Why Is Dynamic Analysis Not Used as Extensively as Static Analysis: An Industrial Study

Radhika D. Venkatasubramanyam
Siemens Corporate Research and Technologies, India
Siemens Technology & Services Pvt. Ltd.,
Bangalore, India
91 80 33132069
radhika.dv@siemens.com

Sowmya G. R.
Siemens Corporate Research and Technologies, India
Siemens Technology & Services Pvt. Ltd.,
Bangalore, India
91 80 33133896
gr.sowmya@siemens.com

ABSTRACT
Code Assessments using static and dynamic analyses are important for the maintenance of code quality of software in the industry. These analyses, though understood to be beneficial, have several practical limitations. The intent of our study was to understand the usage of these analyses across various teams in Siemens during code assessments, reasons for the success of static analysis and the challenges in dynamic analysis. The evaluation was conducted through a survey of the practices of developers in different development teams. Additionally interviews were conducted with expert reviewers and their experiences documented. Our study has revealed that institutionalization of code quality initiatives and customized toolkit support for static analysis are some of the key reasons for the successful implementation of static analysis. Some of the problems hindering the widespread adoption of dynamic analysis include the direct dependency with test cases, scalability, and understandability of the results. Various implications of the results are discussed, including the possible improvements that could increase the acceptance of dynamic analysis on a regular basis.

Categories and Subject Descriptors
D.2.9 [**Management**]: Software Quality Assurance

General Terms
Measurement, Performance, Reliability, Experimentation, Security, Human Factors, Standardization, Languages, Verification.

Keywords
Code quality, code assessment, static analysis, dynamic analysis, software quality

SER&IPs'14, June 1, 2014, Hyderabad, India
ACM 978-1-4503-2859-3/14/06
http://dx.doi.org/10.1145/2593850.2593855

1. INTRODUCTION
Evaluation of the quality of the software is of utmost importance, considering the pervasiveness of software applications in the real-world and the necessity to ensure they are bug-free. Increasing complexity of software and its maintenance has persuaded researchers and practitioners to accentuate on several formal and practical methods towards code analysis.

Software quality consists of external and internal quality. External quality measures whether the system adheres to the requirements and functionality as specified by the stakeholders. Internal quality provides a check on the non-functional requirements aided by static analysis, dynamic analysis through profiling memory and performance, and human analysis that includes manual inspections, code assessments and program comprehension. Code assessments enable continuous and consistent checks on the adherence to software quality. This is vital to the industry in order to deliver error-free applications, while also reducing time and resource overheads.

Static analysis involves inspection of the program code, without program execution and regardless of the input space. Static analysis is based on formal methods with most tools providing a shallow analysis which does not model the heap, not analyze data flow or do alias analysis. The analysis is performed on an abstracted model of the program flow. So a high degree of imprecision is observed in these tools. Dynamic analysis, on the other hand, is based on run-time observation of the code using instrumentation. Instrumentation is performed at source-level or at run-time that determines the time taken for execution of program under observation. Dynamic analysis involves testing, debugging and profiling. Test techniques like functional test, integration test, regression test etc. provide insight into the external quality of the software. In this paper, we focus on the profiling aspect of dynamic analysis to detect performance and memory issues, which help uncover issues related to internal quality attributes such as maintainability, efficiency and reliability.

Though the main purpose of both static and dynamic analysis is program verification, the output sets are disjoint and the two methods complement each other [1][22][23][24]. There are several papers and articles that discuss the technical differences between the two types of analyses [25][26]. In this paper, we compare the reasons for the use and underuse of these analyses as we believe that both static and dynamic analyses are equally important and integral to code assessments. A balanced use of both analyses is crucial to the maintenance of internal quality of the software. Our study intends to provide a comparison between the two types of analyses based on the practicability of their usage

for code assessments and does not intend to compare the inherent characteristics of the two analyses themselves.

The industry believes that the usage of assessment methods like static and dynamic analyses is imperative; however their effective utilization is a challenge. The intent to use them is not essentially an 'after thought' process but more as an inherent part of code development. However, real-world software projects always run short of time and resources and hence many important issues are delegated to a later stage, and only when it manifests as a significant problem do developers use dynamic analysis as a means to detect and fix these issues. At this stage, usage of dynamic analysis tools can become overwhelming. And in most of the cases due to time and resource limitations most developers either abandon the usage and resort to the usual debugging techniques to resolve problems or rely on expert guidance to solve the issues, rather than include dynamic analysis as a part of the project development lifecycle.

The goal of this paper is twofold: To primarily understand how code quality checks are performed and how static and dynamic analysis tools are used during the project lifecycle in the industry. Secondly and more importantly, to understand why dynamic analysis is seldom used by the developer community in comparison to static analysis in addressing code quality issues effectively. Our research also aims at realizing the challenges posed in getting considerable acceptance by the practitioners.

The scope of this study consists of the development projects in Siemens Corporate Technology Development Center Asia Australia (CT DC AA). Our investigation involves understanding the current practices in code review, experiences in tool usage (discussed in Section 3.2), result analysis and feedback from the participants on the expected improvements that could enrich the code review process. A survey was conducted by interviewing about 100 developers from various CT DC AA project teams and expert reviewers.

In the next section we provide an overview of the survey methodology. We discuss the results of the survey involving interviews with the developers and experts in section 3 and section 4. Section 5 contains a discussion of the results and some threats to the validity of our study. Related work is summarized in section 6, followed by conclusion and future work in section 7.

2. OVERVIEW OF THE METHODOLOGY

The examination of the current practice of static and dynamic analyses in Siemens was done by interacting with both the developers and the review consultants through survey questions and discussions. The study was aimed to reveal the usage of such review methods in the industrial context and to understand their shortcomings and benefits; specifically to enable us to understand the role and importance of these reviews and the tools used for code quality checks.

The methodology adopted involved the following:

- *Opinion / Feedback from CT DC AA project teams*

This involved interviewing some of the teams and individual developers working on different types of projects (legacy code maintenance/ new application / iterative development).

- *Expert-review experience*

In order to maintain the code quality across Siemens projects, code quality assessments are being carried out on a regular basis. To facilitate these initiatives, experts from CT RTC (Corporate Research and Technologies) provide consultancy in static and dynamic analysis assessments. We have also taken opinions from these consultants to gather better insight into the usefulness of the analyses. In this section, we will briefly outline the mechanism adopted to carry out the study.

