Static and dynamic analysis of Feedback Gathering tool

VINOD KUMAR KARANTOTHU
950326-8899
vika17@student.bth.se

SNEHITHA MANDEPUDI
960204-0124
snma17@student.bth.se

SYED MUDASSIR MANSOOR SHAH
920325-3977
mudy649@gmail.com

KANCHETI, SRINIVASA VASANTH CHOWDARY
960818-6830
srkb17@student.bth.se

ABDULLAH HOSSAIN
931012-2453
abho16@student.bth.se

NOMAN ASHRAF
810803-5331
noas16@student.bth.se

***Abstract*—This document is a report on static and dynamic analysis performed on an opensource software project Feedback Monitoring and Gathering tool. The tool was developed by BTH in collaboration with universities in Europe. In this project, we go both with static and dynamic analysis on the three components: Orchestrator, Repository, and Web Library. The components orchestrator and repository are written in Java where Web library written in Typescript. In this section, we initially set the inspection goals and then found the tools that are relevant to our defined goals. Every individual of the team accessed the tool and the results are discussed and reported.**

Keywords—Static Analysis, Dynamic Analysis

I. INTRODUCTION

Static automated code analysis refers to an automated process which is used to analyze the code without executing it. Static analysis takes less time than compared to dynamic analysis and it also covers more scenarios than the dynamic analysis [1]. Static analysis techniques explore the program's behavior for all possible inputs and which in turn accounts for all possible states that the program reach [2]. The main objective of this testing is to find out the bugs in early stage of a project. Static testing technique divides into two different parts; review and static analysis. Review is usually used to find out elimination or ambiguous error in the documentation of software like SRS, designing and testing. Reviews have 4 different formality levels. In static analysis, we need tools for analyzing the code. Compiler also can be considered as static analysis tool [3].

II. INSPECTION GOALS FOR STATIC ANALSYSIS

Inspection Goals

Inspections play an important role in static analysis. It is the most formal review type. It is led by the trained moderators. During inspection, the documents are prepared and checked thoroughly by the reviewers before the meeting. According to ISO/IEC 9126[6] standard there are 6 characteristics and these are divided further 27 sub-characteristics. There are sub-characteristics of maintainability; these are adaptability, changeability, stability, and testability [1]. To perform static analysis of the given tool, we considered the following quality attributes into the consideration: correctness, reliability, adequacy, learnability, robustness, maintainability, readability, extensibility, testability, efficiency and portability. But considering the time and people, each attribute cannot be tested. As we mention above maintainability cover some important quality aspect So, we choose maintainability sub-characteristics.

A. Miantainablity

A software product which can be maintained in several factor i.e. clearing defects and their occurrence reason, restore faulty components without replacing the working parts running in the software, controlling unanticipated working condition, increase product usefulness, increasing efficiency, reliability, security, reach out the new requirements or specifications in a changing environment. Maintainability includes a system of a periodical improvements while acquisition of knowledge from past maintenance information. Maintainability is measured on various metrics like lines of code (LOC), McCabe measures and Halstead complexity measures [7][8][9].

The reason for selecting the maintainability attribute is because we have identified issues that do not affect the product in its current form, but could lead to problems during further development. Another reason for selection this attribute for cost and time because maintains take time that's why it is costly, according to one survey maintains take 70% cost [2]. One example of a maintainability issue is multiple occurrences of the same string literal in different places: if these needs to be changed, all occurrences have to be changed and problems could arise if individual occurrences are omitted. Therefore, it may be beneficial to replace these duplicate string literals with named Constants Finding errors and bugs

B. Finding Bugs that may cause failures:

"bug" or "error" are both imprecise words, they both are used at different instances to relate wrong program code, incorrect method instances and incorrect program code behaviour that can be easily seen. One of the examples of those that come under is category to discuss is NullPointerException that evolved by dereference of null pointer. Unintended code path will cause the variables to initialize, these are of most usually cause types. If such a bug is identified in the source code that might lead to crash, causing functionality unavailable. There are several classes of errors/bugs were excluded from this analysis part due to work complexity or not possible to cover all category of errors[10][11][12].

