SCA can be used to conduct static code analysis on C/C++ or Java code. It can be run in Windows, Linux or Mac platforms. The SCA can analyze individual program files or entire projects collectively.

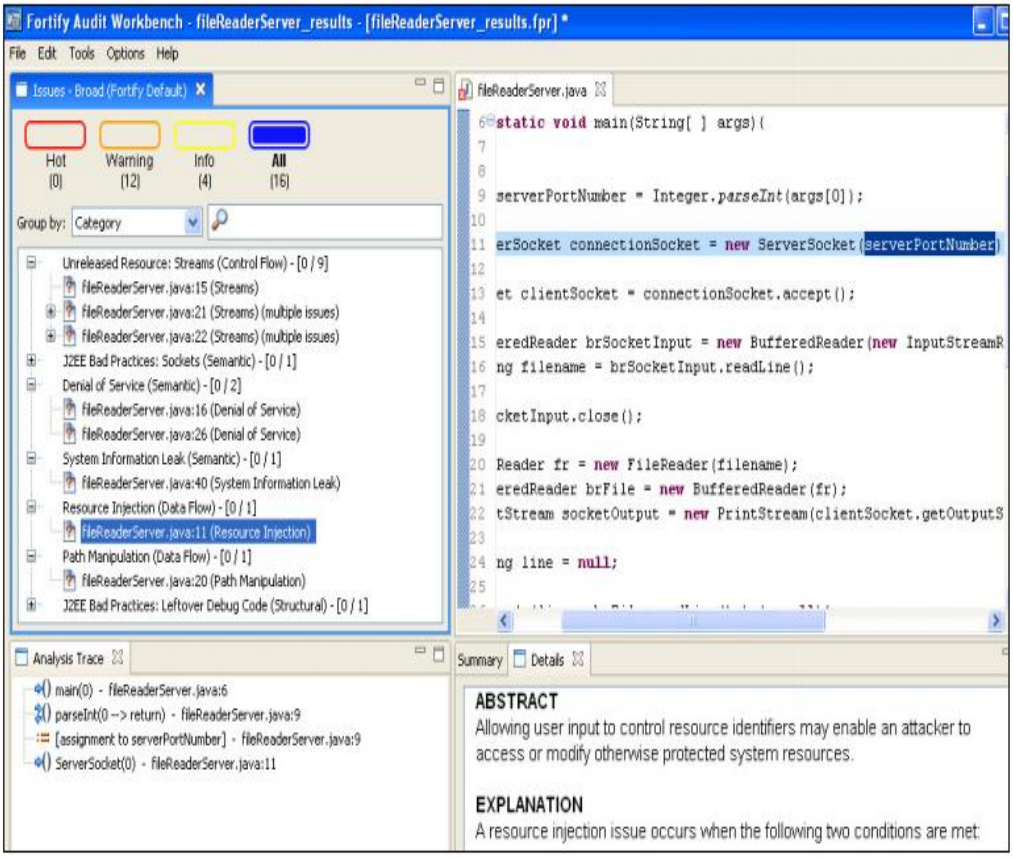

Figure 2-10: Issues Panel and Code Editor displaying Details of a Specific Security Issue.

As a case study a file reader server socket program developed in Java is analyzed by above static tool to illustrate the identification, impact analysis and solutions to remove five important software security vulnerabilities, which if left unattended could severely impact the server running the software and also the network hosting the server. Five security vulnerabilities they discussed in this research are as follows.

- Resource Injection Vulnerability

  Arises because of the functionality to let the user (typically the administrator) starting the server program to open the server socket on any port number of his choice

- Path Manipulation

  Occurs when user input is directly embedded to the program statements thereby allowing the user to directly control paths employed in file system operations

- System Information Leak

  Revealing critical system data, program structure including call stack or debugging information that may help an adversary to learn about the software and the system, and form a plan of attack

- Denial of Service

  The one with which an attacker can cause the program to crash or make it unavailable to legitimate users

- Unreleased Resource

  Occurs if the program has been coded in such a way that it can potentially fail to release a system resource

2.4 Eclipse IDE

Eclipse is a main development IDe for java developers. It is not only for java language It is a multi language development IDE that support for Ada, ABAP, C, C++, COBOL, D, Fortran, Haskell, JavaScript, PHP through the use of plugins. [23] It can also be used to develop documents with LaTeX and packages for the software Mathematics.

Eclipse SDK is free and open source software and it is released under the Eclipse Public License. The eclipse open source community is much bigger in whole around the world. Currently it consists of more than 150 projects covering different aspects of software development.

2.4.1 Eclipse Plug-ins

A plug-in is an eclipse component that allows developer to customize and extend the IDE with additional functionalities. Extensions are supposed to simplify the development process and give additional possibilities for the developer.

Eclipse applications use a runtime based on a specification called OSGi. A software component in OSGi is called a bundle. An OSGi bundle is also always an Eclipse plug-in. Both terms can be used interchangeably.

The eclipse IDE is a Rich client platform application developed to support development activities. Even some core functionalities of the eclipse ide are provided through plug-ins. Java development tools and C development tools are best examples for above statement as they are core functionalities of the ide and are contributed as a set of plug-ins. the Java or C development capabilities are present in eclipse IDE only if these plug-ins are installed. The Eclipse IDE functionality is heavily based on the concept of extensions and extension points.

By using eclipse plug-in any developer can extend existing functionality or can create completely new tool or programming environment.

So there are so many researches that have been implemented new tools and new environment as eclipse plug-ins. One of them is the research of the Automatic and Collaborative Code Review Plug-in that shows possibility to implement a plug-in which facilitates identification of well written and badly written code. Here they implemented a eclipse plug-in to automatically identify well written and badly written code by using static analysis and classification. [19] “Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm” research is also developed based on the eclipse environment [32].

Chapter 3
METHODOLOGY

Current software development methodologies hardly give an opportunity for a developer to actually enhance his skill set while participating in the software development process. From an organizations' point of view there might not be a dedicated process to enhance the skill level of the developer or share and preserve the coding best practices. It would be much more effective and practical if there is a way to improve the developer and complete the development in parallel without altering the actual development process.

This research will be conducted in order to find the feasibility of using programmer specific user data to improve the programmer itself and by doing that share and preserve the coding best practices among the developers. There are very few occasions where an organization preserve and analyze programmer data. Here programmer data is the kind of data that a static code analysis tool generates while doing an development in a local developer environment, and IDE generated data about the application. Most of the time these data are left untouched, but it silently collects a huge pile of information about the developer. This research considers those data as the main asset and uses those data to find a viable solution for the above mentioned problem.

3.1 Research Scope for the Solution Architecture

Today there are many static code analysis tools which mainly do the same thing, analyzing the code base and identifying the mistakes done by the programmer and identifying the ways the code can be improved. Figure 3-1 shows how a static code analysis tool works in high level.

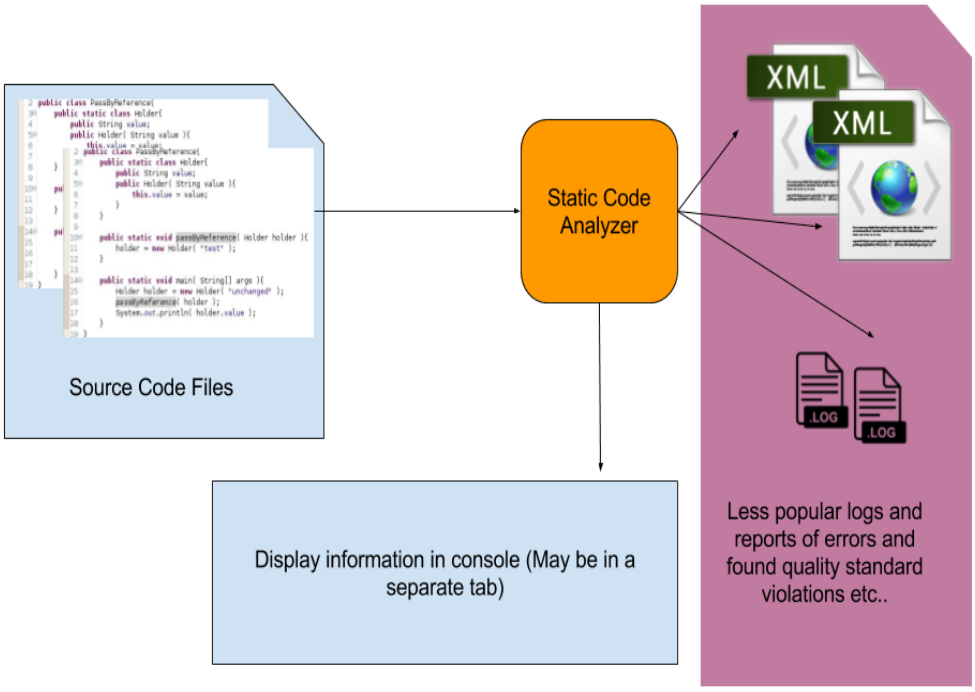

Figure 3-1: Static Code Analysis Process

As described in the image the static code analysis tool will generate log files with the captured data. It may be configured to create those logs in a persistent way if developer wants to retain those data, but most of the time those data will be replaced by the next batch of results generated by the tool. The behavior of the tools and the generated log files was analyzed in order to get an understanding about how to use those logs to capture user specific data.