2.1 Survey Methodology

The survey was intended to get inputs on the practice of code assessments in the course of project development. The participants consisted mainly of developers and architects with an overall experience in the industry ranging from 2 to 15 years.

In order to get a complete insight on their experience with static and dynamic assessments, we wanted to get answers to the research questions listed below.

- **RQ1:** Are code quality checks using static and/or dynamic analysis performed?
- **RQ2:** Do project teams use both static and dynamic analysis tools to check the quality of their code? If so, which is preferred?
- **RQ3:** Are the tools used only on a need basis (reactively on encountering issues) or on a periodic (proactive) basis?
- **RQ4:** How are results from these tools processed?
- **RQ5:** What improvements would you like to see in Static/Dynamic Analysis tools?

2.2 Questionnaire

We created a questionnaire that was divided into four categories with the objective to obtain answers to the above questions. Some of the questions were qualitative in nature and some were meant to quantify to get a statistical overview.

2.2.1 Part A: Current Practice in Code Review

The questions in this category were aimed at getting answers to RQ1. The attempt was to understand the big picture with respect to code assessments and study the existing practices, with a focus on static and dynamic analyses. Some of the sample questions we asked here were:

- *How are Code Quality checks done in your projects?*
- *Do you think using Static/Dynamic Analysis tools increases code quality?*
- *What are the reasons for adopting Static and Dynamic analysis tools?*
- *What tools do you use for Static/Dynamic Analysis?*

2.2.2 Part B: On Static Analysis and Dynamic Analysis Tool Usage

The primary focus of this category was usage of Static/Dynamic Analysis tools. The questions in this category were aimed at finding answers to RQ2 and RQ3. Understanding the pros and cons of the tool usage and the degree to which teams benefit from these tools, are important factors to establish justification of the need to use these analyses regularly in code assessments. Some of the sample questions we asked here were:

- *At what stage are Static/ Dynamic Analysis tools used?*
- *Do you have any positive/negative experiences in usage of Static/Dynamic Analysis tools, what are those experiences?*

2.2.3 Part C: Results of Tools

This category addressed RQ4. The rationale behind this question was to understand the effectiveness of the results and its degree of usefulness in the industrial context. Some of the sample questions we asked here were:

- *How are the results obtained from Static/Dynamic Analysis tools used?*
- *Are the results easy to interpret?*

2.2.4 Part D: Suggested Improvements

This category addressed RQ5. It is best to elicit the suggestions for the required modifications for the existing tools from the practitioners themselves. We also gather the developer's viewpoint in the level of satisfaction in tool usage. Some of the sample questions we asked here were:

- *What problems do you think need to be addressed in these tools?*
- *How satisfied are you with the usage of such tools?*

Some of the participants also elaborated on the challenges with illustrations from their projects.

2.3 Consultant Experience with Assessments

In Siemens CT DC AA, consultants are primarily designated experts in relevant areas of code analysis and support teams to perform code assessments. Consultants facilitate the analyses on the projects and present the team with a detailed report on the issues. The consultants carry out static analysis through EMISQ (Evaluation Method for Internal Software Quality) [6] and dynamic analysis through DAISQ (Dynamic Analysis for Internal Software Quality) [9]. EMISQ is a methodology for systematically assessing the internal software quality through static analysis. Both quality and technical models have been defined in this. The EMISQ quality model is defined in accordance with the quality model defined by the ISO 9126 [7][8] standard consisting of 5 different quality attributes on the top level - Maintainability, Reliability, Efficiency, Security and Portability. DAISQ is a process to systematically assess the internal software quality through dynamic analysis. The methodology involves identification of run-time defects by executing the application under review on a selected workflow using dynamic analysis tools. The DAISQ quality model is the same as the EMISQ model. The Technical model consists of attributes of Performance, Concurrency, Memory, Exception Handling and Language Specifics as the top level attributes. Consultants summarize the results and present it to the project team from both quality and technical perspectives using quality and technical models respectively. Our aim is to interact with the consultants in order to uncover the implications of the analysis methodology and tool usage in the project context.

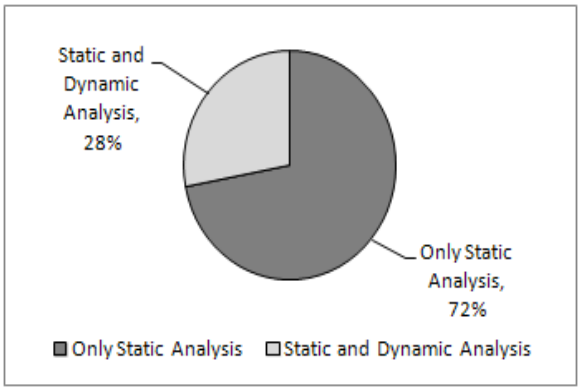

Figure 1: Static and Dynamic Analysis Distribution

3. SURVEY RESULTS

In this section, we discuss the insights obtained from the survey activity. The questionnaire was circulated to 100 developers working in different types of projects and from varied domains in Siemens CT DC AA.

3.1 RQ1: Are code quality checks using static and/or dynamic analysis performed?

The discussion with our respondents provided insights on the existing practices for code assessments.

3.1.1 Code Assessment in Projects

It was revealed that all of the 100 respondents working in various projects performed code assessments using code inspections / manual reviews and also tool based reviews which included static/dynamic analyses. All the respondents concurred on the view that code quality checks are an integral part of the project lifecyle and are indispensable when it comes to identifying errors in code.

Key Insight: This provides quantitative evidence that code quality checks through various means are widely adopted by the project teams.

3.2 RQ2: Do project teams use both static and dynamic analysis tools to check the quality of their code? If so, which is preferred?

The results uncovered the preferences and the hindrances in the usage of static and dynamic tools. It also helped understand which tools were commonly used.

3.2.1 Usage of Static/Dynamic Analysis

- It was revealed that all of the 100 respondents working in various projects used static analysis. However, only 28 of the respondents were using both static and dynamic analysis tools. It was observed that none of the respondents were using dynamic analysis alone. Figure 1 shows the distribution of number of static analysis practitioners and the number of practitioners using both.