C. *Coding standards and good practices*

Coding standards and good practices are kind of disciplined approach to write the code or make software projects [1]. Coding standard is best practice find out violations, defect patterns and nonconformance in source code, it would be unmaintainable code if we need to change in future [2]. Then it could be expensive if we not fix defect on early stage. The reason to adopt best practice approach to reduce the time of retesting and improve the time of market software [3]. After discussion we will point out all possible chance in this section[13][10][11].

1. *Smells in code*: Code smells is a runnable code without any bugs but code in this code should be written in some basic design principle. It would be effect on quality of code, the reason behind this it starts from downward to upward and it cause problem in system [13].
2. *Code Style:* Readability is fully dependent on code style. Readability should be low if the code written in disorder, Code should be written with same style. Something in the code make project complex like multiple class declaration, package name not matching with project.
3. *Control flow:* Control flow is about sequence of statement execution. To avoiding this problem usage of conditional statement should be reduced. Conditional statements make the program complexity high

III. SELECTION OF TOOLS

In the past decade testers have been used manual verification to perform their work but now a days there are many automated tools are available for static code analysis. Static automated testing tools are the tools which are used to understand the structure of the code and enforce the enabled standards of coding. We have come across various tools while searching for the tools to run static analysis. But not all tools are applicable for our project because the inspection goals which we designed doesn't match with the tools. So, we decided to filter the tools on the basis of our inspection goals. Below are the tools which we selected for the static analysis. We found several tools to carry out the JAVA static code analysis, such as CheckStyle, FindBugs, PMD, SonarLint, SonarQube and JTest. And finally, we choose PMD and FindBugs as our testing tools.

Reasons for accepting and rejecting the tools:

I. *PMD*

The reason behind for selecting this tool because it allows to select different kind of instruction and provide also warning based on our wanted project goals. For an example having try to empty catch block it might indicate the errors and PDM includes supports of selection detectors which should run. In our experiments we basically run PMD by the documentation of java code. In PMD provides us new bug pattern detectors using our java code. In that reason we have chosen PMD tools as our static code analysis[4].

This tool helps to find the bugs/errors in the code, identify the coding patterns which helps in assessing the coding standards and code smells[13].

II. *FindBugs*

The reason behind for selecting this tool because of it allow the simple integrate of Eclipse IDE throw a simple plugin and for the time been of each warning it provides the rank wise category. Though we are not preferring to use the same categories most of the time but it may help a lot to make decision based on group of warning. The main positive things of this tool are it can run any virtual machine accordant with the Sun's JDK[12][11][13].

III. *CheckStyle*

CheckStyle it is one of the static code analyzing tool which has been used in software development for checking java source code with a coding rules. CheckStyle was rejected as it checks only code clarity and all other quality attributes with our goals can't achieved with this tool. For adapting a software development project this tool can help to comply with good programming practices which has been helps to improve our code re-usability, readability and make sure the code quality.

For the reason we are not selecting this tool because of From the projects where this tools is utilized and mentioned in the articles, CheckStyle tool will report on the ton of things that are completely irrelevant and not useful also[13].

From our investigation we have found PMD and FindBugs are the two best among the bug finding tools to report more false negatives. CheckStyle does the similar things to PMD, they both come with set of warning patterns and locates them in code matching these patterns. PMD offers many warning patterns categorised under coding standards such as use of braces around code block class design problems, code smells, used variables, unnecessary object creation etc. this helps to achieve our goal to identify coding flaws. Compared to CheckStyle, PMD offers to measure the cyclomatic complexity which helps to address the maintainability. We have found that CheckStyle reports a greater number of false positives for example name of the functions and classes with letter case suggestions like such etc. Considering all these factors CheckStyle was rejected[15][16]

IV. *SonarLint*

SonarLint is also the tool that may allow the java language code and it may also integrated with Eclipse IDE throw a plugin. We are selecting this tool because it suggested us code as a good explanation. This tool helps us to automatically expectation when the java file open and this tools also provide different level of categories based on our code simulation. The main interesting this of this tool is it provide a quick bug fixing ability.

For this time being we are not selecting SonarLint because of we can't get any additional support from java code. It can only work for super- fast if it is an embeds only own of analyzer otherwise it's not quite good for static bug fixing[13].

V. *DeepScan for TypeScript*

The main reason behind selection of this tool is, one of the packages in the project is based on Typescript. The tools FindBugs and PMD are connected with java programming language, that are can't applicable for typescript. In our search we found this primarily for JavaScript and Typescript code static analysis tool and had a good impact with its functionalities. One of the most adopted free source tools for JavaScript and TypeScript projects[10].