As per a proof of concept the Eclipse IDE was selected as the preferred IDE and the FindBug code analyzer was selected as the preferred static code analyzing tool. This research used those to develop a prototype tool and do a proof of concept of the identified methodologies.

The research scope was defined to research develop a methodology

3.2 Solution Architecture

There are few major components. The log collection should be done by a dedicated module for each static code analysis tool. This is because the collection and presentation of each tool is different from each other. One tool might use XML as the output and one might use JSON as the result format. As per the POC the FindBug tool generate an XML results file. So there is a parser for the FindBug results which responsible for extracting the data which can be used to derive information in those files. As per the figure 3-2 Log Collector and Data Extractor will be created per each static code analysis tool in use.

The collected data was stored in a flat file DB (can be change this to a persistence DB like MySQL) for further analysis by the module call Log Data Analyzer. Which is responsible to analyze the data gathered by the log collectors and derive useful information from them.

Each static code analysis tool detect problems related to few areas,

- Basic language features
- Practices that violate recommended coding best practices
- Error prone coding styles
- Inefficient coding styles
- Apparent coding mistakes
- Many related to DB, I/O and etc

In the analyzer module the generated data was used to get the information regarding the developers understanding about the basic language features, areas like exception handling, database management, input/output handling and coding best practices, performance related best practices, security related best practices and many more.
By analyzing those data many insights about the developer can be obtained.

- What are the weak areas of the developer that needs to be improved
- What are the areas the developer is familiar with or mostly in touch with

- What is the trend of the mistakes done by the developer
    - Is he progressing through or not?
- The level of the developer in the specific areas

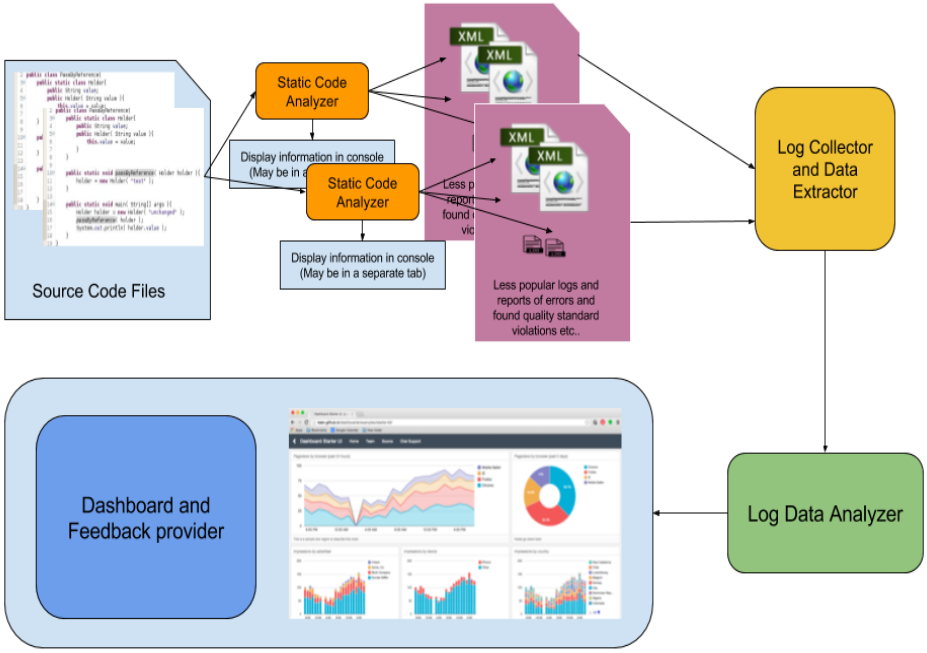

Figure 3-2 Proposed System Architecture

3.3 Evaluation Methodology

The information generated can be used to enhance the developer itself by provide real time feedback to the developer. The user can view the history data like what are the most common mistakes that was done by him and what are the areas that he needs to improve. The tool itself can be configured to suggest the developer about the possible remedies like resources to follow or people to contact.

For the management level this tool generated information can be a huge asset. That information can be used to,

- Identify the developers who need help in certain areas
    - So that the organization can conduct relevant workshops in order to mitigate those issues
- Identify the developers who perform up to expectation and use them to enhance the skill level of others
- Review the developer itself

The developed POC and the idea then evaluated both in theoretical and empirical forms in order to prove that this methodology suggested in the research can effectively used to collect the developer information related to coding best practices and patters without doing a separate manual analysis on developer work.

Chapter 4
SOLUTION ARCHITECTURE AND IMPLEMENTATION

4.1 Analyzing Different Formats of Machine Generated Data

Machine generated data largely used for data analysis and machine learning in different purposes. For example setups for monitoring oil and gas pipelines, natural disaster warning systems based on feeds from marine sensors, forecasting systems that take data from satellites and weather stations to help predict weather in small geographic areas, and a building energy management system that analyzes HVAC and elevator data to improve efficiency. We can see machine generated data which has different types, different formats and often create it on a defined time schedule or in response to a state change, action, transaction, or other event.

In this research the machine data generated by static code analysis are used as the data source for analyzing the coding patterns and timelines of introduction of mistakes to the code. Most of the static code analysis tools automatically generated the results of their evaluations as logs and save them in different formats. These files consist of important information about the bugs and project it run on. Though some static code analysis tools do not support for automatically generating these output files developers can configure them to generate these files in different formats. At least most of the provided options to save the results while doing the evaluation on a buggy code.

Sample log data file content generated by find bug static code analysis tool is shown in the following figures.

```
<BugCollection version="3.0.1-dev-20150306-5afe4d1" sequence="2" timestamp="1497541067962" analysisTimestamp="1497541059230" release="">
-----
-----
<BugInstance type="BC_IMPOSSIBLE_CAST" priority="1" rank="9" abbrev="BC" category="CORRECTNESS" first="2">
<Class classname="tefinBugst.Main">
<SourceLineclassname="tefinBugst.Main" sourcefile="Main.java" sourcepath="tefinBugst/Main.java"/>
</Class>
<Method classname="tefinBugst.Main" name="main" signature="([Ljava/lang/String;)V" isStatic="true">
<SourceLineclassname="tefinBugst.Main" start="7" end="11" startBytecode="0" endBytecode="15" sourcefile="Main.java" sourcepath="tefinBugst/Main.java"/>
</Method>
<Type descriptor="Ljava/lang/Double;" role="TYPE_FOUND">
<SourceLineclassname="java.lang.Double"/>
</Type>
<Type descriptor="Ljava/lang/Long;" role="TYPE_EXPECTED">
<SourceLineclassname="java.lang.Long"/>
</Type>
<LocalVariable name="doubleValue" register="1" pc="5" role="LOCAL_VARIABLE_VALUE_OF"/>
<SourceLineclassname="tefinBugst.Main" start="8" end="8" startBytecode="6" endBytecode="6" sourcefile="Main.java" sourcepath="tefinBugst/Main.java"/>
</BugInstance>
<FindBugsProfile>
-----
------
</BugCollection>
```

4.2 Reading XML Files

XML is the widely used file format for static code analysis tools. A separate parsing mechanism to extract the data from them was created after analyzing few existing mechanisms. There are many different libraries and methods to read xml files in java.

4.2.1 JAXB for XML Parsing

Jaxb is a very popular method frequently used for reading xml as it provides is the easiest way to convert XML to java object and java object to XML. It provides simple API for the developers to read and write java objects to and from the xml files. Jaxb stands for the java architecture for xml binding. Here we don't need to aware of xml parsing techniques. Handling of marshalling (Converting a java object to XML) and unmarshalling (Converting a XML to java object) processes in Jacob is done according to following figures.