3.2.2 Experiences in usage of Static and Dynamic analysis tools

Respondents shared both positive and negative feedback on the usage of these tools. These helped us gain clarity on the reasons for the limited use of dynamic analysis tools. The study also reiterated some of the reasons leading to the difficulty in usage of static analysis tools discussed in detail by Johnson et al. [4]. We discuss below some of the experiences shared. Figure 2 summarizes the positives and negatives of static analysis and dynamic analysis.

Static Analysis: ***Positive experiences.*** Out of the 100 respondents, 64 were of the opinion that the tools are easy to configure and use. The most favored experience by all respondents was incident identification as results can easily be mapped to specific locations in the code. This is due to the fact that most static analysis tools report the line number at which the violations occur. 53 of them felt that reports are easy to comprehend. On further investigating, we found that one reason for this is that the violations reported by tools are quite descriptive, and the fixes to these issues are straight forward. And this task is further simplified due to use of toolkits [5] that provide a more understandable form of tool outputs.

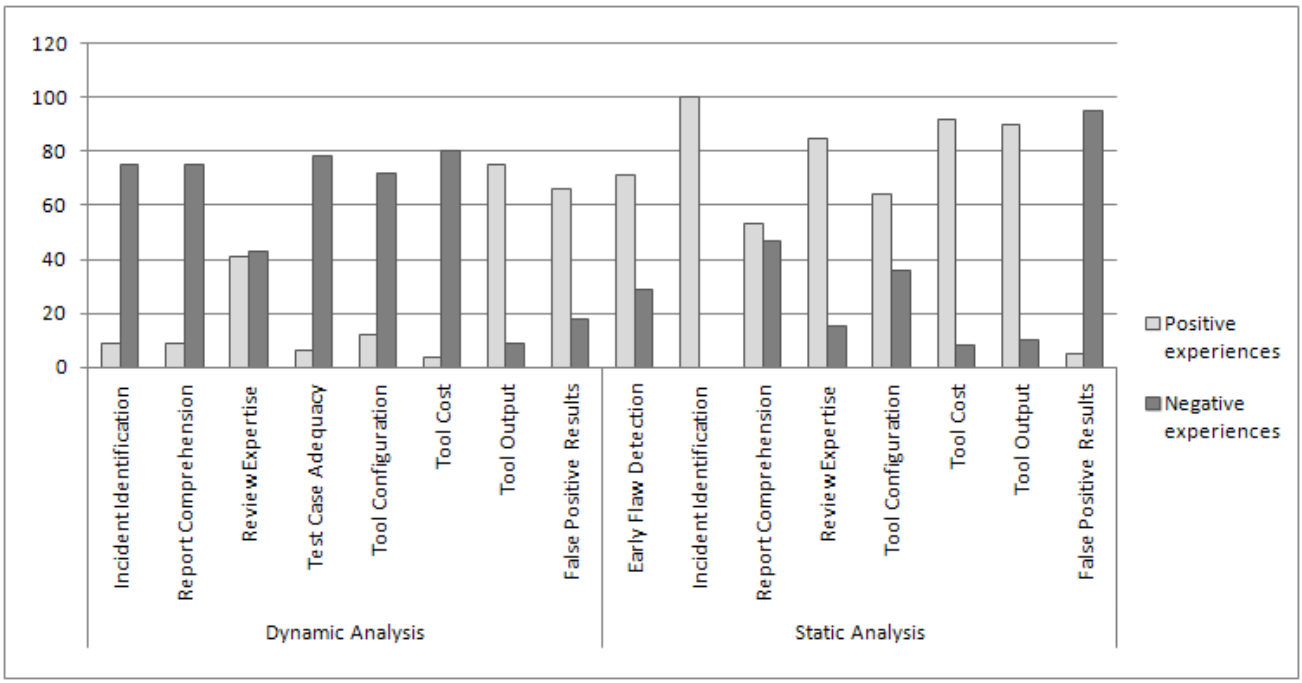

Figure 2: Positives and Negatives of Static Analysis

A significant number of them, 71, felt that use of static analysis tools helped in the preliminary identification of issues and could be done at developer's end before testing. To quote one of the comments from a respondent: *"Static analysis tools serve as a safety net for check-ins; so, code cannot be checked in without analysis of errors"*. This brings out the usefulness of the usage of static analysis tools in the course of project development.

Dynamic Analysis: ***Positive experiences.*** Some of the respondents who used dynamic analysis felt that the most positive aspect of it was that it enabled analysis of specific parts of the application, thus providing focused analysis rather than involving larger parts of the codebase. The fact that it is performed at run-time, and the actual run-time behavior of the code monitored, the accuracy of dynamic analysis is guaranteed to a large extent.

About 72 of the respondents replied that they did not have any positive experiences with dynamic analysis tools. This response is further elaborated in the following sections. Interestingly, the number of years of experience of the respondents expressing this opinion ranged between 6 to 9 years in the industry, implying that even experienced developers had difficulty in using such tools. However, almost all concurred that proactive usage of dynamic analysis tools helps to identify several hard to detect run-time issues related to memory and performance, which otherwise are undetected by static analysis.

We also questioned them on the ease of configuration and use, for which 12 of them answered in affirmation, with most of them having used DevPartner. Only 9 of the respondents also felt that some of the tools presented the outputs in an easily comprehendible manner.

Static Analysis: ***Negative experiences.*** The negative experiences are low with static analysis tools. However, the majority felt that false positives were an inherent problem with static analysis tool outputs. This could be alleviated using manual inspection, but this is a tedious process. Additionally, the participants also considered the time and resources involved in running the tools and fixing the violations as bigger constraints; and a manual review for false positives would only add to this constraint. Only about 10 which is an observably smaller number, reported that the size of the tool outputs were lengthy, perhaps suggesting that this did not pose a large hindrance to the application of static analysis in their projects. Another factor was the cost of the tools, but this was not raised by a significant number of respondents. Some of the teams felt that if the assessments could be strengthened through involvement of expert reviewers which was lacking in their current system.

