SURVEY ON Dynamic Analysis in Software Engineering

Sanket Prabhu
Chirag Chudasama

1. Contents

Survey on Dynamic Analysis in Software Engineering

2. INTRODUCTION

2.1Dynamic Program Analysis

Dynamic program analysis is the analysis of computer software that is performed by executing programs on a real or virtual processor. For dynamic program analysis to be effective, the target program must be executed with sufficient test inputs to produce interesting behavior. Use of software testing measures such as code coverage helps ensure that an adequate slice of the program's set of possible behaviors has been observed. Care must be taken to minimize the effect that instrumentation has on the execution of the target program. Inadequate testing can lead to catastrophic failures similar to the maiden flight of the Ariane 5 rocket launcher where dynamic execution errors resulted in the destruction of the vehicle.
Unit tests, integration tests, system tests and acceptance tests use dynamic testing.

2.2Program comprehension

Program comprehension is a domain of computer science concerned with the ways software engineers maintain existing source code. The cognitive and other processes involved are identified and studied. The results are used to develop tools and training. Software maintenance tasks have five categories: adaptive maintenance, corrective maintenance, perfective maintenance, code reuse, and code leverage.

2.3Program maintenance

Program maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes. A common perception of maintenance is that it merely involves fixing defects. However, one study indicated that over 80% of maintenance effort is used for non-corrective actions. This perception is perpetuated by users submitting problem reports that in reality are functionality enhancements to the system. More recent studies put the bug-fixing proportion closer to 21%.

2.4Evolution

Software evolution is the term used in software engineering (specifically software maintenance) to refer to the process of developing software initially, then repeatedly updating it for various reasons.

3. SCOPE

The scope of this survey was to observe and report our findings from the different topics. Some of the research relied on Program comprehension and some of the papers relied on Program maintenance. There are different software engineering tasks but we chose a few tasks and read research papers dealing with the tasks. Our objective was to obtain information from the papers provided and provide a survey of what each paper related to the topic dealt with.

4. Survey Process

For this survey, we have taken up the following software engineering techniques. Initial survey just concentrated on only a single topic – Dynamic Analysis in Software Engineering. The scope was widened to include other topics that were of our interest.
The topics that were chosen are as follows

1) Program Maintenance
2) Comprehension
3) Evolution

We read the introduction to the papers, get information about the method or tool the authors of the papers relied on for their research, check the hypotheses or research questions the authors attempted to answer, analyze the final results for the research and infer about the methods used.

Under analysis, we checked the final results used, how the results were judged and checked for any trends or if a particular method was similar to another method.

Each summary of the paper consisted of the important details of the paper including the method used, analysis of methods and the results obtained.

Some papers used a combination of program comprehension and Maintenance in the form of tools or new algorithms.

The total number of papers used for the survey came up to 11 papers.

TOPIC	NUMBER OF PAPERS
Program Maintenance	3
Comprehension	4
Evolution	4

All of the papers chosen for the survey were published between 2010 and 2015.

5. Initial Observation

Before we get started with the information we observed from each of the topics, we initially read the all the papers and we observed that all the topics we have chosen for this survey.

The details about the survey for each individual topic considered is listed down below. We have analyzed each topic separately and have provided details about what we observed from the papers related to the topics.

1. Program Maintenance
2. Comprehension
3. Evolution

6. Topics

6.1 Program Comprehension

Program comprehension is a vital software engineering and maintenance activity. It is necessary to facilitate reuse, inspection, maintenance, reverse engineering, reengineering, migration, and extension of existing software systems.
We went through many papers about program comprehension. The paper, Comprehensive Multiplatform Dynamic Program analysis for Java and Android, tested dynamic program analysis for Java and Android using ShadowVM framework. Software behavioral models have proven useful for design, validation, verification, and maintenance. However, existing approaches for deriving such models sometimes overgeneralize what behavior is legal. The authors Ivo Krka, Yuriy Brun, Daniel Popescu, Joshua Garcia and Nenad Medvidovic outline a novel approach in paper, Using Dynamic execution traces and program invariants to enhance behavioral model inference that utilizes inferred likely program invariants and method invocation sequences to obtain an object-level model that describes legal execution sequences in Using Dynamic execution traces and program invariants to enhance behavioral model inference. The key insight is using program invariants to identify similar states in the sequences that improves certain aspects of the state-of-the-art FSA-inference techniques. The scalability of the approach can be sensitive to the complexity and number of objects, which can be a shortcoming compared to the lightweight kTail-based algorithms which does the similar job.
Dynamic program analysis, such as profiling, tracing and bug-finding tools, are essential for software engineering and implementing dynamic analysis for managed languages such as Java is unduly difficult and error-prone, because the runtime environments provide only complex low-level mechanisms. Currently, programmers writing custom tools must expend great effort in tool development and maintenance, while still suffering substantial limitations such as incomplete code coverage or lack of portability. The authors Yudi Zheng, Stephen Kell, Lubomir Bulej, Haiyang Sun and Walter Binder proposed all in one dynamic program analysis framework ShadowVM which uses a combination of techniques to satisfy these requirements. It lets developers retain Java as the primary development language. With this framework, an analysis written for Java applications also supports android applications.