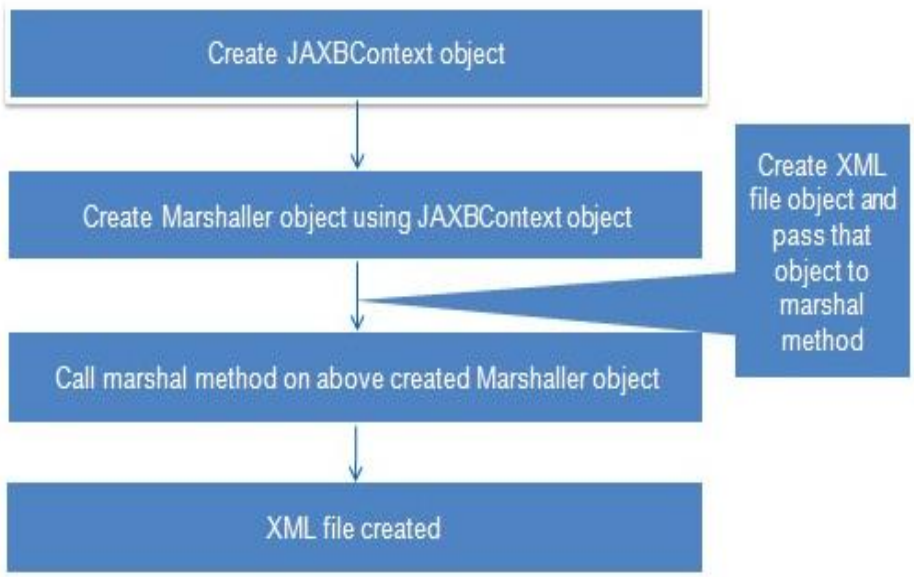

Figure 4-1: Marshalling in JAXB

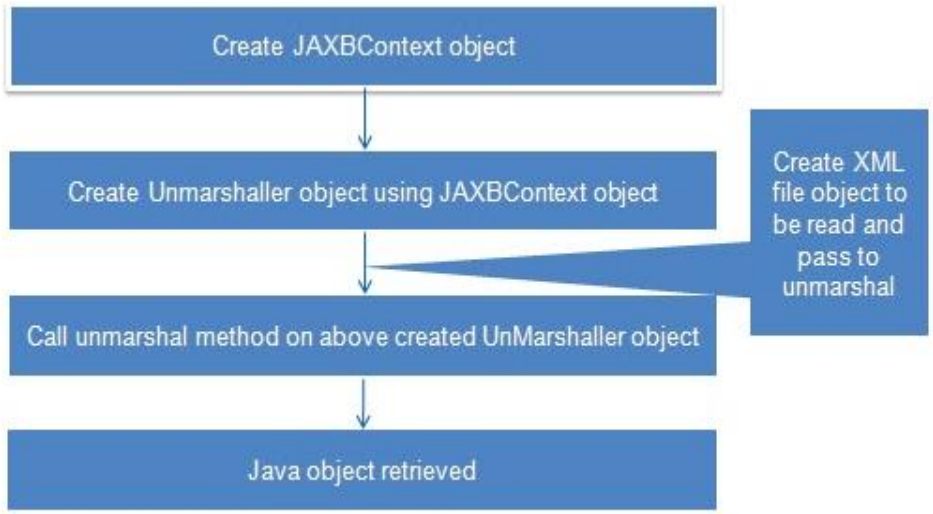

Figure 4-2: Unmarshalling in JAXB

To present the XML document to the program in a Java format the scheme for the xml document should be bind for the xml document into a set of Java classes that represents the schema. The annotation shown below can be used for to bind the used java classes in to xml.

- @XmlRootElement:This annotation defines root element of XML file.
- @XmlType(propOrder = {'list of attributes in order'}):This is used to define order of elements in XML file.This is optional.
- @XmlElement:This is used to define element in XML file.It sets name of entity.
- @XmlElementWrapper(name = 'name to be given to that wrapper'):It generates a wrapper element around XML representation.

Identified advantages off using JAXB for the log data parser as follows,

- Simple in use than SAX parser
- No need to aware of XML parsing techniques
- No need to access XML in tree structure always
- Can marshal XML file to other data targets such as input-streams, URL,DOM node
- Can unmarshal XML file from other data targets

Identified disadvantages off using JAXB for the log data parser as follows,

- As it is high layer API it has less control on parsing than SAX or DOM
- Slower than SAX as has some overhead tasks

4.2.2 Java Document Object Model for XML Parsing

Document Object Model defines an interface that enables programs to access and update the style, structure, and contents of XML documents. This parser parses and entire xml document and load it into memory. When you parse you get back a tree structure that contains all of the elements of the file for easy traversal and manipulation. Dom provides different functions to examine the contents and structure of the xml file. Dom is used when we need to know more details about the structure of the xml file, need to move parts of the file around or need to use the information in the file more than once.

Dom defines several java interfaces. Some of the basic interfaces are as follows,

- Node - The base datatype of the DOM
- Element - The vast majority of the objects deal with are Elements
- Attr - Represents an attribute of an element
- Text - -The actual content of an Element or Attr
- Document - Represents the entire XML document. A Document object is often referred to as a DOM tree

Steps have to be used when parsing a document using DOM Parser is as follows,

- Import XML-related packages
- Create a DocumentBuilder
- Create a Document from a file or stream
- Extract the root element
- Examine attributes
- Examine sub-elements

As identified there are several advantages to use the DOM,

- One of its design goals is that Java code written for one DOM-compliant parser should run on any other DOM-compliant parser without changes
- Can make changes directly to the tree in memory

4.2.3 Reading JSON Files

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. Some of the static code analysis tools use Json as the log file format. SO it is better to analyze about Json parser for future reference.

JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array

- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence

These data structures are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures. So there are numerous libraries and methodologies to manipulate JSON data.

The following are a few advantages of JSON over XML,

- JSON is very clear and easy to understand
- JSON is lighter than XML and very lightweight to transfer in the HTTP protocol

4.3 Solution Architecture

A proof of concept application architecture was designed in order to provide verification on the suggested methodology.

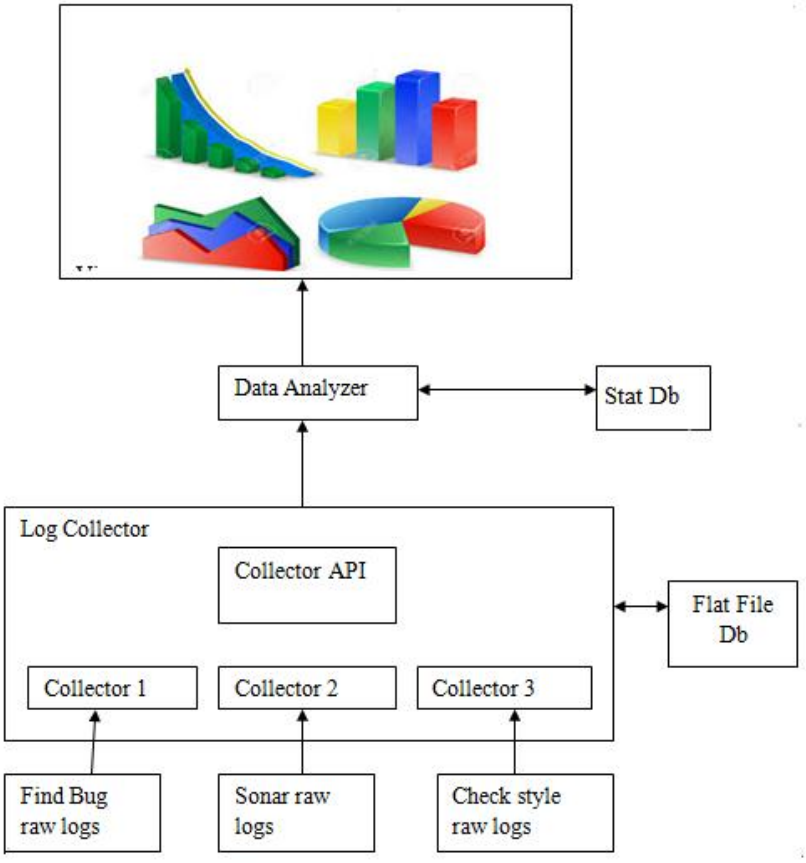

Figure 4-3 : Solution Architecture

4.3.1 Log Collector

The main challenge in using the machine data of static analysis tool is different static analysis tools use different type of structures, different type of formats and save the result in to a different type of files. So these results should be extracted in different ways using methods which are specific for the different type of tools.

The Log collectors are implemented to fulfill that purpose. A log collector is the data collector which collects data from the log (machine generated data) file of a static analysis tool which uses methods that are specific for the tools to extract data. Different types of sources have different type of collectors. Please refer the APPENDIX B for a sample log data parser for FindBugs log files.

According to system architecture separate log collectors used for different static code analysis tools. As described in the above section the collector use particular methods to read log files according to the extension type of file. For example sometimes though they have same type of file type we have to read the data in different ways as their structures and formats are not the same. For example FindBug collector use a xml reading method to extract the data from the log file as FindBug saved the results in a xml file. But among the xml methods it uses the second method (DOM) as it need more control on parsing as the structure of the bug instance can be different according to the bug type.

So the log collector read the data from the log files extract the important information about the bugs and analysis and then save them in the flat file db in a defined manner.

4.3.2 Persistence DB

For the POC purpose this is simply a flat file system (or further improved as a relational data base). Since each data source may have their log files in different formats we need to make sure they'll end up in a common format before pass them to processing and analyzing.

So the tool has a log collector which comprises of dedicated collectors for each data source and stores the details in a common tool friendly format.

4.3.3 Collector API

Collector API provides the functionalities that needed to access collected data in the flat file. So upper layers no need to aware of flat file format or data structuring format. This API hides the complexity of the collectors and the flat file from the upper layers.

Some functions provided by collector API is shown below.

- getCriticalBugList()
- getAlltheBugs(from,to)

4.3.4 Data Analyzer

Data analyzer is the component which analyzes the data collected by the different sources. Data analyzer directly gets data from the persistent database where the data are stored in a defined structure. Here analyzer analyzes the bug information based on multiple views.

One way of categorizing is to analyze the data according to the Severity of the bug. The bugs can have different severity levels. Some Bugs should have more concerns while some bugs can be neglect. So we consider 3 severity levels.

1. Critical
2. Medium
3. Low

The data analyzer analyzed bug information according to the severity and it saved the analyzed data in the stat Database. Data analyzing happens on the bug category also. The bugs found in a source code can be basically categorized as follows.