Dynamic Analysis: ***Negative experiences.*** The most common response for negative experiences in usage of dynamic analysis seemed to be that the respondents lacked the expertise in usage and analysis of the tools. Out of 100 respondents 43 responded with lack of expertise in these tools, while 16 responded with lack of awareness and/or lack of exposure to such tools. This could also be due to lack of experts in topics of memory and performance in each team, and lack of guidelines/best practices or a process that ensures good coding practices. A quote from one of the respondents "we spent a lot of time in figuring out how to run the tool, only to see some results that we could not understand".

Those who had used these tools had difficulty in comprehending the results as it required in-depth analysis to infer the issues as one of them said "we struggled to understand how the results obtained could be utilized". The use of these tools was enforced only upon encountering issues related to memory or performance. In most instances, the team did not have sufficient expertise to assess the results and the learning curve required to comprehend the results posed a big hurdle in using these tools effectively and extensively. The usual problems in using tools on large applications, including difficulty in configuration, frequent crash, tool breakage, scalability issues were also mentioned by the respondents. Some also opined that time involved to execute dynamic analysis was

high. Quote from one of the respondents: *"tools create slowness and make the task of analysis time consuming"*; these occurring due to instrumented code and large amount of profiling data generated.

Iterative development process with strict adherence to timelines had resource constraints to run the tools and fix the issues reported.

Some of them reported that the high cost for procuring these tools was one of the reasons for not being used by individual team members on regular basis.

The other important and significant hindrance in the widespread usage was the inadequacy of test cases and inputs that exercised all possible program behaviors, which in fact is an intrinsic characteristic of dynamic analysis. This poses a challenge for comprehensive defect detection.

In another case, a respondent said *"we observed that the tool crashed due to memory issues at the field, but the error was not observed consistently during testing"*. Sometimes the low probability events occurring at the field might not be reproducible in the test environment.

3.2.3 Commonly Used Tools

During the course of our survey, we observed that some tools were commonly used across most projects. For Static Analysis: FxCop [33], PCLint [34], FindBugs [35], PMD [36], IntelliJIdea [37], ReSharper [38] seem to be commonly used tools. For Dynamic Analysis: DevPartner [10], AQTime [11], Yourkit .NET Profiler [13], Rational Purify/Quantify [12], JetBrains dotTrace [14], CLR Profiler [19], Yourkit Java Profiler [15], Eclipse Memory Analyzer [16], Eclipse TPTP [17] were commonly used.

Key Insight: Static Analysis is extensively used in Siemens. This reveals the importance of static analysis tools for maintaining code quality. Dynamic analysis is not as extensively used as static analysis. However, most realize the importance of using dynamic analysis proactively. Based on the above results, we can infer that experiences with static analysis tools are comparatively better than dynamic analysis tools with respect to tool configuration, usage and result analysis.

3.3 RQ3: Are the tools used only on a need basis (reactively on encountering issues) or on a periodic (proactive) basis?

We also observed if the tools were used on a voluntary basis or on a need basis, such as, on encountering a defect or on customer enforcement. Figure 3 summarizes the usage pattern of these analyses.

3.3.1 Static Analysis Usage Pattern

Almost all of the respondents used static analysis tools on a voluntary basis before code check in. Only 7 out of 100 respondents acknowledged to using static analysis on a need basis or because of customer enforcement. Most of the teams included static analysis tools as a part of their daily builds. We also observed that a significant number of them used it at the end of every iteration of their development process.

3.3.2 Dynamic Analysis Usage Pattern

The number of users of dynamic analysis was observed to be low and the observation of their usage pattern revealed that they used dynamic analysis only on a need basis. In most of the cases, they used it when issues were reported from the field, and at times, on customer enforcement or during testing.

Key Insight: In case of static analysis, most teams used it on a voluntary basis, some regularly - at the individual developer level or daily builds and some at the end of every release. Dynamic analysis was observed to be significantly less used, with the teams performing the assessment only on encountering critical errors.

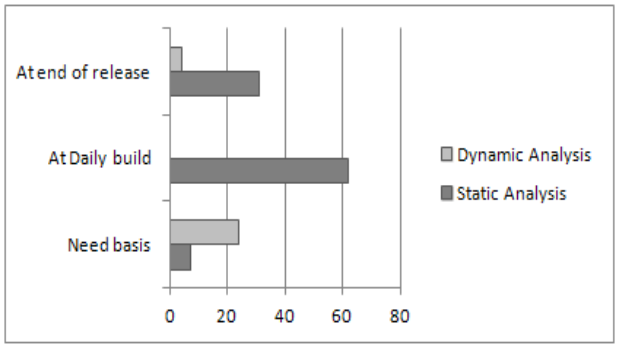

Figure 3: Usage Patterns

3.4 RQ4: How are results from these tools processed?

Our study also attempted to understand the different methods adopted by the teams to use and process the tool results.

The lengthy nature of the static analysis reports incur cost in terms of time and effort of analyzing the results which is mitigated through usage of customized toolkits [5]. These toolkits facilitate the teams to map to quality models, classify, rate and view the results in an organized manner. Most teams expressed that unlike the earlier practice of manual defect classification, the transition to automated mechanisms in processing tool outputs provided better support in code quality management. Some teams also enforced continual monitoring of violations through the use of systems like Sonarqube [20] or ConQAT [21] and aided by visualization of the results.

Unlike static analysis, the respondents felt a greater need for an expert reviewer who could help them analyze the results from dynamic analysis tools better. Given that dynamic analysis is dependent on the test suite designed for the problem space, the effectiveness of the results obtained can be only as complete as the test suite itself. As expressed by the respondents, interpreting and orchestrating results from each individual test run posed a challenge to get a holistic view on the code quality.

Key Insight: The results from static analysis, though voluminous, did not intimidate the developers as toolkits helped process and manage them efficiently. However, dynamic analysis users faced some difficulties in understanding the tool outputs. A comprehensive or detailed run of dynamic analysis was also not practically possible, given the nature of the analysis itself.

3.5 RQ5: What improvements would you like to see in Static/Dynamic Analysis tools?

The survey questioned the participants on the level of satisfaction on the usage of static and dynamic analysis tools. This was weighed on a scale of 5, with 1 being "Highly Satisfied" and 5 being "Highly Dissatisfied". Most of the respondents rated 2 for

static analysis, meaning, they were "Somewhat Satisfied". While most respondents rated 4 for dynamic analysis, meaning, "Somewhat Dissatisfied".