The Web Library component in this project is written in TypeScript instead of direct JavaScript therefore it needed a specific analyser to handle TypeScript. PMD does same thing for this component finding coding flaws if it was written purely in JavaScript so we had thought of another tool. Firstly, we thought of TSLint but while inspecting rulesets available it reports only few that related to semantics. Then we came across DeepScan tool which is a new tool. It has efficient rulesets that find bugs and code smells more exactly with data flow analysis and also reports coding conventions[17].

VI. STAN

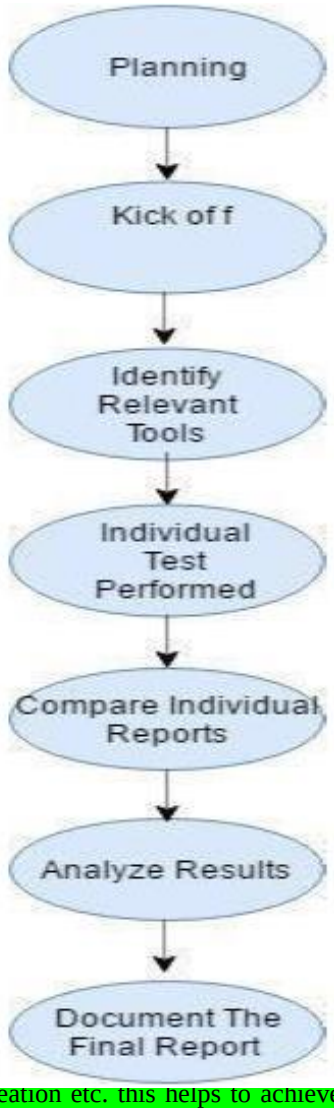
STAN (Structure Analysis for Java) is static analysis tool used for calculating metrics for Java based projects. This give the metrics such as LOC (Lines of Code), Complexity, Cohesion, and coupling etc. which helps assessing the maintainability of code in terms of understandability, readability, modifiability.
Reasons behind selection of this tool is easy to integrated with eclipse IDE as it is available as plugin from the eclipse market store. Another reason is accuracy in giving metrics and available as open source tool and performance even on large size projects[14].

S.no	Accepted tools	Characteristics
1	Find Bugs	- each warning it provides the rank wise category, suggestion to the tester about - warning families including: multi-threade ctness, risky coding practice, style etc
2	PMD	- warning families including: braces arou blems, code smells, used variables, unnecessary object creation etc. this helps to achieve our goal to identify coding flaws - PMD offers to measure the cyclomatic complexity which helps to address the maintainability.
3	DeepScan	-Supports TypeScript - efficient rulesets to find bugs and code smells more exactly with data flow analysis. -reports coding conventions.
4	STAN	- This tool gives the metrics such as LOC (Lines of Code), Complexity, Cohesion, and coupling etc. which helps assessing the maintainability of code in terms of understandability, readability, modifiability.

IV. INSPECTION PROCESS

First, we downloaded the monitor feedback tool from the GitHub. We then did some literature study on inspection goals. Based on our inspection goals, we toolbar.

searched for the relevant tools. After selecting the tools, each of the individual run the tools and then reported their decisions. Then everyone came as together and discussed everyone's decisions and then integrated into single report. Everyone had their own opinions but to deliver the project correctly we considered everyone opinion and came to a decision. Then we reported the final document. Our inspection process is as given below.



V. ANALYSIS OF THE OUTCOMES

After installing static tools, we run over the code and observed some errors in it. Out of all errors found, some errors are true (they affect the program software), some errors are false (they do not affect the program software) and some errors which are uncertain i.e. we could not decide their behavior. Based on the observation made, we categorized errors into three types.

1. True positive: The error is true and it will affect the program.

2. False positive: There may be error but it does not affect the program.

3.Uncertain: These are the warnings which are uncertain and we cannot decide whether it affects the software or not.

Based on these three categorized errors, we inspected each error and reported our decision about that, which then we came together and discussed about our individual decisions and a final decision is made. One major problem which we faced is that most of the errors are repetitive. We observed that the same errors are occurred in different parts of the program and each error occurring at a particular part of the code has its own way of affecting the code. So, we avoided the errors which are repeated more than once Each one of them is observed carefully and the results are drawn out of it. Individual reports are collected and then integrated into a single report.