Understanding the behavioral aspects of a software system is an important activity in many software engineering activities including program comprehension and reverse engineering. The behavior of software is typically represented in the form of execution traces. In An approach for detecting execution phases of a system for the purpose of program comprehension, the authors Heidar Pirzadeh, Akanksha Agarwal, Abdelwahab Hamou-Lhadj proposed a novel algorithm that aims to simplify the analysis of a large trace by detecting the execution phases that compose it. The algorithm processes a trace generated from running the program under study and divides it into phases that can be later used by software engineers to understand where and why a particular computation appears. The authors concluded novel technique for simplifying the analysis of execution traces by devising an algorithm that can divide a trace content into various fragments that correspond to the execution phases of a program that implement specific tasks such as initializing variables, performing a particular computation, etc. The proposed approach is an online phase detection technique that detects the phases and the locations of phase transitions while the trace is being generated. This is contrasted with an offline approach, where the entire trace is first collected before applying the algorithm. Online processing of traces is usually more desirable than an offline approach since the users can see the results early and may need to make decisions based on this early feedback without having to wait until the entire trace is generated.
PROGRAM comprehension has become an increasingly important aspect of the software development process. As software systems grow larger and their development becomes more expensive, they are constantly modified, rather than built from scratch, which means that a great deal of effort is spent on performing maintenance activities.
Typical tasks can be designed to aim at gaining an understanding of a representative subject system, and measured how a control group and an experimental group performed these tasks in terms of time spent and solution correctness.
It is for this reason that the development of techniques and tools that support the comprehension process can make a significant contribution to the overall efficiency of software development.
An important advantage of dynamic analysis is its precision, as it captures the system's actual behavior. The purpose of paper, A Controlled Experiment for Program Comprehension through Trace Visualization, is a first quantification of the usefulness of trace visualization for program comprehension. Furthermore, to gain a deeper understanding of the nature of its added value, author Bas Cornelissen, Andy Zaidman and Arie van Deursen investigated which types of tasks benefit most from trace visualization and from EXTRAVIS. To fulfill these goals, design and execute a controlled experiment in which we measure how the tool affects 1) the time that is needed for typical comprehension tasks, and 2) the correctness of the solutions given during those tasks.
By reading this paper we can illustrate EXTRAVIS' usefulness for program comprehension. With respect to time, the added value of EXTRAVIS was found to be statistically significant.

6.2 Evolution

Software is process of developing software and repeatedly updating it for various reasons. Approach described in paper, modeling the evolution of development topics using dynamic topic models allows to improve performance and scalability of the execution trace splitting. As the development of a software project progresses, its complexity grows accordingly, making it difficult to understand and maintain. None of the existing techniques can capture both strength and content evolution. In this paper the authors Jiajun Hu, Xiaobing Sunz, David Loy, Bin Liz used Dynamic Topic Models (DTM) to analyze commit messages within a project's lifetime to capture both strength and content evolution