Static analysis tools report a large number of **false positives**, a problem, which the respondents felt was not adequately addressed by the toolkits. This was the prime improvement suggested by the respondents.

Dynamic analysis received several suggestions. **Scalability** problems in dynamic analysis tools, particularly when reviewing large applications, need to be improved. One of the respondents using dynamic analysis tools stated *"figuring out how to run the tool is one thing, analyzing the results is another, and both tasks take away a lot of our time"*. Another stated "we would like to use the dynamic analysis tools more frequently, but it is not possible as the time consumed in running and obtaining results is too high". These statements reflect the need for significantly less overheads and better turnaround times during execution leading to an enhanced user experience.

Usability improvement is an important aspect as expressed by most respondents. To quote one of them, *"Can the usability of the tool be improved so that we can use it at intermittent phases in development than only at the time of release"*. Though all tools provide user manuals that explain the tool usage, the respondents felt that the learning curve is high and hence a simplistic beginner's guide with adequate usage scenarios would be more helpful.

Result Understandability is another area in which the respondents expected changes as interpretation and inference of results was an arduous task.

Automation of some of the analysis of results, for instance, automating memory snapshot comparison, would help to bring down the time spent in analyzing the results.

Currently the tool support for dynamic analysis of **interoperable code** is not adequate, respondents felt the need to address this as there are several projects that use interoperable code in the industry.

Key Insight: False Positives in static analysis seem to be of high concern for the practitioners who picked up only this aspect as an improvement area. Dynamic analysis had several areas identified for enhancement, which included, stability and speed in execution for large applications, better user experience with respect to tool configuration and results, improved support for interoperable code and automation of the analysis to some extent.

4. EXPERT REVIEW EXPERIENCE

In this section, we share the experiences of consultants in static and dynamic analysis. Most of the development teams find the need for such experts to review their code both statically and dynamically due to the inherent problems in using the tools and interpreting the results. The aim of this section is to highlight the hindrances in performing dynamic analysis faced by an expert and also demonstrate some of the motivational factors and challenges faced by the teams to take up code assessments proactively.

4.1 Expert Experience in Static analysis assessments

About 40 projects that were assessed by the consultants using static analysis were studied. Static Analysis Assessments involved usage of static analysis tools and custom toolkits [5] that aided in visualization of the results. The lengthy tool outputs are processed by the toolkit, which enables classification of the violations based on the quality model. The tedious task of rating of violations was automated by using an expert based ratings file provided by the consultants. Quick reviews were also performed by the consultants to bring out the systemic issues in a short duration. Experiences in installing and running static analysis tools have been well documented by the consultants for quick reference to other fellow consultants, thus reducing the learning curve. Consultants were of the opinion that usage of toolkits for post processing of the static analysis tool results facilitated easier analysis and provided a saving on time and resources. Lessons learnt from each review were also well documented that included the code context information possibly leading to false positives. The general opinion was that the systematic approach ensured maximum benefit from static analysis.

4.2 Expert Experience in Dynamic analysis assessments

About 10 consultants were interviewed. Some of the projects reviewed by the consultants were studied, with respect to the tool usage, tool findings, result scrutiny by the team, and the challenges faced by the consultants in the course of the assessment. Two of the code assessments that were examined are discussed here briefly. The nature of the two sample projects differed, and so did the nature of challenges.

Assessment 1: Dynamic Analysis of New Development Project. The case study discussed here is that of a project from the Infrastructure sector. The source code is in C#. The purpose of the review was to uncover any possible performance and memory related issues for their application. Their main intent was to ensure that their application did not have any major issues ahead of their first release. For this, they decided to take the support from the consultants as they were a relatively new team and lacked the expertise in usage of dynamic analysis. During our interaction with the consultant, several interesting insights were obtained.

- The very first challenge, as expressed by the consultant, arose due to the difference in geographical locations of the team and the consultant. The project being hardware dependent, portability of the software was not possible. A hardware simulation at the consultant location failed. Hence the application had to be run remotely. Since the application had to be profiled at run-time, availability of the remote machine was a challenge, which got compounded by network issues.
- Tool installation posed another challenge, with one of the tools (AQTime [11]) taking too long to install. Also, memory profiling caused frequent tool crashes.
- The team was relatively new and the average experience of the team was also significantly low (less than 5 years). The test cases given by the team did not provide adequate code coverage. The consultant felt, that this was one of the biggest hindrances in effectively identifying all possible run-time issues.
- The final profiling was done using SpeedTrace [18] and Yourkit .NET Profiler [13]. The team acknowledged that the obtained results revealed problems that were not otherwise detected earlier. However, they felt the need for further investigation to reveal any potential memory leaks for which dot Trace [14] and CLR Profiler [19] were used. One of the interesting facts that were noted here was that usage of several tools unearthed different types of issues, reiterating that a single tool would not be sufficient to discover a multitude of problems in the application.

Assessment 2: Dynamic Analysis of Legacy Project. The case study discussed here is that of a project from the Healthcare sector. The source code is in C++ and the project has been developed over 13 years. The review was initiated by the team due to a customer reported issue. The issue was that image archival failed after several runs which on further examination revealed memory related problems. The team attempted to use Rational Purify and Quantify but was unable to do it successfully initially as they lacked sufficient experience. Due to constraints of time and lack of expertise in the team, they approached a consultant for a thorough run-time investigation.
Due to the nature of the project, challenges faced by the consultant were different.

- One of the main challenges was that of selection of test cases. The team did not want to spend any of its time and effort in the analysis, leaving the onus of the review wholly on the consultant. The project being legacy, did not have good automated test cases, so only manual test cases had to be used. Given the tight schedule and the lack of complete knowledge about the application, selecting and prioritizing the right set of test cases posed a challenge for the consultant that further hindered the effective usage of these tools.
- The application size was large. So instrumenting the entire code and running DevPartner [10] on it resulted in the application crashing. Instrumentation had to be done on a per file basis and the instrumented files added in stages for each run DevPartner, which was a tedious process costing time and effort amidst tight deadlines on fixes.
- In some cases, the results were seen to be very vague, particularly in dependent libraries. It was unclear whether they were truly errors in the framework or dependent libraries or just false positives. For instance, using DevPartner, a memory leak error "*Leak due to global_operator_delete"* was detected in several locations where CArray operations were involved, pointing to MFC libraries. Further clarity on this could not be obtained from the error report.
- The team concurred that the usage of both tools were beneficial. They however felt the need for training on the usage and analysis so that they could use them on a regular basis.