Table 4-1: Bug Categories

Category	**Description**
Bad Practice	Practices that violate recommended coding practices.
Dodgy	Code that is confusing, anomalous, and error-prone
Performance	Inefficient memory usage/buffer allocation, usage of non-static classes.
Internationalization	Use of non-localized methods
Malicious code vulnerability	Variables or fields exposed to classes that should not be using them.
Bogus random noise	Bug data mining related. Not useful in bug-finding.
Correctness	Apparent coding mistakes.
Multithreaded correctness	Thread synchronization issues.
Security	Similar to malicious code vulnerability.

There can be different type of bugs in different source codes and from different static analysis tools. So data analysis has to be done according the types of Bugs. For example some of the bug types of the FindBug tool are shown below.

- BC: Equals method should not assume anything about the type of its argument
- BIT: Check for sign of bitwise operation
- CN: Class implements Cloneable but does not define or use clone method
- CN: clone method does not call super.clone()
- CN: Class defines clone() but doesn't implement Cloneable
- CNT: Rough value of known constant found
- Co: Abstract class defines covariant compareTo() method
- Co: compareTo()/compare() incorrectly handles float or double value
- Co: compareTo()/compare() returns Integer.MIN_VALUE
- Co: Covariant compareTo() method defined
- DE: Method might drop exception

Data analyzer analyses data according to the above views calculates and get their statistic and save them in the stat DB.

4.3.5 Statistics Database

The statistic data about the bug information generated by the data analyses is stored in the stat Db. This data base is used to view the statistics of the data and can be used for further analysis purposes.

4.4 View Layer

This is responsible for displaying the actual statistics of the developer. There are several predefined statistics calculated over the time and analyzed accordingly. The progress made by the developer is also describes using graphical representation. User can track his or her progress using these statistics in real time.

Following images shows two views of the view layer after plug-in to the eclipse.

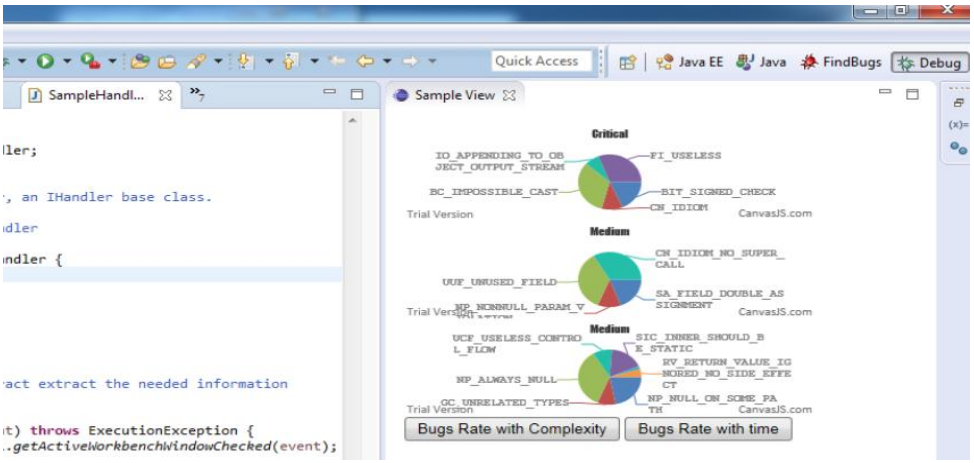

Figure 4-4 : Plug-in View1

Figure 4-5 : Plug-in View2

4.5 Using Collected Statistics To Preserve Coding Best Practices

Once the statistics collected for a significant period the coding patterns and the timelines of the violations of coding best practices can be identified. Each developer statistics can be separately analyzed.

Using these results the organization can identify the skill level of the developer and the developer coding patterns towards bad coding practices. With that data organizations can take actions. So with this approach the organization does not need to conduct separate interviews or manually review the work done by the developer in order to collect information.

Chapter 5
EVALUATION

This research focuses on how to harvest the valuable information from the less interested machine data such as simple static code analysis logs. Implementing a direct comparison based approach to test the results or the impact of this idea is somewhat difficult because of the nature of it. Measuring the correctness of the derived information can be a potential methodology to evaluate this idea [30].

So we can divide the evaluation methodology into few sections.

1) Collect machine data from developers
   a) Collect machine data for two months from 6 developers
2) Identify potential types of information to be generated
3) Manually evaluate the collected machine data and derive the information required
   a) By analyzing the log files provided by the participants we can check what kind of bug patterns that they produced over the time. Using that data we can get an idea of the participants' skill level on those selected categories. For an example if the participants' log data show many entries regarding exception handling, then we can assume that the participant might have a difficulty in writing a safe code using exception handling. Another example may be that if a participants data show many entries regarding basic Java coding standard then we can assume that participant might have difficulties in the knowledge of Java coding standards
4) Conduct one on one interviews with the 6 developers to derive the information required
   a) By conducting interviews regarding the Java knowledge and other related areas we can find out the strength of the participant of those above identified areas
5) Compare the derived information sets
   a) We can compare the information we found by analyzing the log files against the results we gathered from the interviews

5.1 Collect Machine Data

As per this research we are considering FindBug as the tool for the static code analysis. FindBug creates a log file (Most common type is a .XML) including the results of the static code analysis. This log file can be created manually or it can be set to create automatically every time the user runs the code analysis. This collection should be done by the user and there are no timing rules or constraints. Data then can be feed into the DB for further analysis. We selected six junior Java developers to this experiment and they were asked to collect the FindBug generated log data at least few times a week.

5.2 Identify Potential Types of Information to Be Generated

As identified FindBug generates data in few segments which can be directly considered as good candidates for the potential types.

Table 5-1 : Selected categories for the evaluation

Category	Description
Bad Practice	This section catches the practices that violate the recommended coding best practices. Simple example is the use of "==" instead of the .equals() method when comparing the String Objects of when comparing the Objects regarding their contents.
Correctness	Apparent coding mistakes
Malicious code vulnerability	Variables or fields exposed to classes that should not be using them. Returning a reference to mutable object may expose internal representation is an example.
Multithreaded correctness	This related to the thread synchronization issues.
Performance	Inefficient memory usage/buffer allocation, usage of non-static classes. Creating a new String(String) constructor.
Security	Similar to malicious code vulnerability, but with a focus on to the security aspect.
Dodgy code	Code that is confusing, anomalous, and error-prone. For example null-dereference, and catch-all exceptions

By analyzing the FindBug bug descriptions we can divide them to few other categories related to the language features.

- Basic language features related best practices
- Exception handling related best practices
- Database management related best practices
- Security related best practices
- Input/output handling related best practices
- Java Collections framework related best practices

5.3 Evaluate Collected Machine Data and Derive the Information Required

In this phase we tried to detect the patterns of the user by analyzing the log files they have generated during the development.

Below is an example log file which contains few bug instances related to basic Java coding standards and java language. As it mentioned it has few entries regarding method naming conventions, wrong comparison styles, redundant code logics, few errors regarding exception handling and string formats. So as a whole by looking at this we can assume the owner of this log might have some difficulties writing a Java code adhering to the standard guidelines. Might have few problems with exception handling.

```
<BugInstance type="NM_METHOD_NAMING_CONVENTION" priority="2" rank="16" abbrev="Nm" category="BAD_PRACTICE" first="1">
<Class classname="AmmunitionTemplate">
<SourceLineclassname="AmmunitionTemplate" sourcefile="AmmunitionTemplate.java" sourcepath="AmmunitionTemplate.java"/>
</Class>
<Method classname="AmmunitionTemplate" name="CreateAmmunitionTemplate" signature="()V" isStatic="false">
<SourceLineclassname="AmmunitionTemplate" start="11" end="17" startBytecode="0" endBytecode="182" sourcefile="AmmunitionTemplate.java" sourcepath="AmmunitionTemplate.java"/>
</Method>
</BugInstance>
<BugInstance type="NP_ALWAYS_NULL" priority="1" rank="5" abbrev="NP" category="CORRECTNESS" first="1">
<Class classname="AmmunitionTemplate">
<SourceLineclassname="AmmunitionTemplate" sourcefile="AmmunitionTemplate.java" sourcepath="AmmunitionTemplate.java"/>
</Class>
<BugInstance type="ES_COMPARING_STRINGS_WITH_EQ" priority="1" rank="9" abbrev="ES" category="BAD_PRACTICE" first="1">
<Class classname="HumanErrorArea">
<SourceLineclassname="HumanErrorArea" sourcefile="HumanErrorArea.java" sourcepath="HumanErrorArea.java"/>
</Class>
<Method classname="HumanErrorArea" name="getGridValuesinString" signature="()Ljava/util/ArrayList;" isStatic="false">
<SourceLineclassname="HumanErrorArea" start="19" end="24" startBytecode="0" endBytecode="31" sourcefile="HumanErrorArea.java" sourcepath="HumanErrorArea.java"/>
</Method>
<BugInstance type="RCN_REDUNDANT_NULLCHECK_OF_NULL_VALUE" priority="2" rank="18" abbrev="RCN" category="STYLE" first="1">
<Class classname="Polygon2D">
<SourceLineclassname="Polygon2D" sourcefile="Polygon2D.java" sourcepath="Polygon2D.java"/>
</BugInstance>
<BugInstance type="NM_METHOD_NAMING_CONVENTION" priority="2" rank="16" abbrev="Nm" category="BAD_PRACTICE" first="1">
<Class classname="TargetPlacementVisualization">
</BugInstance>
<BugInstance type="RV_RETURN_VALUE_IGNORED" priority="1" rank="3" abbrev="RV" category="CORRECTNESS" first="1">
<Class classname="TargetPlacementVisualization">
<SourceLineclassname="TargetPlacementVisualization" sourcefile="TargetPlacementVisualization.java" sourcepath="TargetPlacementVisualization.java"/>
</Class>
</BugInstance>
<BugInstance type="VA_FORMAT_STRING_ILLEGAL" priority="1" rank="9" abbrev="FS" category="CORRECTNESS" first="1">
<Class classname="TargetPlacementVisualization">
```

