Proceedings of the World Congress on Engineering and Computer Science 2013 Vol II
WCECS 2013, 23-25 October, 2013, San Francisco, USA

Test Driven Mobile Applications Development

Haeng Kon Kim

Abstract— **Mobile applications testing is the most important factor in its software development. Mobile applications testing is a process by which application software developed for hand held mobile devices is tested for its functionality, usability and consistency. Mobile applications either come pre-installed or can be installed from mobile software distribution platforms. Increasing complexity of the mobile applications system makes difficult to test and evaluate the quality properly. As a result, automated testing methodology is becoming popular and in turn decline of manual testing. Because of the characteristics of Mobile applications software, automated testing has difficulty performing all relevant tests and evaluation of the areas of concern. Model-Driven Testing Techniques (MDT) artefacts software engineering bases on model transformation principle. This implies increasing research on automation of the testing processes.**

In this paper, we present an approach to derive tests from the model of mobile applications system. The analysis of methodologies used for mobile applications as well as for standard system development demands creation of a bridge between them. We also will discuss the reliable testing processes. In particular, test development for each phase of system engineering is proposed. Input signals as continuous, discrete and real time constraints are the factors indicating object oriented or function oriented approach. Finally, Model Driven Testing ideas are mentioned so as to elaborate the full overview on test process automation for Mobile applications systems.

Index Terms—

I. INTRODUCTION

New software development methods based on models and distributed component technology is a major step towards more efficient software systems. The growing complexity of such systems increases the need for solid testing to ensure the reliability. However, testing is often not well linked with other development phases. One reason for this is that designers, developers and testers use different languages, file formats and tools, making it difficult to communicate with each other and to exchange documents [1,2]. The early integration of test development into the system development process becomes more and more important. Design mistakes and implementation faults can be detected in an early stage of the system development. This allows for reducing the overall software production time and costs significantly [3]. Mobile applications systems vary in size and complexity, and might include such divergent things. Although the use of models is beneficial to both types, there are certain trade-offs required when using model driven development on such Mobile applications systems. Small and resource-constrained systems may have such limited memory. Within Mobile applications software circles, the practices of Test-Driven Development (TDD) and Continuous Integration (CI) are either unknown or have been regarded as prohibitively difficult to use. Real-Time Operating System (RTOS) becomes prohibitive. In such cases, operating system services like threads, processes, or tasks are not available. Modeling tools that provide MDD capabilities need to take these diverse requirements into account[4].

The idea of this paper is to perform the automation of testing engineering parallel to the automation of system engineering. The innovation bases on bringing together the existing approaches with UML to use in Mobile applications and standard system. The aim is to split testing processes into main software engineering phases so as to begin to test at the early stage of the development. Further, signal types (continuous, discrete) and real time constraints at the model input side distinguish between object oriented and function oriented approaches.

II. RELATED WORKS ON SPATIAL INDICES

2.1 Mobile applications Software Testing

Testing is a method to find defect(s) in software. Especially in Mobile applications software development, there are many hardware related limitations compared to generic software development. Testing must work in a way that does not influence these limitations and must fit the following criteria:

· Real time testing: by definition, testing must be done in real time
· Non-interference of testing: testing must not interfere with elements not being tested.
· Support various kinds of connection methods: an mobile applications system must connect and runs on different kinds of network connections.

Thus, the Mobile applications software developer must have knowledge about software and hardware to best construct and execute the appropriate test(s).

Testing can be divided in a two forms: manual and that using automated tools. Using the manual method, hardware related knowledge are prerequisites and the tester's knowledge and experience is also an important factor in testing.

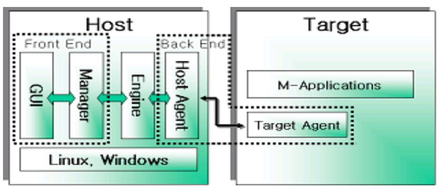

Fig. 1. Mobile applications software testing tool

Heang-Kon Kim is with School of Information Technology, Catholic University of Daegu KyungSan, Daegu, 712-702, Korea (corresponding author to provide phone: 053-850-2743; fax: 053-850-2740;
e-mail: hangkon@ cu.ac.kr).

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Using an automated tool means using source code or a design model in predefined way to create a test case and tests automatically. Today complexity of Mobile applications software makes manual based testing ineffective and requires lot of time so automated testing is taking place of it.

Figure 1 shows the general structure of an automated Mobile applications software testing tool. Source code is analyzed in the host and from that test cases are generated. The test cases are then sent to a target board using various methods and then finally executed. Results of the test case then are sent back to the host and analyzed.

Some of the typical automated Mobile applications testing tools are presented below:

· **Codescroll:** a source code based testing tool.
· **VectorCAST:** a source code based testing tool.
· **Qtronic:** an automated model based testing tool.

There are parts that can't be done using an automated testing tool. Automated test operate in a predefined way and mostly use source code to test factors internal to the system. This makes errors produced in an area not defined, or unpredicted way of error [5], hardware related parts like LCDs and switches cannot be tested or cannot be tested in an efficient way. So, it is said that the importance of the manual testing of hardware related aspects are increased.