A. *Individual assessment*

Every individual did their own observations and the decision on the errors are carefully reported. We reported our decisions in one excel sheet. The errors were reported as true positive if that error affects the program, false positive if that error does not affect the program and uncertain if we are not certain about the error. There is some difference in the opinions of different members and everyone had their own argument whether some error is true positive or false positive. All of them considered and the important thing is to report the document correctly. The difference in the opinion is mainly because of the difference in the individual perspectives, which is later eliminated by discussing in the group. By reviewing the entire test, a total of 90 warnings were found on this basis, our team.

Members	No. of warnings	TP	FP	Uncertain
Vinod	103	52	30	21
Mudassir	85	47	25	13
Abdullah	70	40	12	25
Noman	79	34	28	17
Snehitha	95	45	39	11
Consensus	90	50	29	11

Reflection of Results:
While assessing warnings we categorized all the outcome into LOW, MED HIGH. Where low are the warnings that intend no threat for sample are coding standards and related. Med are the warnings related to threat when the source code develops in size for example warnings related to efficiency and complexity under maintainability. High is the warnings considered to threats immediately for corrections, these are the major bugs.

From the results sheet, the most of bugs reported by the tool are related to warnings for the function names and variables. These are indicated as false positives which may not affect the programs and no immediate are required to modify them. And the other are related to class complexity, braces around code blocks, style, redundant and dead code warnings. These are listed under medium which require an attention to deal with them to achieve the maintainability of the code to allow future modifications to project with less difficult. Class complexity is reported by the tool PMD. This efficient in addressing the maintainability of the code. The warnings under HIGH are related to multithread correctness, multiple variable dependency, Exception handling and risky code patterns which may cause the program failures during run time. These warnings need to be resolved immediately to ensure a reliable software application [18][19].

These tools are efficient in achieving the inspection goals such as maintainability, finding bugs that might cause failures and evaluating the coding standards by the reporting the warning types multi-threaded correctness, bad practice, correctness, risky coding practice, style , code block class design problems, code smells, used variables, unnecessary object creation and class complexity.

VI. CONCLUSION

After conducting the automated static testing, we have drawn our conclusions based on the warnings obtained. conducted a discussion, of which 50 true positive warnings were found, 29 false positives and 11 warnings could not be determined.

B. *Performance of the tools*

PMD: PMD is a Java code detection tool that uses the BSD protocol, it has high scanning efficiency and can make up for FindBugs to detect empty try / catch / finally flaws, and it can generate reports so that programmers can collect information from it.

FindBugs: FindBugs is a bug detection tool for java code, now its function has been quite perfect, it has a detector, including the bugs like Bad practice, Correctness, Internationalization, Malicious code vulnerability, Multithreaded correctness, Performance and Dodgy. This tool does not have much flaws, but in some aspect like array usage and empty try / catch / finally class, it still is not detected.

DeepScan:
It is one of the efficient tools for we have noticed from our work and from literature for JavaScript and TypeScript based projects. Regardless of size of the project size and dependencies it runs well as it should giving effective results.

STAN:
It is a light weight tool which easily integrated with eclipse IDE and gives metrics very accurately. It output the results in the graphical format, where easy to visualize the results and assess the maintainability of a software.

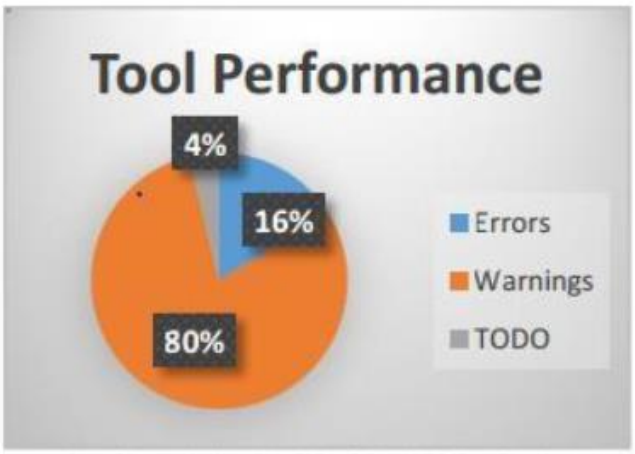


Figure 2: An analysis of Tool Performance

We found from our study that the reliability of the application is at a greater risk, and security is relatively good.