simultaneously by conducting a case study on commit messages of two well-known open source software systems, jEdit and PostgreSQL.
In some cases, developers may have the request to know not only the evolution of the strength of a topic but also the evolution of different aspects within it. The authors compared the results with the link model and hall model and found DTM could produce more complete and comprehensive view of software evolution, which is useful for developers and other project stakeholders to understand the changes of development topics from different aspects in a time interval.
Software processes often devote little effort to the production of end user documentation due to budget and time constraints, or leave it not up-to-date as new versions of the application are produced in the field of Web applications, due to their quick release time and the rapid evolution, end user documentation is often lacking, or it is incomplete and of poor quality. In Using dynamic analysis for generating end user documentation for web 2.0 applications, paper the authors Domenico Amalfitano, Anna Rita Fasolino and Porfirio Tramontana semi-automatic approach for user documentation generation of Web 2.0 applications is presented which exploits dynamic analysis techniques for capturing the user visible behavior of a web application and, hence, producing end user documentation compliant with known standards and guidelines for software user documentation. A suite of tools support the approach by providing facilities for collecting user session traces associated with use case scenarios offered by the Web application, for abstracting a Navigation Graph of the application, and for generating tutorials and procedure descriptions. The obtained documentation is provided in textual and hyper textual formats. Example of generating the user documentation for an existing Web application is presented in the paper.
Software documentation generation processes should rely on technologies that improve automation of the documentation process, as well as facilitating documentation maintenance. The paper presented the features of the tool that was designed to support the proposed process and showed an example of using it for re-documenting an existing application implemented using Ajax technology. The resulting documentation provides both overview and more detailed descriptions of the user functions offered by the application. This approach is scalable since the effort needed for the documentation production just depends on the complexity of the software interactions.
The reliability and security of software are affected by its constant changes. So developers use change impact analysis early to identify the potential consequences of changing a program location. Dynamic impact analysis, in particular, identifies potential impacts on concrete, typical executions. In Estimating the accuracy of dynamic change impact analysis using security analysis, paper the authors Haipeng Cai, Raul Santelices and Tianyu Xu present a novel approach based on sensitivity analysis and execution differencing to estimate, for the first time, the accuracy of dynamic impact analyses. This approach makes changes to every part of the software to identify actually impacted code and compare it with the predictions of dynamic impact analysis. This paper presented a novel approach for evaluating dynamic impact analyses which was applied to PI/EAS in particular, the best in the literature. Using this approach, authors performed the first study of the predictive accuracy of dynamic impact analyses. Results of studies indicate that PI/EAS can suffer from low precision or low recall, or both, and an even lower accuracy in most cases. The accuracy levels observed for this dynamic impact analysis are likely to be lower than what developers would expect and desire. The knowledge gained in the paper therefore gives a note of caution to developers on the usefulness of dynamic impact analysis but can also inform them on how to understand and use the corresponding impact sets.
PHP includes a number of dynamic features that, if used, make it challenging for both programmers and

tools to reason about programs. In Evolution of dynamic feature usage in PHP, paper author Mark Hills examined how usage of these features has changed over time, looking at usage trends for three categories of dynamic features across the release histories of two popular open-source PHP systems, WordPress and MediaWiki. The author has highlighted the changing trends in the use of several of these features in two systems, WordPress and MediaWiki. In these systems, uses of eval and create_function are decreasing over time. Initial results already provide some useful feedback for both developers and researchers, there are further opportunities to expand this research. One is to expand the number of systems studied. Another is to expand the range of features being explored, including features such as dynamic invocation. The results of these additional investigations should provide further information to developers and researchers, both about what features to use or avoid and what features need to be supported to provide precise program analysis algorithms supporting future releases of software systems.

6.3 Program Maintenance

Program maintenance in software engineering is the modification of a software product after delivery to correct faults, to improve performance or other attributes.
While going through some papers we found one paper which shows the maintenance of embedded software and challenges faced. In maintenance of embedded software there are many challenges related to capturing systems runtime behavior for which tracing technique is used. In Maintenance of embedded systems: Supporting Program Comprehension Using Dynamic Analysis paper authors Jones Trumper, Stefan Voigt and Jurgen Dollner proposed software based boundary tracing approach. To further reduce these overheads, instrumentation can be configured per trace, i.e., activated only for a specified group of functions without having to recompile the system. A suitable tracing technique for these systems allows for bridging the gap, i.e., facilitate debugging and program comprehension of embedded systems using existing visual analysis techniques. To reduce runtime overhead and trace size, tracing techniques for general-purpose systems may perform analysis of the recorded trace data at runtime to automatically disable tracing of specific functions. It is important to keep the amount of recorded data low: The size of the trace-data storage may be very limited, recording less data also means lower runtime overhead, and the tracing technique should not distort analysis results and should be suitable for every-day use, even for normal builds. Authors concluded that the existing tracing techniques which aims to provide better tool support rely on dedicated interfaces, are not available for all processor types, and are hardly applicable in production environments. This software based tracing technique enables developers to replace common debugging workarounds, such as log output and guesswork, by an efficient trace analysis technique
For web applications, JavaScript is the most extensively used client side programming languages. Call graph is a human understandable program representation that can be used as a basis for maintenance and further extension of the application. As there is insufficient tool support for generating call graph for JavaScript the authors Tajkia Toma and Shariful Islam proposed dynamic analysis based mechanism to construct the call graph of JavaScript of a client side web application. The paper, An Efficient Mechanism of Generating Call Graph for JavaScript using Dynamic Analysis in Web Application contributes to refine call graph to structure all the JavaScript functions and their relationships using dynamic analysis. The evaluation of mechanism proposed shows that it can identify satisfiable number of functions among the