Mobile applications software testing especially in manual testing where testing is manually intensive; experience is an important factor affecting the test. Therefore the results of the test are highly affected by the tester's experience. This leads to the assessment that there must be a way to share the experience of testers.

2.2 Mobile applications Software Quality and Evaluation

Use of Mobile applications system is rapidly increasing. This increase of Mobile applications systems creates many similar systems for the customer to choose from. This influences the quality required of Mobile applications systems, because the system can't be improved or fixed when the development is over and fatal errors can affect the product, and consequently the evaluation of the company. It's because of consumer requirements that the importance of quality has been increased [6].

Developers and testers are very increasing in serving the customer with quality products, because now quality is an influence on the sales of a mobile applications product. Quality evaluation results can view differently by the people who see it. So we need classification of elements. Quality evaluation is another important factor which effects entire development process. Since evaluation collects element form entire development process, it can be used as resource for manual testing and reducing testing time.

2.3 Mobile applications Software Testing Process

Mobile applications software testing is a disciplined process that consists of evaluating the application (including its components) behavior, performance, and robustness. One of the main criteria, although usually implicit, is to be as defect-free as possible. Expected behavior, performance, and robustness should therefore be both formally described and measurable. Verification and Validation (V&V) activities focus on both the quality of the software product and of the engineering process. These V&V activities can be sub-classified as preventative, detective, or corrective measures of quality. While testing is most often regarded as a detective measure of quality, it is closely related to corrective measures such as debugging. In practice, software developers usually find it more productive to enact testing and debugging together, usually as an interactive process. Debugging literally means removing defects.

Mobile applications software industries in airborne, train, and automotive domains is becoming more mature, and catch up with other industries such as the computer hardware industry. Model based development processes are established and exercised for real-time Mobile applications systems. Nowadays, there is a strong trend to automate the safety-critical functions, which in turn requires application of safety standards. Automation demands the application of formal methods and formal verification. The development process of Mobile applications systems considers the question which process steps in the product life cycle have to be covered [7] and supported by appropriate tools. The most important steps are:
- **capturing** of textual specification
- **modeling** of software and hardware topology and its functions considering all the constraints automatic code generation from the models automatic test code generation from the test models
- efficient software development, **testing and debugging** environment Mobile applications software engineers and test engineers have to develop and verify their software using model checking together.

The corresponding benefits cope with design failures, which are detected early in the overall process. The quality of specification models increases. These improvements result in significant cost reductions during the software development process. The Mobile applications software design methods used for years suffered from informal specifications, lack of adequate support for verification, fairly long design times. This situation has become untenable as their complexity and safety, cost and power consumption requirements put on them has scaled up. The situation has been made even more difficult by the increasing degree of integration in the semiconductor industry that has made possible to build Systems-on-Chips (SOC) with unparalleled compute power. In too many cases, errors in conception and implementation of Mobile applications controllers have caused dramatic problems especially in the area of space exploration and applications. On the other hand, the opportunities offered by technology for Mobile applications controllers are immense.

III. Mobile Applications Systems Testing with MDT(Model Driven Testing)

The development process of Mobile applications systems influences the test methodology. That is why it is very important to choose the appropriate methodology concerning both of them so as to succeed. For Object-Oriented Development (OOD) the emphasis is put on organizing the structure of a system around the fundamental objects and data, and their associated functionality. For Function-Oriented Development (FOD) the emphasis is put on organizing the structure of

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

a system around its required behaviour. Object Oriented Development on the system side implies rather standard approach to testing, however the time constraints and continuity must be preserved. Function Oriented Development (FOD) on the system side means working with a new approach to testing. The main system and test characteristics are defined by the weighting of continuous versus discrete behaviour, existing distribution of functions and communications versus basis user functionality. In this paper, we mainly focus on the appearances especially in the context of safety critical and space restricted mobile applications systems, where test must be reliable, repeatable and flexible. Figure 2 shows the overall architecture for Mobile applications software MDT(Model Driven Testing) to concern it.

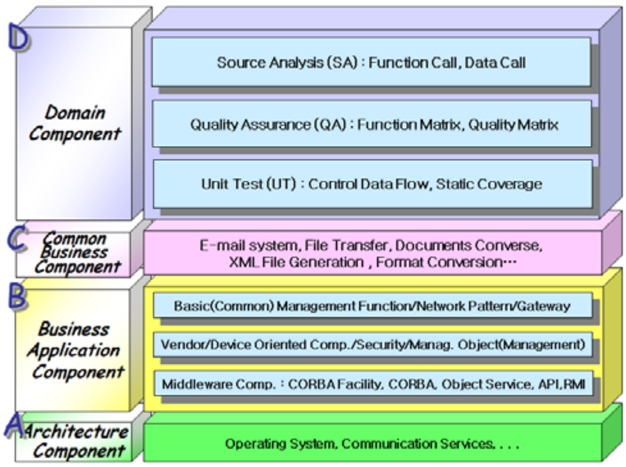

Fig. 2. Overall architecture for Mobile applications software MDT(Model Driven Testing)

3.1 Model Driven Testing for Mobile applications Software

In usual software engineering approaches, object oriented modeling with UML has become popular as in figure 3.

