Binding Requirements and Component Architecture by Using Model-based Test-driven Development

Dongyue Mou Daniel Ratiu {mou,ratiu}@fortiss.org fortiss GmbH

***Abstract*—Model-based testing is a well known technique to generate automatically highly qualitative tests for a given system based on a simplified testing model. Test-driven development is an established development practice in the agile development projects, which implies firstly the partial specification of a system by using tests, and after this, the development of the system. In test driven development the system implementation is continuously checked against the tests in order to assess its correctness with respect to the specification. In this paper we investigate how can these two methods be combined such that the advantages of these two approaches can be leveraged: highly qualitative test-cases used as specification of requirements and support of a continuous checking of architecture. We propose to formalize sub-sets of requirements into models that are amenable to generate tests by using automatic techniques well-known from model based testing. These tests can then be used to check the system architecture specification against the requirements in a continuous manner.**

I. Introduction

In big embedded systems projects much of the effort (60-70%) is dedicated to validation and verification. Whenever the V&V activities can be performed earlier in the process, possible defects are earlier identified and thus cheaper to correct. At the same time, a trend in the embedded system design is the use of a component architecture, which divides the whole system into isolated components and defines their syntactic interface and input/output relations [**?**]. A point in the development process where the V&V can be applied is the architectural design phase, when different functional requirements are allocated on different components of the system to be built, and the interface of different subsystems is specified. This decomposition serves as basis for distributed development, reuse and the division of work between integrators and suppliers.

Test-driven development (TDD) is a light-weighted method for partial specification of requirements that is a central practice in the agile development. It requires developers to write automated functional tests based on the requirements, prior to developing code in small, rapid iterations. In TDD tests tend to be low level tests, similar to unit tests. Tests are written by the same person who implements the code in an incremental manner and do not rely on any specific criterion for the selection of test-cases. In Figure **??**-left are presented schematically the major steps of test-driven development: based on requirements (1), test-cases are defined (2) and they are subsequently used to verify that the developed system conforms its requirements specification. Besides the testing aspect, TDD supports also the analysis and design since it implicitly specifies which interfaces the system implementation will have [**?**]. Empirical studies [**?**], [**?**] show that test-first development increases the productivity as compared to test-last approaches. However, they also report that there is a big difference in quality of the tests written by different study subjects. The quality of test-cases used in TDD highly influences the measure in which the system-under-development (SUD) is kept synchronized with the requirements. The test-cases are traditionally manually built and maintained. Thereby, they can easily become incomplete during the development and cover the requirements only poorly. Furthermore the test-cases are describing the requirements indirectly in an extensional manner (by enumerating input/outputs). Understanding the requirements (intentionally) based only on the test-cases is difficult and thereby the tests are hard to validate. This is especially critical in the case of systems that have complex interactions among their components. Furthermore, once requirements change the tests are outdated and need to be checked again.

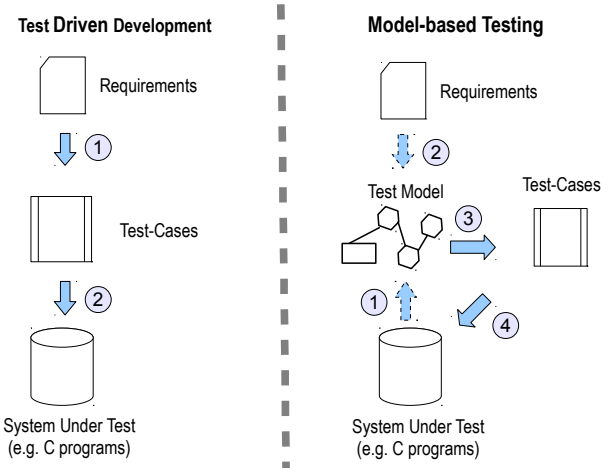

Fig. 1. Schema of model-based testing and test-driven development

Model-based testing (MBT), relies on explicit models that encode the intended behavior of a system and possibly the behavior of its environment and supports the generation of highly qualitative tests based on these models. Recent empirical studies like [**?**] report one the advantages of using model based testing like the reduced number of escaped bugs by over 50%, reduced testing duration by 25% or reduced testing costs by 17%. Figure **??**-right is presenting schematically the major steps of model based testing. Based on requirements, the system is implemented and after that, during the test phase a test model (1) is built. The test model mirrors a part