Study of some of these sample assessments revealed an important observation common to both types of code analyses, that the usefulness of these analyses was also dependent on the nature of the project. Projects having legacy code incur a greater amount of time and effort in modifying and understanding the code. With high risks, the resistance to make changes in legacy code is very high. In most cases, only a sub-set of the really critical and easy to fix issues reported by these tools is taken up for fixing. Another important insight from these two case studies is that a large number of tools are required to unearth different types of problems during dynamic analysis. This does not give much confidence to developers especially in case of legacy systems where it is typically difficult to find out issues and also correct them appropriately. From our interactions with the consultants and from the experiences shared by them, we can infer that the intent to use dynamic tools is as significant as static analysis tools, however lack of experience and expert guidance seem to be the deterrents leading to its reduced use.

5. DISCUSSION

The survey involving project teams and consultants has thrown light on the current trend and practices in the usage of static and dynamic analysis in Siemens. In this section, we discuss the outcome of our study from an industry perspective and draw more general conclusions.

Johnson et al. [4] discussed the reasons for developers not using static analysis tools extensively, some of which our study concurs with. However, an increased trend of usage was observed in static analysis owing to the following factors:

- Code Quality initiatives from the management to enforce quality checks on a periodic basis and institutionalizing it by identifying systemic issues and addressing them is one of the key factors leading to the successful usage of static analysis.
- Visualization of results is enabled by providing postprocessing toolkits that dramatically reduced the time and effort and improve the understandability of tool reports.
- Various teams have various quality goals. To address this, customized toolkits [5] have been provided to the teams to help classify results and rate them according to their required priorities, thus providing an –ility based representation of the results.
- Ease of use of tools have enabled practitioners to reduce time and effort spent in static analysis activities. The learning curve for novice tool users has been reduced. This can be attributed to training initiatives and mentoring by project experts.
- We have observed that static analysis tools have considerably evolved and can detect more errors through deep static analysis. This provides a wider safety net to detect a broader spectrum of errors, thereby deferring the use of dynamic analysis on a regular basis.
- Imprecision in static analysis in the form of false positives was handled through a feature in the custom toolkit that enabled the developer to add annotations for every violation reported. This served as a sort of feedback for selection of rules in the future runs of the tool.

The prevalence of static analysis in code assessments definitely presents an upward trend in usage. However the study of dynamic analysis usage does not present a similar picture. Our perspective is provided below.

- Owing to the fact that dynamic analysis tool result interpretation is a major barrier leading to its adoption, relevant training material that is simple, practical and comprehensive would definitely go a long way in enabling novice users to get started.
- Error detection features of current tools need to be enhanced to solve problems such as memory leaks. This holds particularly for large and complex applications interacting with various libraries and frameworks wherein the problem might not be within the code in the third party components, but in the way these components are used.
- The number of issues identified by dynamic analysis is significantly smaller than static analysis and varies considerably from tool to tool. It would help if the reported errors are supported by possible solutions and recommendations.
- Benefits of using dynamic analysis is more pronounced in languages with automatic memory management like C# and Java, as it helps visualise the memory usage patterns. If every developer can verify their code on a daily basis, the actual time spent in fixing issues can come down drastically. However, to enable this, cost of dynamic tools is one of the

deterring factors as the cost is higher when compared to that of static analysis tools.

- There is a critical need to reduce the tool intrusiveness. Also, since instrumentation poses an overhead, tools can explore the possibility of a combinaton of compile-time and run-time instrumentation. Some tools already use both these methods for instrumentation. More research can be done in carrying out instrumentation in a more optimized manner.
- Dynamic analysis primarily has a direct dependency on the test suite. While static analysis tools work under an abstraction of program behavior, the correctness and precision of dynamic analysis guarantees its effectiveness and hence usage of good testing techniques is fundamental in determining accurate program behavior. The survey results, interviews and also our experience suggest that the use of dynamic analysis typically arises out of need and does not form a systematic methodology. Even in the cases where the analysis is triggered, most of it seems to be with tests that typically run some slices of the program or in the worst case, the entire application in an ad-hoc way and hope that the results will throw some issues that could be inferred further. However, the implications of not having good enough tests directly determine the weakness of the analysis. It is practically impossible to analyse every possible execution. So selection of the right subset of possible program executions remains a challenge. A thorough understanding of which test cases execute which parts of the program is important in choosing the right test cases.

Some of the aspects that could help in writing better test cases are elaborated below.

Improvement of the quality of the test cases. Practitioners could consider how they can make use of both black box and white box testing to effectively facilitate the analysis. Analyzing program behavior with initial tests based on functional or feature test cases could give pointers to the effectiveness of these test cases. Coverage data obtained through the analysis can be used to understand how different parts of the code have been exercised. Based on this data, test cases can then be augmented until the desired coverage is achieved. Exploratory testing [29][31] can also be used to augment the existing set of test cases in a project. The main advantage of exploratory testing is that it depends on the continuously increasing knowledge of the tester and the test cases are improved on a continual basis. Dynamic analysis tools could make use of the exploratory test cases and in turn unearth more issues. Additionally, developers could employ contracts in their development that enables automation of test case generation through extraction of test cases from the contracts [27]. The synergies of both testers and developers can thus be leveraged in test selection and prioritization, thus exploring different program executions and uncovering more issues.

Generation of all possible test inputs. Another significant factor influencing the effectiveness of dynamic analysis is generation of test inputs. There has been a lot of research in several test input generation techniques such as random testing, symbolic execution, concolic testing [28], test automation from contract-driven development [27][30]. Concolic testing has been used to alleviate the limitations of symbolic execution where the approach is to mix both concrete and symbolic execution. There are many tools that aid these different test input generation techniques in the academia, but more work needs to be done to enable a large scale adoption in the industry.

Test Design techniques. There are various test design techniques like Equivalence partitioning, Boundary value analysis, State based testing, decision table testing, use case testing etc. in the black box paradigm, and statement coverage, decision coverage, condition coverage etc in the white box paradigm [32]. Depending on the context, testers have to select the method which determines how best to explore the program behavior, in turn capturing issues effectively.

