On Combining Model-Based Analysis and Testing

Mehrdad Saadatmand  
Xdin Stockholm AB  
Stockholm, Sweden  
mehrdad.saadatmand@xdin.com

Mikael Sjödin  
Mälardalen Real-Time Research Centre (MRTC)  
Mälardalen University, Västerås, Sweden  
mikael.sjodin@mdh.se

Abstract—Testing a computer system is a challenging task, both due to the large number of possible test cases and the limited resources allocated for testing activities. This means that only a subset of all possible test cases can be chosen to test a system, and therefore the decision on the selection of test cases becomes important. The result of static analysis of a system can be used to help with this decision; in the context of model-based development of systems, this means that the analysis performed on a system model can be used to prioritize and guide the testing efforts. Furthermore, since models allow expression of non-functional requirements (such as performance, timing and security), model-guided testing can be used to direct testing towards specific parts of the system which have large impact on such requirements. In this paper, we focus on modeling and trade-off analysis of non-functional requirements and how static analysis helps to identify problematic parts of a system and thus guide the selection of test cases to target such parts.

Index Terms—Model-based development, static analysis, model-based testing, non-functional requirements, test-case prioritization.

I. INTRODUCTION

Nowadays computer systems are present in almost every aspect of our life from TV sets, refrigerators, mobile phones to medical devices, vehicles and airplanes. While the software for each type of such systems requires a different level of quality, there are different methods that help to increase our confidence in the quality of the software product and evaluate it. Testing is the main method for assessing the quality of a software product [1]. Testing methods can be divided into two categories: static and dynamic testing (it is worth mentioning that other categorizations for testing activities can also be considered such as functional vs. non-functional testing, white-box vs. black-box testing). Static testing is basically analyzing the software to discover problems and bugs without actually executing it, while in dynamic testing, the code is executed in order to evaluate its quality and find problems in it. Examples of static methods could be code reviews and tools that analyze the source code and identify, for instance, invalid pointers and uninitialized variables, and even schedulability and response time analysis methods in the domain of real-time system. A collection of static analysis techniques is actually implemented and performed as part of programming language compilers [2].

Developing a totally fault free software is almost impossible [1], however, with the use of different static and dynamic analysis methods for testing, a product with higher quality can be delivered. On the other hand, the costs and efforts of testing is a limiting factor here especially considering the industrial need to reduce time-to-market of software products. This emphasizes the fact that with limited available time and budget for testing, testing efforts should be optimized; in the sense that they should focus more on parts of the software product with more important problems and faults to identify more bugs and perform fixes which improve the quality of the software product to a greater degree. One way to achieve this is to use the result of static analysis to focus the efforts of testing. In the simplest form, it means that if we have to choose whether to test part A or part B of a software product, and static analysis has shown that there are some potential problems in part A (but part B is fine), then we focus our testing efforts on part A. So for example, if there are security and timing requirements in a system, and security requirements are satisfied but timing requirements are not achieved, more testing should be done on timing aspects to identify problems related to this. This trade-off has traditionally been done implicitly and manually by programmers. For example, when they receive errors and warnings as a result of compiling a program, these compiler notes help them to focus their attention on specific parts of the program to fix or improve them before releasing the product to the customer.

With the increasing complexity of software systems, new and different development methods and processes are introduced and also applied in industry to ease the development process with the ultimate goal of producing better quality products and at lower costs. Model-Based Development (MBD) is a promising approach in this regard which helps to: raise the abstraction level to cope with the design complexity, perform analysis at earlier phases of development and thus identify problems earlier and before reaching the implementation phase, and also enable automatic generation of code [3]. In [4] we introduced a UML profile for modeling Non-Functional Requirements1 (NFRs) [5] to enable analysis of their trade-offs and compare different design models with respect to the satisfaction of their NFRs. In this paper, we investigate and demonstrate how the analysis that we have introduced in [4] on the model of NFRs can be used to focus testing efforts. Among others, our analysis of NFRs also helps to identify problematic parts of a system with respect

1In contrast to functional requirements, which define what the system should do, the term non-functional requirement is used for requirements which specify how a system should perform, or as suggested in [5] a non-functional requirement is an attribute of, or a constraint on, a system.”
to the satisfaction of NFRs. By consulting the result of such analysis, we can combine model-based analysis with dynamic testing to prioritize testing efforts by focusing and targeting parts with more important problems. Furthermore, since our modeling-approach explicitly deals with NFRs, we now get the novel opportunity to direct testing efforts towards parts of the systems which have large impact on the fulfillment of these NFRs. Previous, code-based analysis could only direct the attention to functional problems; but told very little about, e.g., potential performance-, timing- or security-problems.

It should also be mentioned that while in this work we discuss model-based analysis and how it can be combined with dynamic testing, we do not here enter the realm of automatic generation of test cases from models and leave it with dynamic testing, we do not here enter the realm of model-based analysis and how it can be combined e.g., potential performance-, timing- or security-problems.

