Model-Based Testing

CS746 - Software Modeling and Analysis

5/8/2020

Ethan Booker

Introduction

Model-based testing is a model-driven technique for testing that focuses on creating a model that represents a set of functional aspects of a system under test to derive test cases. Model-based testing is a rather common type of testing that can be found in practically any domain such as business or critical. Essentially, the technique is heavily focused around developing a model. The model generally represents a subset of functionalities of the system that is under test. The reason why only a subset of functionalities are represented in the model is because certain aspects of the system require different types of attention for testing. Take for instance part of the system prioritizes security while the other parts of the system wants to maintain performance. Certain functionalities would need to be represented in a different model to represent and maintain certain properties. Depending on how the system was modeled, a certain amount of test cases will be derived from the model. This testing technique is able to create the same sequence of actions allowing regression testing. Interestingly enough, this technique came to be able to answer the popular old-school problem of manual testing. Manual testing is rather cumbersome as it consumes a lot of time and can be quite costly. Not to mention, manual testing promotes a higher risk of errors due to human errors. Naturally enough, in the software industry, trying to minimize costs and errors while maintaining or improving the software qualities. These focuses have led developers in attempts to automate testing as much as possible. For many models in the model-based testing, they can achieve a high level of automation. In essence, model-based testing is likely to be encountered in the testing process at some point in time due to commonly used models in the software testing.

In the following sections, each section will be based on a survey on model-based. Although model-based testing has already been introduced, there are certain aspects of model-based testing that will be elaborated on. The subsequent three sections will describe the model-based testing papers on their objective, approach that was used, significant results reported in the paper, and any limitations or continuing work reported in the paper. Following the papers section will be a short summary on what I learned about model-based testing throughout the course and reading these surveys.

An Exploration of Model Based Testing

From the section header, the paper's entire focus was on software testing, specifically on exploring model-based testing. At the introduction of the paper, software testing is explained. Near the end of the introduction, model based testing is introduced as a better form of software testing than manual software testing. Aftwards, model-based testing is further explained along with different forms of models used in model-based testing. Moreover, the paper showcases model-based testing being applied through a realistic case study of a website.

The paper was not too heavily focused on showing statistical facts but more informative information on model-based testing. Based on an assessment in the paper, on average 50-60% of development time is spent on software testing in the industry. There are two main methodologies that generate test cases known as black box and white box testing. A precise definition on model-based testing was described as "software testing in which test cases are derived in whole or in part from a model that describes (usually functional) aspects of the system under test (SUT). Since the test cases are derived from the model, model-based testing would be more likely to be considered as a form of black-box testing. It was stated in the paper that the main advantage of using model-based testing is that the time required to model the system's behavior is less than manual test case writing and execution. Interestingly enough, manual testing does not guarantee good test case generation however model-based testing is capable of generating a wide range of test cases. One important aspect of model-based testing that was mentioned in the paper was that because the functionality of the system is represented in the model, any changes made in functionality should reflect in the model as well. The constant updating on the model due to system changes helps reinforce documentation to be consistent. Conversely, changes also reinforce traceability because the test cases must match with requirements. Ultimately, model-based testing establishes a systematic approach in software testing. An aspect of model-based testing that has not yet been explained is the models. Model-based testing is based on the model, so ensuring that you have the correct model is vital otherwise there will most likely be a fault. Choosing the model is based on what aspect of the system you would like to express. Such models that the paper mentions but not limited to are: finite state machine, statecharts, UML, markov chain, and grammars. The last important finding that was discovered while reading was applying test suits. Generally, test suits can be written in some sort of spreadsheet document where they are fed to "automation testing" to give results if they pass. The automation will help provide regression testing for those selected test cases.

Despite being an exploratory paper on model-based testing, there was no conclusive evidence of any real limitations or continuation of any work. It is duly noted that model-based testing is a rather efficient and adaptable method for testing software. Model-based testing can be applied in many domains such as the web applications.

Model Based Testing for Web Applications

From a shift in desktop applications to web applications, due to the dynamic nature of the web, testing is crucial as bugs are more likely to arise. This paper is a survey on model-based testing techniques that have a reference to web based applications. The scope of the paper was driven by three research questions. The shortened version of the research questions are:
RQ1: What extent can MBT be applied to web application testing?
RQ2: Which MBT models are used in web application testing?
RQ3: These approaches address what and what are their limitations?
They used a systematic literature review in order to answer their questions. A systematic literature review can be broken down into the steps of identifying, evaluating and interpreting all available research relevant to a research question. To briefly describe their steps, they first used a search strategy using a set of keywords to search in the databases as title, abstract, and keywords and full text for a collection of resources relevant to their keywords. Next, they filtered out the resources based on inclusion and exclusion criterias. For example, they only accepted publications of 2006 - 2014, title or abstract discusses model-based testing for web applications, exclude articles that are duplicates of any already included articles, and etc. After the inclusion and exclusion, they read every single paper and selected the ones that were relevant to their research questions.

The three research question findings were discussed and rather not too shocking. For the first question about to what extent can MBT (model-based testing) be applied to web application testing, there were a few examples. Models such as simple load model, on the fly testing model, web penetration test model were used for security, CMC, UCTM were used for load management purposes, simulation workload performance analyzer was used for performance, and etc.   Which MBT models were used in the testing had quite a few familiar models. Some of the common models that were mentioned are: finite state machines, Z specification, state charts, extended versions of state machines, and activity diagrams. Of course there were others, but these were the most notably common and recognized from their examples. For the final question, I will only list a few approaches used. Take for instance, model based crawling was useful for navigation but the limitation was the performance as a test case being not satisfactory. Another example would be a model checker which was used for security but lacked the ability to bridge the gap between abstract attack trace output and penetration test on real web applications. Last example is the URMG technique being used to optimize evaluation and execution process but limited to the fact that it’s not suited to performance evaluation and loss of characteristics in characterization. It seems as though these approaches tend to focus on a software quality aspect but are limited due to sacrifice or missing some type of functionality or criteria.