This log file has some entries regarding unconfirmed casts, stream clean up issues and some database related issues such as non constant strings passed to execute batch Methods on an SQL statement (which can be lead to SQL injection vulnerability)

```
<BugInstance type="BC_UNCONFIRMED_CAST" priority="2" rank="17" abbrev="BC" category="STYLE" first="1">
<Class classname="DBHandler">
<SourceLineclassname="DBHandler" sourcefile="DBHandler.java" sourcepath="DBHandler.java"/>
</Class>
</BugInstance>
<BugInstance type="OBL_UNSATISFIED_OBLIGATION" priority="2" rank="20" abbrev="OBL" category="EXPERIMENTAL" first="1">
<Int value="1" role="INT_OBLIGATIONS_REMAINING"/>
<SourceLineclassname="DBHandler" start="18" end="18" startBytecode="29" endBytecode="29" sourcefile="DBHandler.java" sourcepath="DBHandler.java" role="SOURCE_LINE_OBLIGATION_CREATED"/>
<String value="{Statement x 1}" role="STRING_REMAINING_OBLIGATIONS"/>
</BugInstance>
<BugInstance type="SQL_NONCONSTANT_STRING_PASSED_TO_EXECUTE" priority="1" rank="10" abbrev="SQL" category="SECURITY" first="1">
<Class classname="DBHandler">
<SourceLineclassname="DBHandler" sourcefile="DBHandler.java" sourcepath="DBHandler.java"/>
</Class>
```

This log file has some entries related to few dodgy codes and some apparent mistakes including erroneous switch fall through situations and impossible downcast codes. Have some mistakes in exception handling.

```
<BugInstance type="BC_IMPOSSIBLE_DOWNCAST_OF_TOARRAY" priority="1" rank="5" abbrev="BC" category="CORRECTNESS" first="1">
<Class classname="Class">
<SourceLineclassname="Class" sourcefile="Class.java" sourcepath="Class.java"/>
</Class>
</BugInstance>
<BugInstance type="SF_DEAD_STORE_DUE_TO_SWITCH_FALLTHROUGH_TO_THROW" priority="1" rank="1" abbrev="SF" category="CORRECTNESS" first="1">
<Class classname="Class">
<SourceLineclassname="Class" sourcefile="Class.java" sourcepath="Class.java"/>
</Class>
</BugInstance>
<BugInstance type="SF_SWITCH_FALLTHROUGH" priority="2" rank="17" abbrev="SF" category="STYLE" first="1">
<Class classname="Room">
<SourceLineclassname="Room" sourcefile="Room.java" sourcepath="Room.java"/>
</Class>
</BugInstance>
<BugInstance type="SF_SWITCH_NO_DEFAULT" priority="2" rank="19" abbrev="SF" category="STYLE" first="1">
<Class classname="Room">
<SourceLineclassname="Room" sourcefile="Room.java" sourcepath="Room.java"/>
</Class>
</BugInstance>
<BugInstance type="VA_FORMAT_STRING_ILLEGAL" priority="1" rank="9" abbrev="FS" category="CORRECTNESS" first="1">
<Class classname="Teacher">
```

For the information about data collected for two months using log files please refer to APPENDIX A. The results summarized as in the Table 5-2 and Figure 5-1 .

Table 5-2: Overall data collected for 8 weeks

Overall	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	20	21	15	14	10	9
Correctness	17	13	14	13	8	8
Malicious code vulnerability	6	4	1	2	2	4
Multithreaded correctness	8	5	3	1	3	2
Performance	16	17	10	10	23	8
Security	7	4	4	3	3	15
Dodgy code	16	16	12	21	12	10

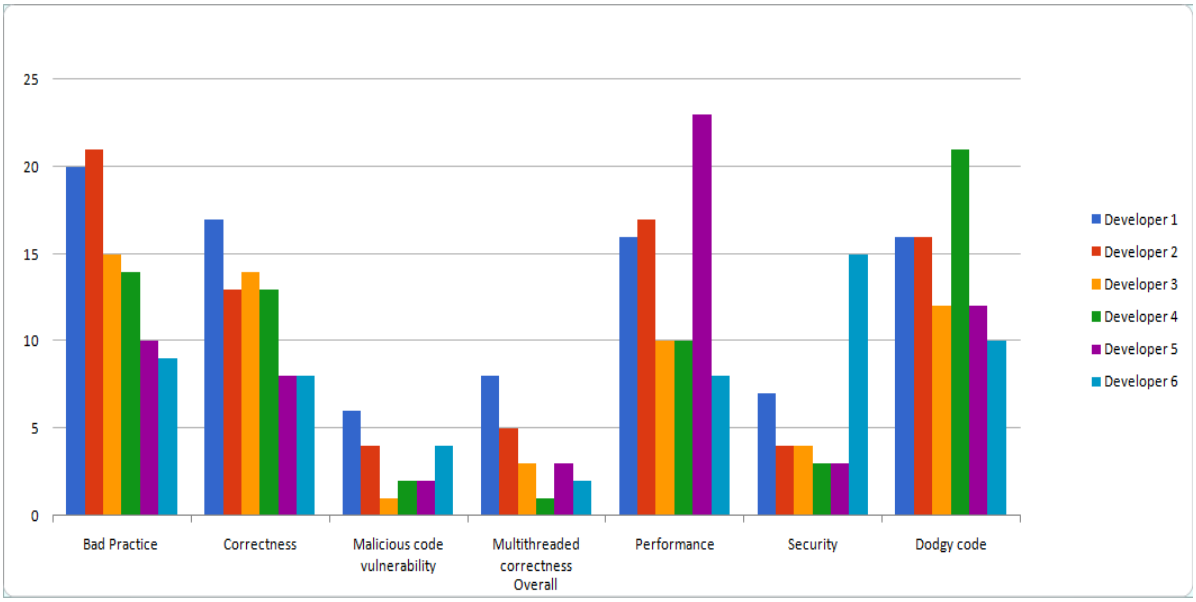

Figure 5-1: Comparison chart of bug instances

5.4 Conduct One on One Interviews

In this evaluation we are focused on detecting technical difficulties faced by the developers. Since we are focusing on FindBug tool, Six Java developers are selected to this experiment. These interviews are conducted without a prior notice so that the interviewee does not get a chance to get ready for it.

We will be asking questions towards the categories identified in the Table 5-1 in order to get an idea about how that individual possess the required knowledge in that category. Then we can roughly give a performance value for each user for each category.

The selected six junior Java developers participated in this experiment.

Sample breakdown of the interviews

- Introduction and the discussion on the actual research we are doing
- They were asked to describe the level of Java knowledge they possess
- They were asked questions from basic Java knowledge
- They were asked questions from intermediate Java knowledge including the areas that users mostly tend to do mistakes when writing Java codes. (FindBug bug descriptions were used in order to generate the questions)
    - They were asked to provide a code snippet for a scenario which FindBug might listen in actual scenarios
- They were asked questions from Java threads knowledge
- They were asked questions from Java coding best practices knowledge
- They were asked questions from basic Java security knowledge
- They were asked questions from Java I/O knowledge
- They were asked questions from basic Java application performance
- They were asked questions from basic database management in Java

Then some insight of the participants was gathered. Data collected by conducting interviews are as follows.