REFERENCES

[1]: Heitlager, Ilja, et al. "A Practical Model for Measuring Maintainability." 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), 2007, doi:10.1109/quatic.2007.8.

[2]:Hashim,K.andKey,E.,1996.Asoftwaremaintainabilityattributesmodel.MalaysianJournalofComputerScience,9(2), pp.92-97

[3]:Asadi,Morteza,andHassanRashidi."AModelforObjectOrientedSoftwareMaintainabilityMeasurement. "InternationalJournalofIntelligentSystemsandApplications8.1 (2016): 60-66. Web.

[4]: Hopkins, Paul. "Secure Systems Development and Secure Coding." Engineering & Technology Reference, 2014, doi:10.1049/etr.2014.0016.

[5]: Muslu, Kivanc, et al. "Reducing Feedback Delay of Software Development Tools via Continuous Analysis." IEEE Transactions on Software Engineering, vol. 41, no. 8, 2015, pp. 745–763., doi:10.1109/tse.2015.2417161.

[6] P. Botella and J. P. Carvallo, "ISO/IEC 9126 in practice: what do we need to know?," no. January, 2004.

[7] B. Scholz, G. Denaro, and U. Milano-bicocca, "Automated Software Testing and Analysis : Techniques , Practices and Tools," p. 7695, 2007.

[8] S. P. Corina and W. Visser, "A survey of new trends in symbolic execution for software testing and analysis," pp. 339–353, 2009.

[9] T. Xie, "Cooperative Software Testing and Analysis : Advances and Challenges," vol. 29, no. July, pp. 713–723, 2014.

[10] "Detecting JavaScript Errors and Code Smells with Static Analysis," no. April, pp. 1–16, 2018.

[11] A. Vetro', M. Torchiano, and M. Morisio, "Assessing the precision of FindBugs by mining Java projects developed at a University," *Proc. - Int. Conf. Softw. Eng.*, pp. 110–113, 2010.

[12] N. Ayewah, D. Hovemeyer, J. D. Morgenthaler, J. Penix, and W. Pugh, "Using Static Analysis to Find Bugs," *IEEE Softw.*, vol. 25, no. 5, pp. 22–29, 2008.

[13] S. Wagner, J. Jürjens, C. Koller, and P. Trischberger, "Comparing Bug Finding Tools with Reviews and Tests," 2017.

[14] "STAN - Structure Analysis for Java." [Online]. Available: http://stan4j.com/. [Accessed: 20-Aug-2018].

[15] F. Thung, Lucia, D. Lo, L. Jiang, F. Rahman, and P. T. Devanbu, "To what extent could we detect field defects? An extended empirical study of false negatives in static bug-finding tools," *Autom. Softw. Eng.*, vol. 22, no. 4, pp. 561–602, 2015.

[16] S. H. Edwards, N. Kandru, and M. B. M. Rajagopal, "Investigating Static Analysis Errors in Student Java Programs," *Proc. 2017 ACM Conf. Int. Comput. Educ. Res. - ICER '17*, pp. 65–73, 2017.

[17] "Detecting JavaScript Errors and Code Smells with Static Analysis," no. April, pp. 1–16, 2018.

[18] H. Khalid, M. Nagappan and A. E. Hassan, "Examining the Relationship between FindBugs Warnings and App Ratings," in IEEE Software, vol. 33, no. 4, pp. 34-39, July-Aug.2016.

[19] ""H. Shen, J. Fang and J. Zhao, "EFindBugs: Effective Error Ranking for FindBugs,"2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, Berlin,2011, pp. 299-308.

Part 2
Dynamic Analysis

Abstract*— *In dynamic testing the software is deployed into a server-side and made to run. In this Feedback Monitoring and Gathering tool is to be deployed on to a system where using Apache Tomcat and MySQL. Later a dynamic testing tool is selected and performed dynamic testing at different levels and logs are to be recorded and analysed. The purpose of dynamic testing is to observe the behaviour of code.