In recapitulation, the evolution of static analysis has come a long way and its usage is far more prevalent. The motivation to use dynamic analysis exists, and factors discussed above could alleviate the challenges, increase its effectiveness and encourage frequent usage in the industry. A relatively large body of work exists from the academia to address some of these challenges but practical problems in realizing them still exist.

5.1 Threats to Validity

Some of the threats to the validity of our study are discussed below.

Interviewees. We interviewed participants who were primarily either developers or architects, some of whom could additionally perform the activities of code assessments within their teams. The diverse background within this controlled group may have resulted in subjective opinions, constituting a threat to internal validity. The opinions are also influenced by the availability and/or awareness of static and dynamic analysis tools relevant to their projects. We have attempted to alleviate this threat by increasing the representativeness of this selection through inclusion of consultants, who are technical experts in code assessments, in the interviews.

Assumption of static analysis prevalence. The research study was conducted at a stage when the usage of static analysis was more prevalent and so the research questions posed might have influenced the negativity in dynamic analysis usage during code assessments. The generalizability of our study is biased by the environment in which the study is conducted, where successful usage of static analysis tools was advocated and enforced at regular intervals. It might be difficult to generalize the results in a setting where static analysis tool usage itself is not extensive [4].

Nature of projects. The scope of our study is limited to some of the current projects in specific domains in Siemens. This may not represent all of the different types of projects developed in the software industry. The type of projects also influence the type of static and dynamic analysis techniques used to assess code quality. We have tried to mitigate this by considering varied projects across the different domains in Siemens that involved a major set of the popular tools used for code analysis. But we cannot generalize beyond the scope of the subjects and applications studied.

6. RELATED WORK

Cornelisson et al. [25] have discussed the usage of dynamic analysis in program comprehension. The benefits and the limitations have been documented briefly. The paper mainly focuses on providing a detailed overview of the research body of work by conducting a literature survey, as a result, providing insights into future research directions. Our work relates to this as it focuses more on the advantages and limitations and discusses them at length.

Improving the effectiveness of dynamic analysis through execution hijacking has been presented by Peter Tsanknov et al. [2]. The concept of execution hijacking is to force executions of

program behavior along unexplored paths by flipping the outcome of predicates at runtime. This would help dynamic analysis to observe new program behavior without having to generate more inputs. Although hijacked executions are infeasible, the authors concur with the opinion that if suitably performed, it can indeed help dynamic analysis techniques.

The role and importance of dynamic analysis is discussed in [3]. The author discusses future directions in dynamic analysis such as to effectively exploit run-time behavior to optimize programs and research on the application of dynamic analysis in improving software tools that assist programmers in understanding and maintenance of software. He also discusses mechanisms that can aid in efficient profiling methods so as to make dynamic analysis more successful.

Johnson et al. [4] investigate why static analysis is not widely used in the industry. Through their survey and participant interaction they discuss several reasons as to why developers underuse these and also implications for current and future static analysis tools. They also provide several potential improvements that can be made to improve the usage of static analysis tools thereby leading to a higher usage and thus improve code quality and maintenance

The synergy and duality of static and dynamic analysis has been explored by Ernst [1]. He presents the differences in the two types of analysis and also how the strengths and weakness of both can be exploited so that they can interact in ways that are more efficient and provide ways to augment each other, inspire analogous analyses and also create hybrid analyses that would combine both. He also points out that the two approaches are not as different as they might appear but are rather duals in exploring program behavior and hence use of both can create a continuum between the two.

Aggarwal et al. [22] have taken note of the complementary nature of static and dynamic analyses and discussed a methodology that combines both. Their tool focuses on security issues, buffer overflows in particular. The methodology involves identifying unsafe library functions and marking them for dynamic analysis. This reduces the coverage area and further reduces the number of test cases required to detect issues. Zhang et al. [23] also propose a framework that integrates the two analyses and a tool based on this, to discover security vulnerabilities. Elberzhager et al. [24] conducted a survey of the existing approaches of integrating static and dynamic analyses and concluded that an increased trend in this integrated approach was observed.

Thus far, no study has been undertaken to assess the reason for the reduced usage of dynamic analysis when compared to static analysis in the industry. Most of the existing papers discuss the features and techniques for performing dynamic analysis. None address the challenges in detail from an industry perspective.

7. CONCLUSIONS AND FUTURE WORK

This paper discusses the findings from the study conducted to investigate the current practices in code assessments using static and dynamic analyses in the context of our organisation. The evaluation involved interviewing various project teams and review consultants to gather experiences from a developer perspective and also from an expert viewpoint. The experiences in tool usage, result analysis and feedback from the participants on the expected improvements that could enrich the code review process are discussed.

We have discussed the several reasons for the use and the underuse of static and dynamic analysis tools in this paper. The study revealed the factors leading to the successful implementation of static analysis, namely: code quality initiatives to enforce intermittent quality checks, trainings in tool usage and result analysis, use of toolkits for result visualization and annotations for false positives.

Our results show that although both tools have their merits and demerits, the underuse of dynamic analysis is significant owing to several factors that we have discussed, the primary reason attributed to the selection of the test cases. Lack of a comprehensive test suite not only impacts the correctness of the software, but also influences effective dynamic analysis, and in turn the internal code quality. The task of automating test case selection needs to be refined. Other hindrances included difficulty in tool configuration, challenges in result comprehension, scalability issues and tool intrusiveness.

The purview of this survey is Siemens specific and more insights have to be gathered from those outside of Siemens and their practices can be studied to help necessitate further inputs in improving dynamic analysis.

As future work, we intend to regularize the usage of dynamic analysis tools within the organization, with a stress on a more proactive usage, rather than a reactive one. As of now, we have rolled out a prototype along the same lines as static analysis, with expert coaching and training in the use of tools. We still need to refine the task of result understandability through customized toolkits. Issues related to scalability still remain, and is an open research topic.

Another significant aspect that we would like to draw greater attention to is a combined approach to code assessment that leverages the synergy between static and dynamic analysis to increase the effectiveness of the assessments. This could provide pointers in the direction of identifying the problematic slices of program execution. Research and techniques based on this will go a long way in the use of these analyses in a systematic way to improve code quality standards and integrate them in a seamless manner in the developer's routine.