The importance of these contributions becomes more emphasized when we remember that dynamic testing can usually be a more expensive verification and validation activity than static testing. The remainder of the paper is structured as follows. In Section II, we describe how this work was motivated and initiated. Section III provides a brief introduction to the NFR profile which is used to model non-functional requirements. Something, that is impossible without a formal model of those non-functional requirements.

As an example, modeling and analysis of non-functional requirements and how the result of that analysis can be used in the selection of test cases is demonstrated.

The main contributions of this paper can thus be summarized in the following points:

- We show how using model-based analysis can guide testing activities and prevent blind selection of test cases which if happens can lead to poor focusing of testing activities on minor problems and thus neglecting more serious ones which may remain undetected in the final product that is delivered to end customers.
- We show how traditional, code-based, static analysis is extended to allow test case selection that focuses on non-functional requirements. Something, that is impossible without a formal model of those non-functional requirements.
- As an example, modeling and analysis of non-functional requirements and how the result of that analysis can be used in the selection of test cases is demonstrated.

As in shown in Figure 1, analysis is done on system models and similarly test cases are generated from them. Generally in model-based testing, test cases can be generated from the system development model or a separate and dedicated test model maybe used to generate them [7]. In the latter case, two types (in terms of purpose and use) of models can be identified: one for system development which may also be used for automatic code generation, and another which is specifically created to incorporate testing information and to derive test cases from. In the former case, there is mainly one model type. In this case, usually the same development model is also enriched to incorporate necessary information for the generation of test cases. Similarly for performing analysis, different analyses may be performed on the development model or a separate model for each analysis maybe derived depending on several factors like the type of analysis and input format of an analysis tool.

As for the combination of analysis and testing and making use of the results of each, different scenarios can be identified some of which are as follows:

- Analysis results can identify some design errors before the implementation phase which can then be fixed and this way test cases defined to detect those errors can be decided to be dropped and not executed. Of course, if time and budget allows, it is always advisable to also perform testing when a fix and modification is made.
- When analysis shows that there is a problem in some part of the system, then test cases can be selected or generated to test and examine that specific part more carefully. In other words, analysis result can guide the generation and execution of test cases.
- When a change and modification is made (for example, after finding a bug by executing some test cases or through analysis), by performing impact analysis other
parts of the system which may be affected by this change can be determined. Such parts can then be tested or analyzed to ensure their correctness. This again gives a vision and focus to the testing efforts by guiding them similar to the previous item as well.

As mentioned, different types of analysis can be performed on the model, based on the interest and need, availability of required information in the model for performing a specific analysis, tool support and so on. One type of such analysis is analyzing the requirements which is one of the interests in MBAT. Addressing requirements and in particular non-functional ones (NFRs) such as timing and performance, memory usage, energy consumption, safety, to name a few, are of great importance for the embedded systems. These requirements have inter-dependencies and mutual impacts, crosscut different parts of a system and thus cannot be considered in isolation [4]. Therefore, to satisfy one, its impacts on other requirements should also be taken into account and a careful trade-off should be done to establish balance among them. Towards this goal, requirements need to be modeled along with other parts of the system and treated as first class design artifacts. Moreover, to enable analysis of requirements, necessary information and properties should also be considered for them. In the next section, we describe NFR Profile [4] which is our proposed approach for generic modeling of non-functional requirements and performing analysis on them to identify their trade-offs and enable comparison of different design models. Later in the paper, we show how the result of the model-based analysis that is enabled by our approach can be combined and used in testing to focus and guide test efforts.

III. NFR PROFILE

NFR Profile is a UML profile that we have defined to provide semantics for generic modeling of Non-Functional Requirements (NFRs) and analysis of system design models with respect to the satisfaction of them and to enable the evaluation of their trade-offs. In this section, we provide a concise description of the NFR profile that we originally introduced in [4], with some improvements.

Using a UML profile to capture NFRs versus defining a Domain-Specific Language (DSL) from scratch provides some benefits (while it also has its own drawbacks). UML is accepted as a standard modeling language already adopted and used in industry. This means that the learning curve of a UML profile will be less as the new modeling concepts in a UML profile are based on and actually specializations of the standard UML language itself. Moreover, there are already existing UML modeling tools which can be used and developers might already be familiar with. The discussion of benefits and disadvantages of UML profiles and DSLs is actually a hot and controversial topic. More detailed description of each approach and its benefits and drawbacks can be found in [8], [9].

Figure 2 shows the modeling concepts that have been defined in the NFR profile.