Aside from the research questions, there was an analysis on the MBT approaches for web applications. There were four coverage parameters: model type, automation level, algorithm support, and testing coverage. These results disclose more revealing information. It appears that the most common model types starting from most popular: finite state machines, graphs, algorithms, and a five-way tie with activity diagram, state diagram, formal languages, trees, and extended finite state machines. What was rather more shocking is the massive automation support that MBT approach has. They noted that choosing an approach without tool support implementation is difficult. From their analysis, approximately 58% of approaches have a medium level of tool support and 39% have full tool support. MBT is not well known for relying on algorithm support as that became evident with the overwhelming 65% approaches not having to provide a stepwise implementation by algorithms. Moreover, around 57% of approaches support functional testing. As for the distribution in non-functional testing, there were greater interests in performance, reusability, security, vulnerability, and other qualities. These statistical facts do reflect the data recorded from 2006-2014.

As a followup on their work, their research was to be able to identify the gaps in the MBT approaches for future research. In other words, the limitations that were discovered are going to be the next area of focus for this group. They plan on developing tools that will help support these approaches to overcome the stated limitations.

Model-based testing is able to be applied to the web application domain, model-based testing is not limited to being a testing technique to test things. Model-based testing can become an inspiration as the basis foundation for tools being developed.

Model-Based Testing of Breaking Changes in Node.js Libraries

Model-based testing is often a technique used for testing, but some people have developed tools with the idea of model-based testing within their tools as a basis for development. Essentially, model-based testing was the idea of how they would structure their tool to be designed around rather than actually using model-based testing to solely test. The objective of the paper was to present a model-based variant of type regression testing. So although it's a regression testing tool, it is a model-based variant that incorporates aspects of model-based testing. More specifically they introduce the problem of library developers using a semantic versioning scheme to indicate if an update is backwards compatible or it has breaking changes. They do acknowledge the lack of tools that exist to help developers detect breaking changes. Throughout the paper, they did a comparison approach with their tool compared to an existing tool along with explaining how their tool works.

How does model-based testing relate to a regression testing tool? Interestingly enough, they presented a new model-based approach to type regression testing. Their tool, NOREGRETS+, an improved version of NOREGRETS, uses test suites of clients that infer to an initial API model. They then check the types of the library API by dynamically exploring it based on the information in the model. Through complicated manners of specification, they do manage to state that the model generated will include paths that were generated by the client test code. The paths will act as the model's test code to use for testing on the test suites of clients of the library to ensure there are no breaking. The paths are dynamically created using a specific grammar that is out of the scope for this paper. In being able to generate the model, they cleverly used the ES6 proxies in order to see the interactions and run the test suites between the library and clients. This only describes phase I which is to generate a model of their tool as phase II is where model-based testing usage is really used.

Where is the actual use of model-based testing?
It's in their phase II where they do the actual type regression testing. The developer will obtain an API model of one version of a library. Developers are then able to use that model to check an update in which a dynamic exploration of the updated library while testing for type regression relative to the model is run. The model will have generated test cases to test against the new updated library to check for any issues. If an issue were to occur such as an invalid type (type checking), invariant doesn't match such as the type different from library and the model, or the contravariant does not line up such as the library adds extra functionality.

They did have some discoveries with improvement of their tool compared to the older version. They asked three research questions and here the three shortened versions of their questions:
Q1: Are breaking changes detected?
Q2: Testing breaking changes in updates, is it comparatively faster and less space?
Q3: Does it find breaking changes with fewer clients?
Using 25 npm packages using their first major version they recorded quite positive turnout for their tool. They noted that except for 2 packages, the newer tool was able to detect more breaking changes, 84 - 28 respectively. Shockingly, there was an average of a 25x speed up and less space per client. Due to their idea to incorporate model-based testing with their regression testing tool, they were able to create a rather dynamic tool that is centered around model-based testing concept.

The researchers had been working on other related work at that time. They were studying the breaking changes in library updates with other tools and languages. More importantly there was another problem they raised and may tackle which is dealing with collateral evolution where they need to answer how to update clients when their libraries evolve along with determining if and where breaking changes are introduced.

What I learned

After learning material on software modeling and analysis along with reading these papers, I have naturally gained a better understanding of model-based testing. This terminology really was not introduced at the beginning of the semester but perhaps that was for good reason. All throughout the semester we worked with different types of models that are used for model-based testing and I did not realize that until this report. I've even grown more fond of model-based testing as they really are a good way to pick out certain aspects of a system to test. Moreover, they have very good tool support and are used in every domain. Not only do these models serve as testing but they are a form of documentation as well. In conclusion, model-based testing is so ubiquitous in software testing that model-based testing an integral part of testing.

References

Ramasamy, Subburaj & Singh, Vasudha. (2015). An exploration of model based testing. International Journal of Scientific and Engineering Research.

Javed, Hasan, et al. "Model Based Testing for Web Applications: A Literature Survey Presented." *Journal of Software*, vol. 11, no. 5, Jan. 2016, pp. 347–361., doi:10.17706/jsw.11.4.347-361.

Møller, Anders, and Martin Toldam Torp. "Model-Based Testing of Breaking Changes in Node.js Libraries." *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*, 12 Aug. 2019, doi:10.1145/3338906.3338940.