8. ACKNOWLEDGMENTS

We would like to thank all the Siemens CT DC AA project teams and the consultants who participated in the survey and gave their invaluable feedback.

9. REFERENCES

[1] Ernst, Michael D. "Static and dynamic analysis: Synergy and duality." In *WODA 2003: ICSE Workshop on Dynamic Analysis*, pp. 24-27. 2003.

[2] Tsankov, Petar, Wei Jin, Alessandro Orso, and Saurabh Sinha. "Execution hijacking: Improving dynamic analysis by flying off course." In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pp. 200-209. IEEE, 2011.

[3] Sabatier, Paul A. "Top-down and bottom-up approaches to implementation research: a critical analysis and suggested synthesis." *Journal of public policy 6*, no. 1 (1986): 21-48.

[4] Johnson, Brittany, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. "Why don't software developers use static analysis tools to find bugs*?." In Software Engineering*

(ICSE), 2013 35th International Conference on, pp. 672-681. IEEE, 2013.

[5] Kothapalli, Chaitanya, S. G. Ganesh, Himanshu K. Singh, D. V. Radhika, T. Rajaram, K. Ravikanth, Shrinath Gupta, and Kiron Rao. "Continual monitoring of code quality." In *Proceedings of the 4th India Software Engineering Conference*, pp. 175-184. ACM, 2011.

[6] Plosch, R., Harald Gruber, A. Hentschel, Ch Korner, Gustav Pomberger, Stefan Schiffer, Matthias Saft, and S. Storck. "The EMISQ method-Expert based evaluation of internal software quality." In *Software Engineering Workshop, 2007. SEW 2007. 31st IEEE*, pp. 99-108. IEEE, 2007.

[7] ISO/IEC 9126-1:2001, Software engineering - product quality - part 1: Quality model, 2001.

[8] ISO/IEC 14598, International Standard, Standard for Information technology – Software product evaluation – Part 1: General overview.

[9] Venkatasubramanyam, Radhika D., and Snigdha Nayak. "An Overview of Technical Models for Dynamic Analysis." *Lecture Notes on Software Engineering 1*, no. 2 (2013).

[10] "DevPartner," [Online]. Available: http://www.microfocus.com/ products/productsa-z/productsa-zD.aspx

[11] "AQTime Pro," [Online]. Available: http://smartbear.com/prod-ucts/qa-tools/application-performance-profiling.

[12] "Rational Purify/Quantify," [Online]. Available: http://www-01.ibm.com/software/awdtools/purify/

[13] "Yourkit .NETProfiler," [Online]. Available: http://www.yourkit.com/overview/index.jsp.

[14] "JetBrains dotTrace," [Online]. Available: http://www.jetbrains.com/profile

[15] "Yourkit JavaProfiler," [Online]. Available: http://www.yourkit.com/overview/index.jsp.

[16] "Eclipse Memory Analyzer," [Online]. Available: http://www.eclipse.org/mat/

[17] "Eclipse TPTP," [Online]. Available: http://www.eclipse.org/tptp/

[18] "SpeedTrace," [Online]. Available: http://profiler-and-tracer.com

[19] "CLR Profiler," [Online]. Available: http://www.microsoft.com/ en-in/download/details.aspx?id=14727

[20] "Sonarqube," [Online]. Available: http://www.sonarqube.org/

[21] "ConQAT," [Online]. Available: https://conqat.cqse.eu/

[22] Aggarwal, Ashish, and Pankaj Jalote. "Integrating static and dynamic analysis for detecting vulnerabilities." *In Computer Software and Applications Conference, 2006. COMPSAC'06. 30th Annual International*, vol. 1, pp. 343-350. IEEE, 2006.

[23] Zhang, Ruoyu, Shiqiu Huang, Zhengwei Qi, and Haibin Guan. "Combining Static and Dynamic Analysis to Discover Software Vulnerabilities." *In Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pp. 175-181. IEEE, 2011.

[24] Elberzhager, Frank, Jürgen Münch, and Vi Tran Ngoc Nha. "A systematic mapping study on the combination of static and dynamic quality assurance techniques." *Information and Software Technology* 54.1 (2012): 1-15.

[25] Cornelissen, Bas, Andy Zaidman, Arie van Deursen, Leon Moonen, and Rainer Koschke. "A systematic survey of program comprehension through dynamic analysis." *Software Engineering, IEEE Transactions on* 35, no. 5 (2009): 684-702.

[27] Ball, Thomas. "The concept of dynamic analysis." *In Software Engineering—ESEC/FSE'99*, pp. 216-234. Springer Berlin Heidelberg, 1999.

[28] Leitner, Andreas, Ilinca Ciupa, Manuel Oriol, Bertrand Meyer, and Arno Fiva. "Contract driven development= test driven development-writing test cases." In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 425-434. ACM, 2007.

[29] Sen, Koushik. "Concolic testing." In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pp. 571-572. ACM, 2007.

[30] Itkonen, Juha, and Kristian Rautiainen. "Exploratory testing: a multiple case study." *In Empirical Software Engineering, 2005. 2005 International Symposium on*, pp. 10-pp. IEEE, 2005.

[31] Ciupa, Ilinca, and Andreas Leitner. "Automatic testing based on design by contract." *In Proceedings of Net. ObjectDays*, vol. 2005, pp. 545-557. 2005.

[32] Tinkham, Andy, and Cem Kaner. "Exploring exploratory testing." *In STAR East conference*, www. testingeducation. org/articles/exploring_exploratory_testing_star_east_2003_paper. pdf (accessed March 30, 2003). 2003.

[33] Copeland, Lee. *A practitioner's guide to software test design*. Artech House, 2004.

[34] "FxCop," [Online]. Available:http://msdn.microsoft.com/en-us/library/bb429476(v=vs.80).aspx

[35] " PCLint," [Online]. Available: http://www.gimpel.com/html/pcl.htm

[36] "FindBugs," [Online]. Available: http://findbugs.sourceforge.net

[37] "PMD," [Online]. Available: http://pmd.sourceforge.net/

[38] "IntelliJIdea," [Online]. Available: http://www.jetbrains.com/idea/

[39] "ReSharper," [Online]. Available: http://www.jetbrains.com/resharper/