I. INTRODUCTION

Context

In dynamic testing the entire software and parts of it are to be run on the system and resulting warnings and bugs are to be logged This thing is made happened after the static analysis that was described in the previous section part-I. In this part of dynamic testing we need to select a tool to perform dynamic testing before we need to define goals for this analysis. As there are many limitations of the tool on which we need to perform the dynamic analysis the next section describes the software and its limitations.

Background

The tool Feedback Monitoring and Gathering tool on which the dynamic test is to be performed on three components (Orchestrator, Repository and Web Library) that are written in Java and Typescript languages respectively. As mentioned in the project description need the deploy the tool but none were successful doing that as there are many issues were raised and source didn't give much information about configuring this tool [5].

we were about to use the demo version of the tool available on [ref: http://demofeedback.ronnieschaniel.com/] instead deploying it on our local system. So, for the form we could do acceptance testing, therefore we tried to find a tool that would help in doing it.

Acceptance testing is of validation of software's requirements. In this case, we did not have any such information available which meant we could not do any validation. The feedback tool is aimed at webpage visitors that most likely do not have any previous experience or knowledge about this specific tool. For Web Library component, we thought about another testing of the methods that were identified by Sonar Lint tool during Static testing. After having looked in the code of these functions, we decided that they are more purely handled with rendering each element on the form. We felt testing these on these aspects would turned difficult without resulting proper results, therefore the test is repeated.

II. INSPECTION GOALS

Dynamic analysis is done by the real-time analysis to perform testing and examine the program.

Goals[6][7]:

1) To verify the functionalities of the tool (Feedback Monitoring and Gathering tool) (main functionalities are giving feedback with screen capture, voice message and text message as inputs is).
   The functional requirements specification describes what the system must do (available at https://github.com/supersede-project/monitor_feedback). Therefore, in our case, the tool must be able to do the following:
   • Submit the feedback with screen capture.
   • Submit the feedback with voice input.
   • Submit the feedback with text message
2) To check the behaviour of the tool on different platforms (As it is a web-based tool we target on different browsers)
3) To test the performance of the tool in terms of responsiveness for multiple feedback inputs.

The performance of the tool in our case will be measured in terms of response time. The performance requirements of the tool will be considered according to the book written by Jakon Nielsen [11]in which general advice on response time is given.

- A response time of 0.1 second means that the system is reacting instantaneously, therefore no special feedback is required.
- For a response time of more than 0.1 second and less than 1 second, there is no need for special feedback, though the user notices the delay.
- A response time of 10 seconds is the limit for user's attention to be focussed. For longer delays, feedback should be given letting the users know the response time, especially when the response time is highly variable.

In our case, the response time is taken in terms of Average feedback submission time, which is the time taken by the browsers in taking the input from the user and submitting the feedback to the user.

4) To evaluate the easiness of the tool interface in terms of usability and interaction.

The key purpose to do dynamic test is to assure consistency to the software not only restricted to functionality but also to different standards such as usability, performance compatibility.

III. Tool Selection

Selecting the tools

As in project description we need to perform the both acceptance and integration testing. We gone through some literature available online and selected Selenium and Robot Framework to do the dynamic analysis [4].

As we selected Selenium tools stills there are two versions available in that one is Selenium IDE and other one is WebDriver plugin. Web driver is tool which integrates with API (Application Interface Programming) and allows to write the scripts in more than one programming language available.

Reasons for selection of Selenium IDE and WebDriver plugin[8]*:*

- This features the integration of web driver API. WebDriver's goal is to give an excellently designed object-oriented API which furnishes better support for currently advanced web-app testing issues.
- Its functionality is to make direct calls to the web browser and then the whole test script is run. And it utilizes browsers support and its ability to automation.
- We selected it because it communicates directly with web browser without any outside involvement. So that it takes advantage of the browser's owned abilities towards automation. It also provides access to user to run web-based mobile testing.

Reasons for selection of Robot Framework[9]*:*

- Gives easy-to-use tabular syntax for design and produce test cases in a uniform way.
- Gives capability to generate reusable intense level keywords from the existing keywords.
- Gives easy-to-read result reports and logs in HTML format.
- Is a platform and application non-dependent.
- Gives a simple library API for generating customized test libraries which can be executed natively with either Python or Java.
- Gives a command line interface and XML driven output files for integration into existing build infrastructure (continuous integration systems).
- Gives aid for Selenium for web testing, Java GUI testing, Telnet, SSH, and so on.
- Assists in creating data-driven test cases.
- Has built-in assist for variables, empirical specifically for testing in unlike environments.
- Gives tagging to classify and select test cases to be implemented.