Table 5-3: Developer rankings in various categories

Interview	Dev1	Dev 2	Dev 3	Dev4	Dev 5	Dev6
Basic Java knowledge	2	2	3	3	3	3
Java coding standards	2	2	2	2	3	3
Java Coding best practices	1	1	2	1	3	3
Java I/O best practices	1	1	1	2	3	2

Performance related best practices	2	1	2	1	2	2
Security related best practices	1	1	1	2	2	1
Database management	3	2	3	3	3	1
Exception Handling	2	2	3	3	2	3

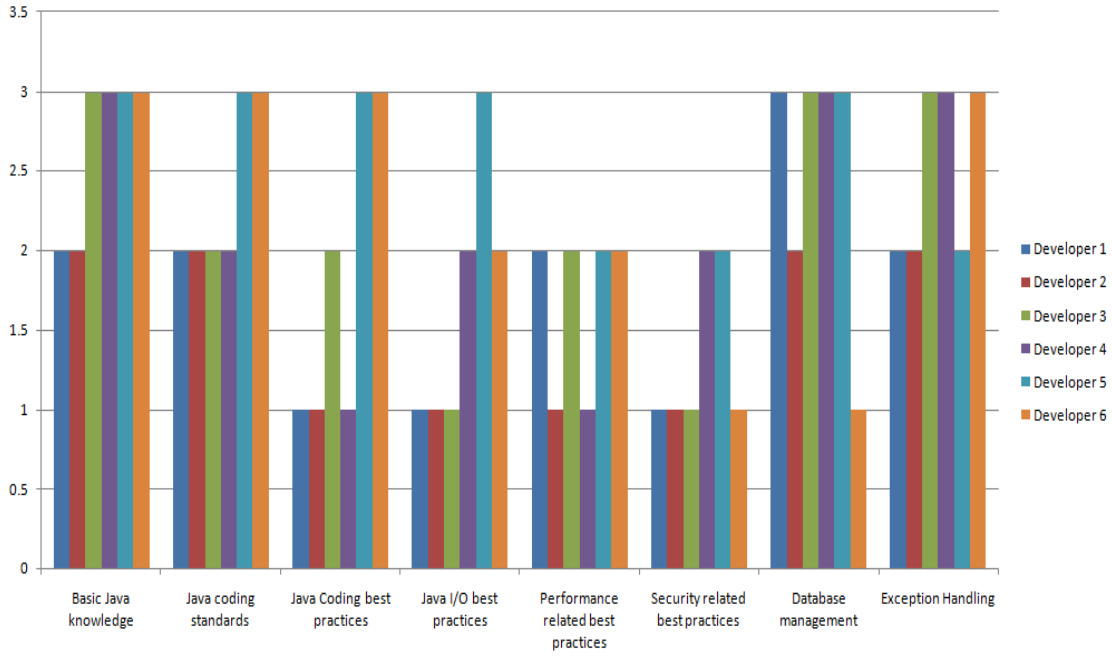

Figure 5-2 : Comparison of java language related best practices knowledge

5.5 Comparison of the Derived Datasets

The idea here is to check how accurate the derived results from the known data sets.

Below table represents the distribution of the bug instances in different java language feature sections. As it shows clearly most of them are in the basic language feature related section.

Table 5-4: Number of bug instances in different java language feature sections

Overall	Dev 1	Dev 2	Dev3	Dev 4	Dev5	Dev6
Basic language features	70	65	43	52	49	43
Exception handling	5	2	3	2	4	3
Database management	2	2	1	1	1	4
Security	3	2	4	3	1	1
Input/output handling	5	6	4	3	3	3
Java Collections framework	5	3	4	3	3	2

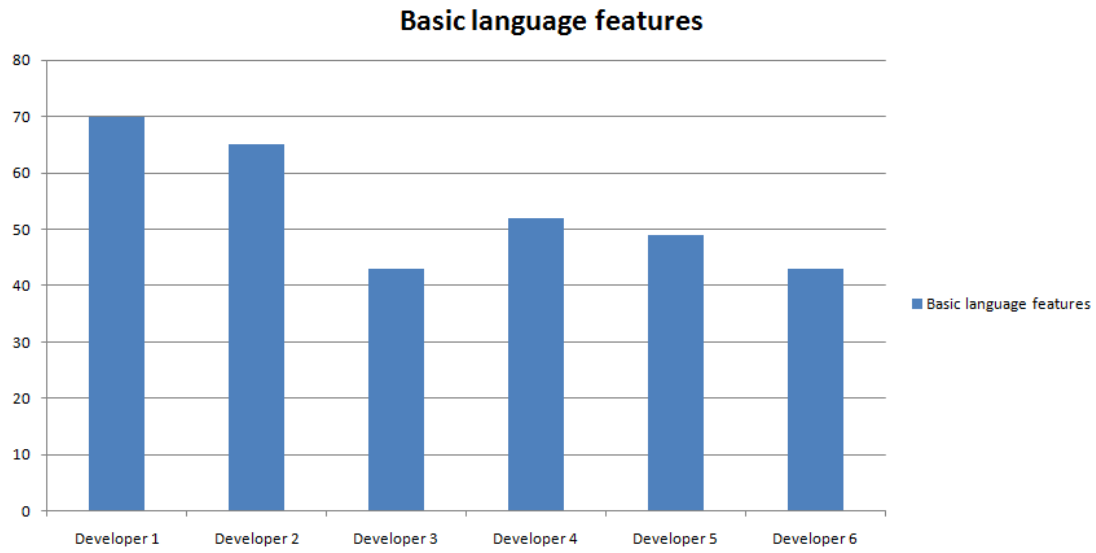

Figure 5-3 : Comparison of bug instances related to basic language features

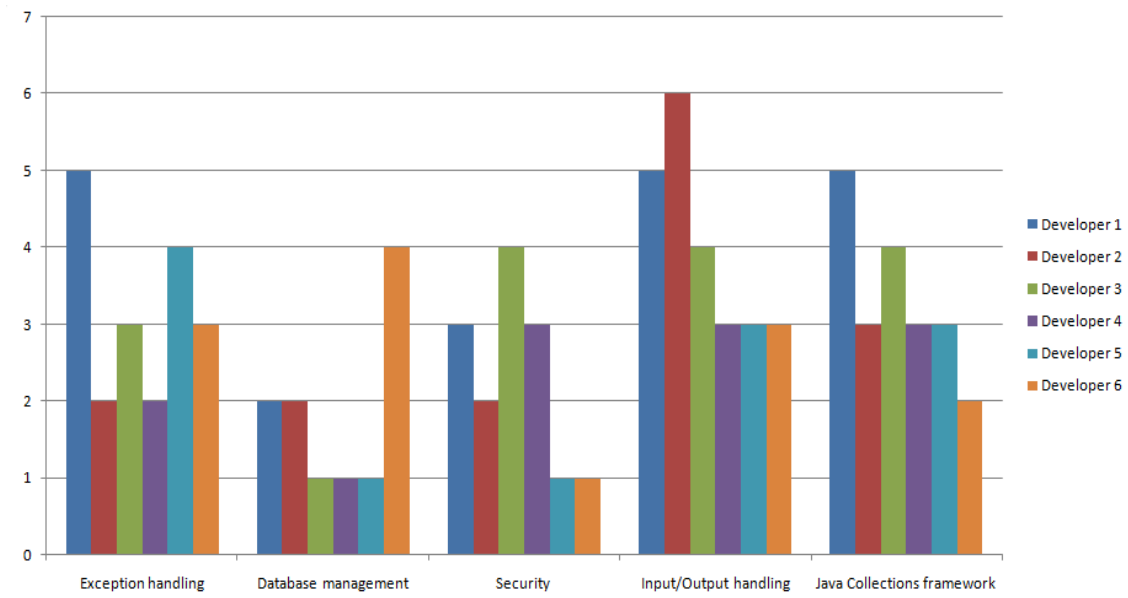

Figure 5-4 : Comparison of the bug instances related to other java related feature sections

Overview of the results gathered by the interviews and log data analysis are as below.

Dev 1	
Log data	Interview
• Most of the bug instances are due to violations of recommended coding best practices for Java • Has significant bug instances for correctness related and performance related best practices • Tend to write dodgy code since has significant bug instances from them • Among all the participants this one as the most number of instances from basic language features. So from above we can suggest that he needs improvement on basic language features, java coding best practices, performance and meaningful and unambiguous code writing techniques.	• Good understanding on database management • Average understanding on basic Java • Average understanding on Performance related best practices • Average understanding on Java coding standards • Average understanding on exception handling in Java • Needs improvement on Java coding best practices • Needs improvements on security related best practices • Needs improvements on Java I/O best practices

<table>
<tr><td colspan="2">Dev 2</td></tr>
<tr><td>Log data</td><td>Interview</td></tr>
<tr><td>Most of the bug instances are due to violations of recommended coding best practices for Java
Has significant bug instances for correctness related and performance related best practices
Tend to write dodgy code since has significant bug instances from them
Has significant entries from I/O best practice violations
So from above we can suggest that he needs improvement on java I/O, java coding best practices, performance and meaningful and unambiguous code writing techniques.</td><td>Average understanding on basic Java
Average understanding on Java coding standards
Average understanding on Database management
Average understanding on exception handling in Java
Needs improvements on Java I/O best practices
Needs improvement on Java coding best practices
Needs improvement on Performance related best practices
Needs improvements on security related best practices</td></tr>
<tr><td colspan="2">Dev 3</td></tr>
<tr><td>Log data</td><td>Interview</td></tr>
<tr><td>Most of the bug instances are due to violations of recommended coding best practices for Java
Has significant bug instances for correctness related and performance related best practices
Few instances regarding dodgy code segments
Has average entries in all categories
So from above we can suggest he needs improvement on Java coding best practices and meaningful and unambiguous code writing techniques</td><td>Good understanding On Basic Java knowledge
Average understanding On Java coding standards
Average understanding On Java Coding best practices
Needs improvements On Java I/O best practices
Average understanding On Performance related best practices
Needs improvements On Security related best practices
Good understanding On Database management
Good understanding On Exception Handling</td></tr>
<tr><td colspan="2">Dev 4</td></tr>
</table>