functionality of the developed system or a part of the original requirements (2). Based on this model a set of test-cases are generated (3) and they are checked against the developed system (4). Most discussions about MBT [**?**], [**?**], [**?**] focus still on the system test phase.

Therefore, a very interesting point is, whether the MBT and TDD can be combined together, so that the developer can gain all advantages from both approaches and avoid their drawbacks. In [**?**], Wieczorek et. al. described how to use MBT with TDD. Their core idea is "to use TDD for the business component development while MBT supports the inter-component service integration". Therefore, the TDD and MBT are not solid connected together but used to solve problems from different domains. The testing target is not the model but the implemented codes.

A. Proposal

In this paper, we advocate for a method to systematically combine model-based requirement engineering (MbRE), model-based testing and test-driven development into model-based test-driven development (MBTDD). As a result, all activities for requirements engineering and system architecture design are based on well defined, formal and analyzable models. Furthermore, with the help of formal refinement strategies, the requirement models and architecture models can be bound together by automatically generated test-cases. It also enables front-loading of quality assurance activities for model-based embedded software development, because these test-cases can be used continuously for checking the consistency between the requirements and architecture at the very begin of the system design phase.

II. Model-based Test-driven Development

A. Overview

As its name says, the model based test driven development is based on models for the definition of the tests. These models are also formalizing requirements and thereby the obtained tests are valid scenarios through the system. The tests are used as main links between the formalized requirements and the architecture. The conformance of architecture with requirements can be automatically and continuously tested.

In Figure **??**, we present the overview of our approach, which contains 5 basic activities.

1) Formal requirement models shall be built based on the informal requirements.
2) Test-cases are generated based on the requirement models using techniques from model based testing.
3) The refinement of system requirements into a more concrete architecture is captured explicitly through refinement models.
4) Component architecture will be built based on the requirement models and refinement models.
5) Component architecture can be verified at any time with the help of the test-cases and refinement models.

Note that, the step 4 and 5 are not carried out in sequence, but rather literately.

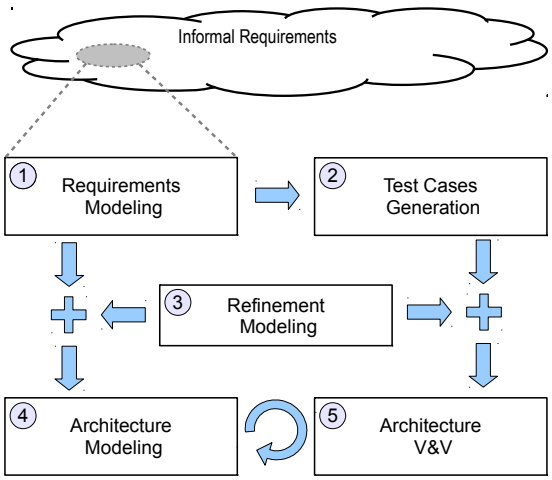

Fig. 2. An overview about the Model-based Test-driven Development

B. Modeling Requirements

Our method starts from the modeling of requirements. Here, we assume requirements to be implemented are already identified and documented. These requirements can be informal, semi-formal or formal. If a requirement is not formal, a model which represents the requirement must be built at first. The type of the model is not explicitly constrained, but only limited by the capability of used tools, such as state machine, message sequence diagram, simulink model etc. The requirement models shall be built as simple as possible and only focus on the core functionality described by the original requirement. The implementation details and design decisions shall not be involved in this step and thereby the model should be easily validated by experts (e.g. with the help of simulation). Thereby, the requirements engineers can gain confidence on the quality of the requirement models and ensure the correctness of test-cases generated in the next step.

C. Generation of Test-suite

In this step, the formal requirements will be used as the testing model. Besides the construction of the requirement models, the criterion for the generation of tests shall also given – e.g. for a state machine model, the test criteria may be transition coverage, so that the generator will try to find a test-suite that triggers all transitions in the model.

For each formalized requirement, a test-suite will be generated by the model-based test-cases generation method. The test-suite contains a set of test-cases and each test-case consists of a sequence of input data and expected output data. The test-suite offers an extensional view over the requirement while the test model is an intentional description.

D. Refinement Modeling

Since the requirement models focus on the abstract functionalities described on the requirement level, they shall provide very limited information about the concrete architecture. Therefore, in this step, the strategy for an implementation of requirement models shall be defined. The strategy may be

modeled as a transformation procedure. The typical transformations can be:

- The abstract data type used in the requirement models shall be concretized.
- The inputs and outputs in the requirement models shall be mapped onto the architecture level.
- The logical behavior must also be correspondents to certain components on the architecture.

E. Modeling Architecture and Verification

Based on all artifacts provided by the three steps above, the modeling of architecture can now be carried out. The system designer can implement the requirements into a component architecture. The component architecture shall follow the rules defined in the refinement model and implement the requirement one by one. Each time a requirement model is considered as implemented, the corresponding test-cases can be executed on the architecture with the help of the refinement model. In [**?**], for example, a systematical approach of applying test-cases from the abstract requirement level onto the concrete system architecture level is described.

III. Expected Advantages and Challenges

A. Expected Advantages

The main expected advantage of this development method is due to the unification of test-driven development and model based testing. More specifically, we envision the following aspects:

- *Straighten the requirements:* Through early modeling, the requirements are improved and put in a more precise form.
- *Simplified validation of the tests:* Since the testing models are directly derived from the requirements, they follow the requirements in a closer manner and thereby can be easier to validate than the manually written tests.
- *Improved cooperation between RE, modeling and testing activities:* The requirement/test model is directly done in the requirement phase.
- *Enable model-based agile development:* Enable agile development with rapid iterations in a model-based manner. One piece of requirements is formalized, tests are generated, the requirements are implemented and the generated tests are used to check the consistency.

B. Challenges

a) Trading-off the light-weighted character of test-driven-development: The TDD is traditionally seen as light-weight technique to provide partial specification for the system that is built. In our approach, we propose to firstly define models that can be subsequently used as basis for generating tests. Building such models implies more effort that writing simple test-cases and thereby the light-weighted character of TDD is partially lost. An important investigation direction in the future is how to keep the balance between the heavy weighted models and the agility of test-driven development.

b) Front loading model based testing: Model based testing is traditionally performed in the testing phase of the system. Our approach uses the model based testing already in the system specification phase when the system under test does not exist yet. Therefore, there might be a big conceptual gap between the models built in the specification phase and the actual implementation of the system. This gap is smaller at higher levels of granularity, we therefore envision that test-driven model based development can be used to link architecture parts with their corresponding requirements.

c) Integrated tool support: In order to operationalize our approach we need a deep integration between requirements, their formalization, the architecture description and verification through tests. Using different tools for each of these artifacts – like for example the requirements are managed in MS Word/Excel documents, their modeling is done in MathLab/Simulink models, the architecture in MathLab/Simulink again – would introduce unwanted accidental complexity.

To investigate the interplay between requirements modeling, test-cases generation and links to architecture, we are currently building a model based framework called AF3[1]. In AF3, each requirement can be directly assigned with a formal model and that can be directly linked to the corresponding architectural part that implements the requirement.

[1] http://af3.fortiss.org