- provides easy unification with source control: test suites are just files and directories that can be categorized with the generated code.
- Gives test-case and test-suite level structure and teardown.
- The modular architecture supports generating tests even for applications with various interfaces.

We firstly referred to the tool documentation directly from their websites to understand the capability of the tools. As this is a web-based project, the testing tools need to support them. The most common characteristic of both tools is that they use web API libraries to call the actions in the browser. Robot framework is dependent on the selenium web driver libraries. These two tools used for functional testing and testing the performance of the system under test on different platforms. The main reason selecting these once the test script developed for selenium web driver, robot framework makes use of the keywords from the test scripts and runs the test in browser.

Opposing to static analysis, Dynamic analysis examine the code during the runtime of the software. During the development of the test scripts many issues may prevail such as performance, compatibility and other, one reason to use these tools is to solve these problems. Since there are wide range of tools available considering inspection goals for dynamic analysis this tool has been selected.

Supporting to the goals, using these tools test scripts are developed based on the test cases to verify the functionalities of the Feedback Gathering tool to achieve acceptance testing. And to check the performance of the feedback tool on different browsers the same test cases are run calculating the test completion time. To address the goal to evaluate the easiness of interface in terms of usability and interaction individual user satisfaction is collected from our team members.

IV. Review Process

Keeping a view on the work process we followed doing static analysis we started initiated dynamic analysis with past knowledge. The process steps are as such provide below.

1. Pointed the dynamic analysis tools and section is done for analysis
2. Assured the tool which is to be used.
3. Developing the test cases, scripts and testing the tools during the runtime of the program
4. Check the performance of the tool
5. Compared the performance of the tool
6. Data analysis by among the team by discussion
7. Data collection
8. Documented the dynamic analysis process

V. Results

As shown in Tables below, are the results of the tools during running test scripts.

Selenium WebDriver					
Errors	T	F	U	Total	
Implementation errors	15	19	5	39	

T: Implementation errors F: Code Errors

Table 1: Results table for Robot Framework

Robot Framework				
Errors	T	F	U	Total
Implementation errors	13	20	3	36

T: Implementation errors F: Code Errors
Table 2: Results table for Robot Framework

Post of selecting the tools Selenium and Robot Framework for functional testing of the feedback monitoring tools, a number of test cases are created and after the test cases are created we implemented all the test cases and outcomes of the test cases from the both tools are collected. We have categorised all the negative outcomes into two points on the basis of whether the test case is an implementation error or code errors.

Implementation errors: implementation errors are those where the test scripts are coded well but we failed in executing them because of one reason browser compatibility.

Code Errors: Code errors are the errors in the test scripts where execution is not the actual problem. It may be syntax error or deference of the variables and identifiers.

Implementation errors are the errors that arise due to the failure of executing the test scripts, because of the incompatibility of the browser, even though the test scripts are coded well. Code errors are the result of bad code in a program that is involved in producing the erroneous result. The execution may not be the problem, but it can also be a syntax error which is an error resulting from code that does not conform to the syntax of the programming language[11].

This dynamic analysis focus on the executing the test cases, keeping in context to verify the result out comes and estimated the results of the comparisons, and gathering the dynamic analysis in context to the goals.

This part of dynamic test analysis includes:

- Functional and interface validation
- Performance analysis
- Usability

Results Reflection:

Usability of Feedback tool: Data is collected on individual user satisfaction using the tool by submitting feedback on different browsers. The satisfaction levels are easy, medium and hard.

Easy- easy to understand options and use

Medium- not as easy interacting with options

Hard – difficult to understand and use the tool[10].

User	Satisfaction level
Vinod	Easy
Mudassir	Easy
Snehitha	Medium
Noman	Medium
Vasanth	Easy
Abdullah	Easy

Average result: the tool is easy to use

Acceptance Testing: This Feedback gathering tool met all the requirements we have identified through the project description in context to the functions and passed all the test cases[6][7].

Test Case 1: submitting feedback with screen capture.

Test Case 2: submitting the feedback with voice input.

Test Case 3: submitting the feedback with text message.

Test scripts are developed based on the functional requirements of the Feedback gathering tool. The primary goal is to perform functional testing. These two tools support doing the functional testing. Once the test script is executed the actions were called in to the browser. The input data is supplied based on the function's specification and is embedded in test script. The same test case is executed on the different browser to measure the performance of the application on different browser platforms based on the average time for test execution.

Behaviour of the tool and responsiveness on different browser:

Test Case/Browser	*Chrome*	*Edge*	*Firefox*

Average feedback submission time (in Seconds)			
1	*25secs*	*19Secs*	*21secs*
2	*15 secs*	*15secs*	*13 secs*
3	*15secs*	*15secs*	*15secs*

The results show that there is no much difference between the browsers in taking input from the user and submitting the feedback.

This is the average test execution time. This data is collected running test cases by every individual in the team on their own systems and different browsers (chrome, Firefox, edge). Same test cases are employed and the input data is same and is imbedded in the test script. No manual input method is done during the test cases execution. The test runtime time is measured and this allows user to evaluate the responsiveness of the system under test.

VI. Conclusion

To our first goal for functional testing, the results provided clearly indicate that system under test met all the functional requirements of the feedback gathering tool described on its GitHub page. The functional testing is performed based on the functional specification available on tool description README.md file. Since there is no full documentation of feedback gathering tool is available this testing limited to some functional requirements, this is an be extended further testing if complete tool specification is provided.

The results for usability test clearly show that the feedback gathering tool is easy to use. Easy to understand the options. New users can easily understand options with no requirement of more additional time or feeling difficulty. Coming to the behaviour of the tool on different platforms, it is being very responsive to the user input. But the feedback b button chrome Browser is behind the window frames and is sometimes feels difficult to click on it, this can be improved in the further development. Many times, the rating option is unresponsive but we noticed it due to browser compatibility. The developer should also concentrate on this part.

The selection of the tool for this dynamics analysis satisfied the inspection goals. The capability of the tools agrees with goals. The tools are efficient in assessing usability, reliability performance of the tool.

This dynamic testing is aimed at executing the testing tools, considering to investigate the test results and differences of the results, finally test results are gathered and for the two where categorised anteriorised in to tables. Finally, we conclude that the test results from both the tools are efficient and accurate at relatively equal levels in all test cases.

REFRENCES

[1] K. P. and L. L. Dejan Baca*,†, Bengt Carlsson, “Improving software security with static automated code analysis in an industry setting,” *Softw. - Pract. Exp.*, vol. 39, no. 7, pp. 701–736, 2012.

[2] A. Datta, S. Jha, N. Li, D. Melski, and T. Reps, *Analysis Techniques for Information Security*, vol. 2, no. 1. 2010. [3] “Static Testing.” [Online]. Available: https://www.tutorialspoint.com/software_testing_dictio nary/static_testing.htm. [Accessed: 15-Apr-2017].

[4] B. Haugset and G. K. Hanssen, “Automated Acceptance Testing: A Literature Review and an Industrial Case Study,” in Agile, 2008. AGILE ’08. Conference, 2008, pp. 27–38.

[5] N. Rutar, C. B. Almazan, and J. S. Foster, “A comparison of bug finding tools for Java,” in 15th International Symposium on Software Reliability Engineering, 2004. ISSRE 2004, 2004, pp. 245–256.

[6] C. Padmini, “Beginners Guide To Software Testing,” pp. 1–41, 2004.

[7] A. Arora and M. Sinha, “Web Application Testing: A Review on Techniques, Tools and State of Art,” *Int. J. Sci. {\&} Eng. Res.*, vol. 3, no. 2, pp. 1–6, 2012.

[8] "Selenium WebDriver." [Online]. Available: https://www.seleniumhq.org/projects/webdriver/. [Accessed: 20-Aug-2018].

[9] "Robot Framework." [Online]. Available: http://robotframework.org/. [Accessed: 20-Aug-2018].

[10] I. Certification, P. Guide, and D. T. Techniques, "Chapter 3 : Dynamic Testing Techniques," no. October, pp. 1–9, 1992.

[11] J. Nielsen, "Usability metrics: Tracking interface improvements," *IEEE Softw.*, vol. 13, no. 6, Nov. 1996.