A brief description of the stereotypes that are defined in this profile are provided as follows. The NFR stereotype is used to capture and model an NFR, and Feature stereotype is used to represent a feature in the system which is defined to satisfy an NFR and contributes towards its satisfaction. The System stereotype is used to annotate the root node to which all the NFRs belong and is also the context of the analysis that is done on them. The relationships between different model elements are annotated using: NFRContributes, which is used to annotate the relationship between a model element that contributes directly to the satisfaction of another element, NFRImpacts which marks the impact of a model element on another and is basically a representation for the side-effects of different NFRs and Features, and NRFCooperates which is used to relate model elements that contribute together to satisfy an NFR and implement a Feature in the system. The contribution value property of NFRContributes, and the impact value property of NFRImpacts stereotype convey the magnitude of contribution and impact relationships accordingly. The NFR and Feature stereotypes have a property called satisfactionValue which contains a numerical value representing the satisfaction level of them in the system. System stereotype also has the satisfactionValue property, but in this case, it is basically a value representing the total satisfaction degree of NFRs. The priority property of NFR and Feature convey the user preferences and relative importance of each model element annotated with these stereotypes and can be set to one of the following values: 1 (very low), 2 (low), 3 (medium), 4 (high), 5 (very high).

DeviationIndicator is also another property which is defined for NFR and Feature stereotypes. This stereotype contains a value which is calculated based on the satisfaction value and priority of a model element. While the satisfaction value does not reflect user preferences, the deviation indicator value actually provides a number to the system designer by taking into account both satisfaction value and priority, indicating the magnitude of the deviation in the satisfaction of an NFR or Feature from its best case. This helps to understand which parts are good candidates to be considered first for any modification and fixes with respect to the preferences and priorities of the customer.

Several rules are also defined for the modeling concepts of the profile and the calculations that are performed on them.
The satisfaction value of each leaf node is assumed to be 1. The contribution value of a child node to its parent (set as the value of the NFRContributes link) is a positive value between 0 and 1 in a way that the sum of the contribution values of all child nodes to their common parent is, at maximum, 1. The impact value that is set on NFRImpacts links, however, should be in the range of -1 and 1. A negative value for NFRImpacts means the side effect and thus negative impact of the source model element on the target. The contribution of a node to its parent is then calculated as the multiplication of its satisfaction value by the contribution value of the NFRContributes link that connects it to the parent.

Considering these rules to calculate the satisfaction value of a node, first the total impact of other nodes on it is calculated using the following formula:

\[
I = \begin{cases} 
Min(\sum i_j, 1) & \text{if } \sum i_j > 0 \\
Max(\sum i_j, -1) & \text{if } \sum i_j < 0 
\end{cases} 
\]  

(1)

Formula 1 will result in the total impact value to be in the range of -1 and 1. The satisfaction value for a node can then be determined by calculating the contribution of child nodes and taking into account the total impact value. If \( s_k \) is the satisfaction value for each child node of a node, \( l_k \) is the value on the link that connects the child node \( k \) to its parent node (NFRContributes relationship), and \( I \) is the total impact value, the satisfaction value of the parent node is calculated using Formula 2 which will be in the range of 0 and 1:

\[
S = \begin{cases} 
Min((\sum s_k * l_k) + I, 1) & \text{if } (\sum s_k * l_k) + I > 0 \\
0 & \text{if } (\sum s_k * l_k) + I < 0 
\end{cases} 
\]  

(2)

After the calculation of the satisfaction value, the deviation indicator value is then calculated as:

\[
DeviationIndicator = Priority - Priority \times SatisfactionValue 
\]  

(3)

The UML profile is implemented using MDT Papyrus [10] which is a graphical editing tool for UML2 in Eclipse [11]. In creating the model for NFRs of a system, values for contribution value, impact value and priority properties are also set by the system designer. These values are subjectively determined and set by the system designer considering the type and purpose of the system, domain knowledge, etc. Although these values are assigned subjectively, there are methods such as sensitivity analysis [12] that help to increase the confidence in the chosen set of values. Quantification of NFRs is an interesting topic which deserves a separate study and is beyond the scope of this paper. There are also several works which specifically address this topic such as [13]–[15].

The next step is to perform analysis on the model and calculate satisfaction and deviation indicator values. To traverse the model and perform calculations based on the rules that were introduced above, a model-to-model transformation is developed using QVT Operational language (QVT-O) [16]. We use an in-place transformation approach meaning that the same model is used as input and output: the transformation reads a UML model that is annotated with the our profile, traverses the model and performs calculations, and then the results are written back to the same model. In this work, we are mainly interested in the calculated deviation indicator values to identify which parts have greater deviations and therefore focus the testing efforts on those parts.

IV. ANALYSIS AND TESTING EXAMPLE

In this section, through an example, we show how it becomes possible to use the result of model-based analysis to guide testing efforts. This way appropriate test cases can be selected when only a limited number of test cases can be executed due to resource limitations.

Figure 3 shows the application of the NFR profile in building and customizing a laptop computer product. There are several non-functional requirements that are defined for this system such as low boot-up time, increased battery life and security and to satisfy each, several features are used and applied. For example, to satisfy the security requirement, having the option to use the BIOS password checking at startup time and also finger print mechanism for authentication are considered. However, the use of such features has also impacts on other parts of the system. For example, using a password check during the boot-up process affects the requirement to have low boot-up time negatively. Similarly, adding the finger print feature will add to the energy consumption of the system and thus affects the battery life time. These impacts and dependencies are established using the NFRImpacts links which are shown in red color in the figure. The magnitude of these impacts are stated on each of these links through the impactValue property. Customer preferences are captured by setting the priority properties.

To test this system, there are several test cases that target and cover its different requirements. Assuming that the cost-efforts of each test case can be estimated as shown in Table I, the question is which test cases should be selected under a limited time and budget for testing (when it is not feasible to execute all). The cost-efforts estimation for a test case could, for example, be the time, person month, and financial cost and budget that need to be spent to perform each test.

As an example, considering the information in Table I, if we have limited resources of 85 cost-efforts to test the system, the information in the requirement model shown in Figure 3 does not help us in prioritizing and selecting appropriate test cases to discover the most important problems. However, by analyzing the NFR model and performing the NFR analysis described in the previous section, we can then prioritize test cases and select ones that help us fix the major problems in the system first.

Figure 4 shows the analyzed NFR model. The analysis is done using the rules and formulas introduced in the previous
Consulting the calculated deviation values in the model, we can identify parts of the system where more deviation has occurred and require the attention of system designers. It can be observed that the highest deviation has occurred for the battery life requirement (with deviationIndicator value of 2.0), then for the boot-up time (with deviationIndicator value of 1.6), and finally regarding the security requirement, it can be fully satisfied using the designated features and no deviation exists for it.

Assuming that in our example we have available resources of 85 cost-efforts to perform testing, we can now select and prioritize which test cases to execute by consulting the cost-efforts of each test case in Table I and the deviation indicator values. We start first from the requirement with the highest deviation indicator value which is battery life (RQ2). TC4 and TC5 can be selected for this requirement, however TC6 cannot be chosen since the available amount of resources we have is 85 while TC6 consumes 100 cost-efforts units. By selecting TC4 and TC5, 25 units of resources will still be available (85 - 10 - 50 = 25) and since there is no possible test case that can be chosen for RQ2, therefore, the next requirement with high deviation value is selected, which is RQ1 (boot-up time). Considering the cost-efforts of the test cases for this requirement, the only test case that can be executed is TC1 with 20 units of cost-efforts. With the remaining amount of available resources (85 - 10 - 50 - 20 = 5), no other test case can be selected.

This way, by consulting the result of model-based analysis...
of requirements, we managed to prioritize test cases and guide the selection of them to focus on more serious problems in the system considering the amount of available resources for performing testing activities. Without using the model-based analysis result for testing activities, the selection of test cases would have been blind and thus target a part of the system with less important or even no problems. As a consequence the product that would be shipped to the customer would contain and suffer from some major problems.

One point to mention regarding the example is that, in this work we considered test cases as atomic, meaning that it is beneficial to only execute a test case completely and partial execution of it would be of no value. Therefore, the remaining amount of available resources (which in the example was 5) was not used to select and perform any more test cases. However, if in a system partial execution of a test case can somehow be considered beneficial then this remaining amount of resource can also be used to continue the test case selection process and execute another test case.

The process of test case selection based on the analysis results can easily be automated which will be especially interesting and important for highly complex systems with large models and big number of test cases. Considering that the model-based analysis that we introduced here for NFRs is large models and big number of test cases. Considering that the model-based analysis that we introduced here for NFRs is automated, and discussed also how it enables the model-based analysis results as well.

On the other hand, while in this work we did not enter into the realm of model-based test case generation, it would be interesting as a future work to bring this topic also into the picture and investigate the feasibility of the automation of the whole process. The interplay between the analysis results and test case generation is also another interesting topic; in the sense that how the result of model-based analysis can be used to derive the generation of certain and necessary test cases out of the big number of possible test cases that can be created. This could, for example, be that when by model-based analysis a problem is identified, appropriate test cases can be generated to investigate and test more carefully that problematic aspect of the system. Similarly, if analysis results indicate a problem, system designers can fix the problem and therefore, there could be no need to generate or execute test cases for that problem anymore and thus save on testing efforts and costs.

VI. ACKNOWLEDGMENTS

This work has been supported by the MBAT European Project [6] and Xdin Stockholm AB (formerly Enea Services Stockholm AB) [18] through the ITS-EASY industrial research school [19]. The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement no 269335 (see Article II.9. of the JU Grant Agreement) and from the Swedish Governmental Agency for Innovation Systems (VINNOVA) [20].

REFERENCES