<table>
<tr><td>Log data</td><td>Interview</td></tr>
<tr><td>Most of the bug instances are due to the writing of dodgy code segments
Has significant number of entries in bad coding practices and correctness related issues.
From above we can suggest he need improvements on meaningful and unambiguous code writing techniques as well as basic java coding best practices.</td><td>Good understanding On Basic Java knowledge
Average understanding On Java coding standards
Needs improvements On Java Coding best practices
Average understanding On Java I/O best practices
Needs improvements On Performance related best practices
Average understanding On Security related best practices
Good understanding On Database management
Good understanding On Exception Handling</td></tr>
<tr><td colspan="2">Dev 5</td></tr>
<tr><td>Log data</td><td>Interview</td></tr>
<tr><td>Most of the bug instances are from performance related best practices violations
Tend to write dodgy code
Relative to others has low number of java coding standards violations
Significant entries in exception handling section
From this we can suggest that he needs improvement in performance related best practices and meaningful and unambiguous code writing techniques. Some brush-up in exception handling best practices.</td><td>Good understanding On Basic Java knowledge
Good understanding On Java coding standards
Good understanding On Java Coding best practices
Good understanding On Java I/O best practices
Average understanding On Performance related best practices
Average understanding On Security related best practices
Good understanding On Database management
Average understanding On Exception Handling</td></tr>
<tr><td colspan="2">Dev 6</td></tr>
<tr><td>Log data</td><td>Interview</td></tr>
</table>

• Most of the bug instances are from security related best practices violations • Tend to write dodgy code • Relative to others has low number of java coding standards violations • Has significant number of entries in database related and exception handling related issues From above we can suggest that he needs improvement in security related best practices and meaningful and unambiguous code writing techniques. Needs to improve database and exception handling best practices.	• Good understanding On Basic Java knowledge • Good understanding On Java coding standards • Good understanding On Java Coding best practices • Average understanding On Java I/O best practices • Average understanding On Performance related best practices • Needs improvements On Security related best practices • Needs improvements On Database management • Good understanding On Exception Handling

The table 5-5 summarizes the above comparisons numerically. The categories were decided by a subject matter expert (The bugs can be categorized as required for the monitoring party) and the FindBugs bug instances were categorized to those selected categories by a manual process. We have introduced a rating schema for that as well.

- 1 - Needs improvements
- 2 - Low understanding
- 3 - Average understanding
- 4 - Good understanding
- 5 - Excellent understanding

The final match percentages were calculated by the below formula,

$$\left(\frac{\sum_{i=1}^{n}(100 - |InterviewRating - SystemRating| * 25)}{n}\right)\%$$

Here the 'InterviewRating' is the rating given with respective to the interviews and code reviews. The 'SystemRating' is the rating given from the implemented system by analyzing logs. The value 'n' is the number of categories.

Table 5-5 : Rated comparison of interview results and log analysis results

	Developer 1		Developer 2		Developer 3		Developer 4		Developer 5		Developer 6	
Interview	Interview	Log analysis	Interview	Log analysis	Interview	Log analysis	Interview	Log analysis	Interview	Log analysis	Interview	Log analysis
Basic Java knowledge	2	2	2	2	3	2	3	2	3	3	3	2
Java coding standards	2	1	2	1	2	1	2	2	3	2	3	2
Java Coding best practices	1	1	1	1	2	1	1	1	3	2	3	1
Java I/O best practices	1	1	1	1	1	1	2	1	3	2	2	2
Performance related best practices	2	1	1	1	2	1	1	2	2	1	2	2
Security related best practices	1	1	1	2	1	1	2	2	2	2	1	1
Database management	3	2	2	2	3	3	3	2	3	2	1	1
Exception Handling	2	2	2	2	3	2	3	1	2	1	3	1
Match percentage (%)	90.625		93.75		84.375		81.25		81.25		81.25	

We can observe that the match percentage is above 80% for most of the cases. So we can assume that there is a significant correlation between the developer level of coding best practices and the generated log data by the developer.

Chapter 6
CONCLUSION

Improving, sharing and preserving coding best practices through programmer data analytics is a new area for software development. Since it's at its early phase contributing to that aspect from this research will help the future researchers to adopt this findings and develop a better solution for the upcoming problems. This research is a significant one in many areas including its proposed way to develop applications while improving the developer itself.

6.1 Research Contribution

This research focuses on finding a viable solution for a real problem in current software development methodologies. Improving the developer while developing the software will be a worth area for contributing since the skill level of the programmer is much more important in certain situations. This research contributes to the existing researches by considering an untouched area. While most of the other researchers consider changing the development methodologies and resource management enhancements this research will actually focus on something that was there for almost few decades. Static code analysis tools will leave log files with valuable information of the developer itself. Those log files were not considered as valuable and left for consider as garbage. With this research those data get a value to them self and play a major role in this research methodology.

So this research actually does two major contributions.

- Discovered an area of analysis using programmer related machine data
- Enhance the existing software development methodologies by proposing a way to share and preserve coding best practices which is important to the quality of the product

This research provides a good evaluation process to prove that the actual methodology works in most of the scenarios.

6.2 Future Work and Conclusion

This research contains an evaluation where it proves that the proposed methodology can actually work in real scenarios. So as seen from the results it can be seen in most of the cases the results obtained by the interviews are matched with the results obtained by the log analysis. Since these results matched to nearly 75% in most of the cases we can safely assume that the results obtained by the log analysis are reliable (With assumption on that the results obtained by the interviews are reliable, It was made sure of that by arranging spot interviews without prior inform to the participant so that we can measure participants' skill as is. Participant did not have additional preparation time for the interviews).

If these results obtained by the log analysis are accurate up to that much of an extend then it can be assumed that we have a mechanism to measure the skill level of a developer without interviewing them. Importantly we have a mechanism that can check and track the skill sets of a developer related to coding best practices.

Since we can actually get an understanding about the level of each section that a developer is performing (basic Java, Exception, Database, etc) we can select appropriate customized skill development mechanism for each developer. For an example from the evaluation, regarding the first and second developers we might need to arrange a knowledge sharing session in java coding best practices and meaningful and unambiguous code writing techniques. Regarding the fifth developer we might need him to brush up his exception handling skills.

One thing is we can't actually compare the developers with each other since the load and the nature of the workload that they are performing during the two months of time is different. The reason for the fifth developer to make few bug instances regarding exception handling is that he is working on some complex task than others, and the reason why the sixth developer made few bug instances in database side is that he is working on a database related project work.

Even though we can't compare the developers among each other we can have a solid understanding about how each developer is performing in his work. Since these data and information is also visible to the developer, he can also self-evaluate the areas that he needs improvement.

So far in this research study it considered Java programming language and the FindBug static code analysis tool. But this can be extended to other languages such as C#, C++, Python and etc. Static code analysis tools like Sonar, FxCop, cpplint and etc can also be considered.

By providing an evaluation of the methodology and open up a new area for sharing and preserving coding best practices through programmer data analytics this contribute to the field of study of software architecture. Finally as a research this can be considered as a successful one because this try to propose a way to enhance the existing software development methodologies in order to open a way for the developers to enhance their skills while participating in the development process.

REFERENCES

[1]D. Starr, "Defining Code Quality", *Blog.smartbear.com*, 2015. [Online]. Available: http://blog.smartbear.com/code-review/defining-code-quality/.

[2] "Code Quality - How It Affects Your Bottom Line", Castsoftware.com. [Online]. Available: http://www.castsoftware.com/glossary/code-quality. [Accessed: 04- Jan- 2017].

[3] D. Athanasiou and J.Visser, "Test Code Quality and Its Relation to Issue Handling Performance".

[4] R. Baggen, J. Correia, K. Schill and J.Visser, "Standardized code quality benchmarking for improving software maintainability"

[5] B. Luijten and J. Visser, "Faster defect resolution with highertechnical quality of software," in 4th International Workshop onSoftware Quality and Maintainability (SQM 2010), 2010.

[6] J. McGonigal, Reality is broken: Why games make us better and how they can change the world. Penguin, 2011.

[7] Philipp Lombriserand Roald van der Valk , "Improving the Quality of the Software Development Lifecycle with Gamification"

[8] G. Zichermann and C. Cunningham, Gamification by design: Implementinggame mechanics in web and mobile apps. " O'Reilly Media, Inc.," 2011.

[9] O. Pedreira, F. García, N. Brisaboa, and M. Piattini, "Gamification in software engineering – A systematic mapping," Information and Software Technology, vol. 57, pp. 157–168, Jan. 2015.

[10] J. Fernandes, D. Duarte, C. Ribeiro, C. Farinha, J. M. Pereira, and M. M. daSilva, "iThink: A Game-Based Approach Towards Improving Collaboration and Participation in Requirement Elicitation," *ProcediaComput. Sci.*, vol. 15,pp. 66–77, Jan. 2012.

[11] E. De Bono, *Six thinking hats*. Taylor & Francis, 1999

[12] N. Tillmann, J. Bishop, R. N. Horspool, D. Perelman, and T. Xie, "Code hunt: searching for secret code for fun," in *Proceedings of the 7th International Workshop on Search-Based Software Testing*, 2014, pp. 23–26.

[13] K. Januszewski, "Announcing Visual Studio Achievements Beta," 2012. [Online]. Available: http://channel9.msdn.com/Blogs/C9team/AnnouncingVisual-Studio-Achievements. [Accessed: 15-Sep-2014].

[14] Steve Bygren, Greg Carrier, Tom Maher, Patrick Maurer, David Smiley, Rick Spiewak, Christine Sweed, "Applying the fundamentals of quality to software acquisition", Systems Conference (SysCon) 2012 IEEE International, pp. 1-6, 2012.

[15] A. Aurum, H. Petersson, and C. Wohlin, “State-of-the-art: software inspections after 25 years,” SoftwareTesting, Verification and Reliability, vol. 12, no. 3, pp. 133–154, 2002.

[16]M. Harman, "Why Source Code Analysis and Manipulation Will Always Be Important".

[17] “Minimizing code defects to improve software quality and lower development costs” [online]. Available: ftp://ftp.software.ibm.com/software/rational/info/do-more/RAW14109USEN.pdf

[18] M. Gegick and L. Williams, “Towards the use of automated static analysis alerts for early identification of vulnerability- and attack-prone components,” in Proc. ICIMP, 2007, pp. 18–23.

[19] V. Barstad, V. Shulgin, M. Goodwin, and T. Gjøsæter, “Towards a Collaborative Code Review Plugin,” in Proceedings of the 2013 NIK conference, 2013, pp. 37–40.

[20] D. Hovemeyer and W. Pugh. Finding Bugs is Easy. In Onward!, OOPSLA’04, Vancouver, BC, October 2004.

[21] V. Barstad, M. Goodwin, and T. Gjøsæter, "Predicting Source Code Quality with Static Analysis and Machine Learning,".[Online]. Available: http://ojs.bibsys.no/index.php/NIK/article/download/26/22. Accessed: Aug. 15, 2016.

[22] N. Meghanathan, "Source Code Analysis to Remove Security Vulnerabilities in Java Socket Programs: A Case Study", International Journal of Network Security & Its Applications, vol. 5, no. 1, pp. 1-16, 2013.

[23] G. Murphy, M. Kersten, and L. Findlater, “How are Java software developersusing the Eclipse IDE?”Software, IEEE, vol. 23, pp. 76–83, 2006.

[24] W. Snipes, B. Robinson, and E. Murphy-Hill, "Code Hot Spot: A Tool for Extraction and Analysis of Code Change History".

[25] E. Murphy-Hill, B. Johnson, and Y. Song, "Why Don’t Software Developers Use Static Analysis Tools to Find Bugs?,". [Online]. Available: http://people.engr.ncsu.edu/ermurph3/papers/icse13b.pdf.

[26] J. Foster and M. Hicks, "Improving Software Quality with Static Analysis ∗".

[27] M. Gegick and L. Williams, “Towards the use of automated static analysis alerts for early identification of vulnerability- and attack-prone components,” in Proc. ICIMP, 2007, pp. 18–23.

[28] T. Menzies, J. Greenwald and A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictors", IEEE Transactions on Software Engineering, vol. 33, no. 1, pp. 2-13, 2007.

[29] "FindBugs Bug Descriptions", Findbugs.sourceforge.net, 2018. [Online]. Available: http://findbugs.sourceforge.net/bugDescriptions.html. [Accessed: 24- Aug- 2017].

[30] A guide to research evaluation ...". [Online]. Available: http://www.rand.org/pubs/monographs/MG1217.html. [Accessed: 28- Nov- 2017].

[31] R. Powell, "Evaluation Research: An Overview", Library Trends, vol. 55, no. 1, pp. 102-120, 2006.

[32] C. Catal, U. Sevim, and B. Diri, “Practical development of an Eclipse-based software fault prediction tool using Naive Bayes algorithm,” Expert Systems with Applications, 2011.

[33] "Findbugs - Static Code Analysis of Java", *Methodsandtools.com*, 2018. [Online]. Available: http://www.methodsandtools.com/tools/findbugs.php. [Accessed: 28- Nov- 2017].

[34] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. Hudepohl and M. Vouk, "On the value of static analysis for fault detection in software", *IEEE Transactions on Software Engineering*, vol. 32, no. 4, pp. 240-253, 2006.

[35] I. Stamelos, L. Angelis, A. Oikonomou and G. Bleris, "Code quality analysis in open source software development", *Information Systems Journal*, vol. 12, no. 1, pp. 43-60, 2002.

[36] P. Emanuelsson and U. Nilsson, "A Comparative Study of Industrial Static Analysis Tools", Electronic Notes in Theoretical Computer Science, vol. 217, pp. 5-21, 2008.

APPENDIX A

Data collected for two months using log files. The data are presented in weekly format.

Week 1	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	3	4	2	2	1	1
Correctness	3	2	1	2	0	0
Malicious code vulnerability	1	1	0	0	0	0
Multithreaded correctness	0	0	0	0	0	1
Performance	2	3	1	1	4	0
Security	1	1	1	0	0	3
Dodgy code	2	1	1	4	1	1

Week 2	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	2	2	1	1	1	1
Correctness	2	2	2	0	1	0
Malicious code vulnerability	1	1	1	0	0	1
Multithreaded correctness	2	1	1	0	0	0
Performance	3	1	1	2	3	1
Security	0	1	0	0	0	2
Dodgy code	1	1	1	3	2	1

Week 3	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	3	2	2	2	2	1
Correctness	3	2	0	1	0	2
Malicious code vulnerability	0	0	0	1	0	0
Multithreaded correctness	1	1	0	0	1	0
Performance	2	3	2	1	3	1
Security	1	0	1	0	0	1
Dodgy code	3	2	2	3	1	2

Week 4	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	2	2	1	1	1	1
Correctness	2	1	2	2	1	1
Malicious code vulnerability	0	0	0	1	0	1
Multithreaded correctness	1	1	0	0	0	0
Performance	1	4	2	2	4	2
Security	1	1	1	2	1	2
Dodgy code	2	1	1	2	1	1

Week 5	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	3	3	3	3	1	2
Correctness	2	2	3	3	3	1
Malicious code vulnerability	1	1	0	0	1	0
Multithreaded correctness	0	0	0	0	0	1
Performance	2	2	2	1	2	1
Security	1	1	1	1	1	2
Dodgy code	2	3	2	3	2	1

Week 6	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	2	3	2	1	2	1
Correctness	1	1	1	2	1	2
Malicious code vulnerability	1	1	0	0	0	1
Multithreaded correctness	1	0	0	0	1	0
Performance	2	1	1	0	2	1
Security	1	0	0	0	1	1
Dodgy code	2	3	1	2	2	1

Week 7	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	2	3	3	2	1	1
Correctness	2	1	2	2	1	1
Malicious code vulnerability	1	0	0	0	1	0
Multithreaded correctness	2	2	1	1	1	0
Performance	2	2	1	2	2	
Security	1	0	0	0	0	3
Dodgy code	2	2	2	3	1	2

Week 8	Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Developer 6
Bad Practice	3	2	1	2	1	1
Correctness	2	2	3	1	1	1
Malicious code vulnerability	1	0	0	0	0	1
Multithreaded correctness	1	0	1	0	0	0
Performance	2	1	0	1	3	2
Security	1	0	0	0	0	1
Dodgy code	2	3	2	1	2	1

APPENDIX B

Sample Java implementation of a basic parser for FindBugs log files.

```
import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class BasicFindBugsLogParser {

    public static void main(String[] args) {
        ArrayList<Bug> bugs = new ArrayList<>();
        Map<String, Integer> typeToNumberofBugs = new HashMap<>();

        File fXmlFile =
                new
File("path/to/log_file/Foresight.fbwarnings.xml");
        DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder;

        FileWriter fw;
        BufferedWriter bw;
        PrintWriter out = null;
        try {
            dBuilder = dbFactory.newDocumentBuilder();
            fw = new FileWriter("path/to/data", true);
            bw = new BufferedWriter(fw);
            out = new PrintWriter(bw);
            Document doc = dBuilder.parse(fXmlFile);
            doc.getDocumentElement().normalize();

            NodeList nList =
doc.getElementsByTagName("BugInstance");

            for (int index= 0; index< nList.getLength();index++) {
                Node nNode = nList.item(index);

                // Traverse and collect the relevant data
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    String type = eElement.getAttribute("type");
                    String rank = eElement.getAttribute("rank");
```

```
                    String priority =
eElement.getAttribute("priority");
                    String category =
eElement.getAttribute("category");
                    out.println(type + "\t" + rank + "\t" + priority
+ "\t" + category);
                    String key = priority;
                    if (typeToNumberofBugs.containsKey(key)) {
                        typeToNumberofBugs.put(key,
typeToNumberofBugs.get(key) + 1);
                    } else {
                        typeToNumberofBugs.put(key, 1);
                    }
                    bugs.add(new Bug(rank, type, priority));
                }
            }
            // Just to log the collected data
            for (Entry<String, Integer> entry :
typeToNumberofBugs.entrySet()) {
                System.out.println(entry.getKey() + "-" +
entry.getValue());
            }

        } catch (ParserConfigurationException | IOException |
SAXException e) {
            throw new Exception("Parsing failed", e);
        } finally {
            if (out != null) {
                out.close();
            }
            bw.close();
            fw.close();
        }
    }

    static class Bug {
        String rank;
        String type;
        String complexity;

        Bug(String rank, String type, String complexity) {
            this.rank = rank;
            this.type = type;
            this.complexity = complexity;
        }
    }
}
```