existing total functions in the system, while the result does not include any wrong detection. Authors demonstrated that call graph can be generated for a highly dynamic language like JavaScript using information gathered in early phases of SDLC. The results shows that the functions identified by the mechanism are all included in system domain, however some functions remain unidentified. The reason behind unidentified functions is to consider only functions in namespace.
Concept identification is the task of locating and identifying concepts (e.g., domain concepts) into code region or, more generally, into artifact chunks. Concept identification is fundamental to program comprehension, software maintenance, and evolution. Different static, dynamic, and hybrid approaches for concept identification exist in the literature. Sometimes only a single execution trace is available, however, only few works attempt to automatically identify concepts in a single execution trace. The author Soumaya Medini in the paper Scalable automatic concept mining from execution traces proposed an approach built upon a dynamic-programming algorithm to split an execution trace into segments likely representing concepts. The approach improves performance and scalability with respect to currently available techniques. Concept location helps developers to perform software maintenance by identifying concepts and the location of the implementation of these concepts in the source code. Specifically, they aim at identifying parts of code that are responsible for the implementation of domain concepts. To help developers find the sequence of methods that lead to a particular bug.

7. Conclusion

Thus in this survey we went through many papers related to Dynamic Analysis in Software Engineering which were published after 2010. Since dynamic analysis was very broad topic we decided to narrow down the topic. Prof Andrian Marcus was very helpful narrowing down such broad topic. So we mostly focused on topics such as program comprehension, maintenance and evolution. Each paper had its own pros and cons. Some paper gave us tool for software comprehension which were really impressive. We got to learn lots of things doing this survey and we can consider this topic for our future project implementation.

8. References

[1] Jonas Trümper; Stefan Voigt; Jürgen Döllner " Maintenance of embedded systems: Supporting program comprehension using dynamic analysis " Software Engineering for Embedded Systems (SEES), 2012 2nd International Workshop.

[2] Tajkia Rahman Toma; Md. Shariful Islam. "An efficient mechanism of generating call graph for JavaScript using dynamic analysis in web application " Informatics, Electronics & Vision (ICIEV), 2014 International Conference, 2014.

[3] Soumaya Medini. "Scalable Automatic Concept Mining from Execution Traces" Program Comprehension (ICPC), 2011 IEEE 19th International Conference on. IEEE, 2011.

[4] Ivo Krka; Yuriy Brun; Daniel Popescu; Joshua Garcia; Nenad Medvidovic. "Using dynamic execution traces and program invariants to enhance behavioral model inference" 2010 ACM/IEEE 32nd International Conference on Software Engineering.

[5] Yudi Zheng; Stephen Kell; Lubomir Bulej; Haiyang Sun; Walter Binder "Comprehensive Multi-platform Dynamic Program Analysis for Java and Android" IEEE Software, 2015.

[6] Heidar Pirzadeh; Akanksha Agarwal; Abdelwahab Hamou-Lhadj. "An Approach for Detecting Execution Phases of a System for the Purpose of Program Comprehension" Eighth ACIS International Conference on, 2010.

[7] Bas Cornelissen; Andy Zaidman; Arie van Deursen " A Controlled Experiment for Program Comprehension through Trace Visualization " IEEE Transactions on Software Engineering,2011.

[8] Jiajun Hu; Xiaobing Sun; David Lo; Bin Li. "Modeling the evolution of development topics using Dynamic Topic Models" 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER).

[9] Domenico Amalfitano; Anna Rita Fasolino; Porfirio Tramontana. "Using dynamic analysis for generating end user documentation for Web 2.0 applications." 2011 13th IEEE International Symposium on Web Systems Evolution (WSE).

[10] Haipeng Cai; Raul Santelices; Tianyu Xu. "Estimating the Accuracy of Dynamic Change-Impact Analysis Using Sensitivity Analysis" Software Security and Reliability (SERE), 2014 Eighth International Conference.

[11] Mark Hills. "Evolution of dynamic feature usage in PHP" 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER).