Automated Model in the Loop for Embedded Systems Testing

SAMAA A. ABDEL SAMIE
Department of Computer Systems, Faculty of Computer and Information Sciences
Ain Shams University
Abbāseyya Square, Khalīfa Al-Maamoun St., Cairo 11566
EGYPT
samaa_awad@yahoo.com

Abstract: - At present there is a new trend in the embedded systems industry towards model-driven engineering. Software components are no longer handwritten in C or Assembler code but modelled with MATLAB/Simulink™, Statemate, or similar tools. However, quality assurance of model-driven engineering, especially testing, is still poorly supported. Many development projects require creation of expensive proprietary testing solutions.

Due to the complex context of embedded systems, defects can cause life threatening situations. Delays can create huge costs, and insufficient productivity can impact the entire industry. The rapid evolution of software engineering technologies will be a key factor in the successful future development of even more complex embedded systems.

Nowadays, Model Driven Engineering has become a promising approach. Instead of directly coding the software using programming languages, developers model software systems using expressive, graphical notations, which provide a higher abstraction level than programming languages. Model Based testing as well will help identify problems early thus reduce rework cost. Applying tests based on the designed models not only enable early detection of defects, but also continuous quality assurance. Testing can start in the first iteration of the development process.

In this paper, we present an approach to functional black-box testing based on the system’s model. It provides a test model that is executed on the system model itself. The idea is to validate the software model itself, as it is the base that will be used for code generation later on. It is contrasted to the current approaches which extract the test model from the software model, thus defects in the software model during development will continue to appear in the test cases, and later in the code because the model itself was not validated. The proposed approach is called Automated Model-in-the-Loop for Embedded Systems Testing (AMiLEST).

Key-Words: - Embedded Systems, Model Driven Engineering, Model Driven Testing, Model in the Loop

1 Model Driven Engineering

The main principle of Model Driven Engineering (MDE) is that “Everything is a Model” [1]. Modeling, in the broadest sense, is the cost-effective use of something in place of something else. It is an abstraction of reality in a simpler, cheaper, and safer way. It helps avoid the complexity and irreversibility of reality. Although models are used in various engineering domains, they only recently played a role in the development process of embedded systems.

MDE aims to improve productivity and to facilitate the system’s development by creating, maintaining, and manipulating models. A system is developed by refining models, starting from higher and moving to lower levels of abstraction until code is generated. A MDE approach can imply several different developers, modeling languages and tools that model the same system.
1.1 Model Driven Development

Software development is shifting from manual programming, to MDE. One of the most important challenges is to manage the increasing complexity of embedded software development, while maintaining the product’s quality, reducing time to market, and reducing development cost [2].

Model Driven Development (MDD), as any other development activity, should controlled by a development process. Ensuring product’s quality is crucial in such development, particularly in safety-related systems. Applying validation and verification techniques before code generation can ensure software quality.

To improve the automation of the design from the initial requirement specification until the final system, researchers look for modeling methods to specify, analyse, and verify embedded systems in a fast and accurate way.

Due to the increasing complexity of embedded software development over time, organizations have invested in modeling, starting by early approaches till the establishment of Unified Modeling Language (UML).

The promise of model based design approach is to develop design models and subject them to analysis, simulation, and validation prior to implementation. Performing analysis early in the development life cycle allows early detection of problems and thus fixing them at a lower cost [2].

1.2 Model Driven Validation

Embedded systems are present, and enormous in number. Their growing complexity calls for not only new design approaches, but also new powerful, automated and enhanced testing methods.

Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing them during the requirements and design phase [3].

Finding and fixing bugs is overall the most expensive activity in software development. Embedded software tends to perform more kinds of testing than other forms. The following figure shows that the cost of fixing bugs rises exponentially with each phase of the software development life cycle.

Thus reduce rework cost. However, models are usually generated manually thus require time and effort.

Specifically for embedded systems, late defect correction costs much more than that of other software types due to the close hardware interaction. The clear focus therefore is to detect defects as early as possible. Best would be in the phase where they are inserted.

The two major cost drivers in embedded software development are requirements and test [4].

- Requirements

We often develop the wrong things due to lack of reviewing and analyzing requirements or missing and vague requirements. 40% of all software defects in
embedded systems result from insufficient requirements and analysis activities.

The typical effort allocated to requirements engineering is 3 to 7% of total project cost. Doubling this effort has the potential to reduce life cycle cost by 20 to 40%. The cost reduction mostly is due to earlier analysis and defects removal during specification and requirements verification.

- **Test**
  30 to 40% of the project’s effort is allocated to testing. That’s almost half the cost of the product. A major advantage would be, how to minimize test effort? The answer would be first, to detect defects close to the phase where they are created, and second, by removing redundancy.

Model-Based Testing is the technology in which test cases can be executed on a model. After their execution, test cases can also be automatically evaluated and documented. In all cases, the user must work with a model and follow a systematic working method in testing [5].

As the integration level changes during development (model, component, software, ECU), individual test execution environment also changes.

Four levels are explained here [6]:

- **Model in the loop**: the developed model is subject to a test. In other words, inputs are changed, and outputs are monitored. Feedback is always needed to maintain model’s compatibility with the original requirements. Many modeling environments offer the capability of executing tests on this level.

- **Software in the loop**: at this level, code is generated from the model. A partial manual implementation can be also added. Testing the code is then executed on a simulation environment.

- **Processor in the loop**: the code will be then added to the target processor, and testing is executed then.

- **Hardware in the loop**: now the physical environment is present, and the transition from PIL integration is smooth.

The traditional V-model is a commonly used approach as a software development life cycle. Its acceptance is due to its simplicity and ease of application. The following figure shows the different tests across the development v-cycle model

![Fig.2 - MiL, SiL, PiL, HiL across V-Cycle](image)

Two major points represent a challenge in MBV:

- The test sequence used on a specific level should be reusable.
- It must be possible to automate all test sequences, in order to achieve reproducible results.

The MBV approach can be described as a form of black box testing, where structured models are used to automatically generate test cases, thus automate the test cases design process.

2 **Model In The Loop**
The main advantage of model based validation is increased quality and productivity by shifting the testing activities to an early stage of the software development life cycle and generating test cases that are independent of any implementation.

In MiL, the developed model is subject to a test. In other words, inputs are changed, and outputs are monitored. Feedback is always needed to maintain model’s compatibility with the original requirements [7].

Very few model based validation techniques use the developed models as a testing target [8]. While all other techniques focus on testing the implementation through the test cases generated from the models. The major problem in generating test cases from the models is that there’s no guarantee that the developed model itself is reliable, or correctly represents the system’s requirements. Thus extracting test cases from it and then executing it either on the auto-generated code or the hardware platform does not help detect development issues that will later on appear in the software.

The following section describes a proposed approach that automates MiL testing in a way that the developed model itself is the target.

3 Proposed Approach
Our proposed AMiLEST approach consists of the following steps:

1- **Model Based Development**: in this phase, a model is developed based on the requirements. A general overview of the system can be modelled using class diagrams, and then the detailed behavior can be represented by means of state charts, with states and transitions between them.

2- **Model based validation**: based on the model obtained from step 1, a test model is created as a first step in MBV approach. In our approach, the test model is a series of state charts, each representing one test scenario to be executed on the original model. Each test scenario consists of a series of inputs, and expected outputs.

During step 2, the proposed tool chain will include:

- Automatic test cases execution, where the developed test state charts are automatically executed on the system’s model.
- Automatic test cases evaluation, where the execution results are documented and analyzed.

A tool chain will allow the model to be tested in its original form, to test its functional behaviour and conformance to the software requirements. After the model is tested and stabilized, conversion process can then start to auto-generate the code to be used in SIL.

The proposed flow is shown in the following figure. First, the requirements/specifications are interpreted by both, developers and testers. The developers will construct the system model, and the testers will construct the test model. After that and as a first step, the test model will be executed against the developed system’s model and the results will be analysed. During this stage, continuous feedback is needed, and ongoing system model and test model corrections are made, to ensure compatibility between the requirements and the model.

After that, the code can be generated from the model, which will then has a high degree of reliability. The same test scripts generated by the testers can now be executed on the generated code, either on a PC-simulated mode (SIL) or on the target processor (PIL).
Note that it’s preferable to keep the development and testing tasks separate, allowing for different interpretation of the requirements. This normally helps discovering specifications errors and ambiguity.

4 Summary & Conclusion

Embedded systems development is shifting towards Model Driven Engineering that is embedded systems requirements are represented in graphical models, using statecharts and activity diagrams. Embedded software is then auto-generated from these models. Thus the validity of the models is a very important issue.

Since development platforms offer a code generation feature, most current approaches extract test cases from the developed models, then execute them on the code in later stages. This is not a very reliable technique, since system’s model is not validated, thus defects continue to appear in the test cases, and also in the code.

We propose an Automated Model-in-the-Loop for Embedded Software Testing (AMiLEST) approach, which focuses on using the developed model as the validation target. This way, defects are detected at a very early stage, and generated code is guaranteed to be reliable and with minimal error rate. This approach is a major source for cost reduction, and increased software quality.

References:

[1] Iliane Ober, “UML-based modeling and verification of real-time embedded systems”, IRIT, University of Toulouse 2009