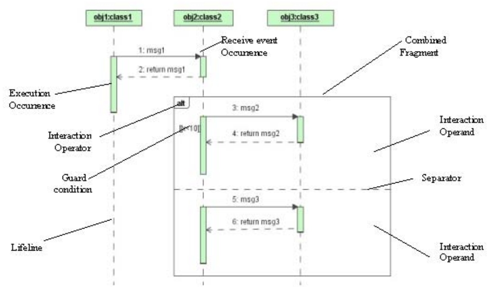

Fig. 3. UML Architecture to use for Mobile applications software testing tool

In this area, object orientation has been extremely beneficial: object oriented modeling allows structuring the application data in a way that eases maintenance and allows distributing the responsibility for certain functionality among the participating objects or components. Finally, object oriented techniques enable the use modern design pattern that improves flexibility and maintainability. In the area of Mobile applications systems, models like Simulink [8] or languages like Very High Speed Integrated Circuit Hardware Description Language (also VHSIC Hardware Description Language) shortly VHDL [9] are used to describe hardware circuits. C is used to program electronic control units (ECUs). Programmable logic controllers are programmed in some kind of assembler, using function block diagrams or Pascal like structured text. Object oriented concepts are widely considered as inefficient and unsafe. They usually imply some kind of pointer concept and dynamic organization of heap memory. Pointers may be null or point to already freed memory cells. The heap consumption may grow uncontrollably. Numerous functional requirements in an automotive electronic control unit (ECU) cannot be handled at once. Thus these aspects are handled separately in a deeply structured process, e.g. user functions, communication, software and hardware architecture as well as software distribution on the components. The realisation of a structured process handling different abstractions along the path requires the use of the most appropriate notation for each abstraction. UML provides a much broader range of means of description, with advantages especially for the analysis and design phases. In Simulink a clear defined methodology and modelling guidelines are necessary in order to realise the necessary abstractions and topology descriptions.

Dealing with hybrid signals (continuous and discrete ones) with additional real-time conditions, it is difficult to choose the methodology/technology to design a model of Mobile applications system. Furthermore, the real-time constraints on system side are still valid for the test model. That is why the decisions taken for system development influence the test. In the case of standard software these effects are not as critical because the test methodology is more independent of the system development methodology. In the case of real-time Mobile applications software one deals with the simulation based on the system model. This simulation is defined on the system side, however from certain perspective it impacts the test. For example, running simulations of the model provides information about model coverage that helps testing groups to determine which aspects of the implementations are covered by equivalent tests. Test data and vectors can be generated for use in test harnesses, either directly from the requirements or based on the design. This is particularly valuable for systems that contain large amounts of logic, where designing of test sequences is particularly difficult. It results from the critical nature of the system, which implies performing the simulation fed with the concrete examples of data, taking into account real-time conditions.

3.2 From Specification, through Models, towards Tests

Test specification can be partly derived from system specification. This implies formalisation of the system requirements. Such requirements-driven testing enables test teams to develop tests against current requirements, rather than building them in isolation. Since testing is based on conformance to requirements instead of general test statistics, this approach delivers higher quality results: test teams can validate that the system, software or product does what is required. The approach is provided by DOORS tool. The automated transfer of information appears, however manual methods to retrieve the final test requirements are used.

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

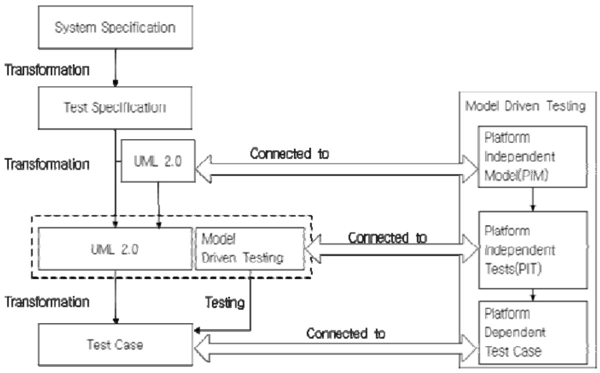

Fig. 4. Mapping Between Test specification and MDT (PIM and PSM)

MDT gives the possibility to design and implement functions of time continuous behaviour, while discrete behaviour or modeling the architecture on abstract level can be done in UML (but also in Simulink). Depending on the system engineers one has to adapt testing methodology to their requirements. One of the options is to use UML 2.0 models as completion of MDT models for Mobile applications systems. Both kinds of models should be mapped and developed parallel completing each other and taking into account system requirements as given in figure6. Model transformation between the different philosophies of object orientation in UML and functional orientation in Simulink is still a matter of research. Successful results and prototype realizations. In this paper, we promise practical solution in mid-term future. Simulink model is tested during its simulation. It can be additionally tested using Classification Tree Editor for Mobile applications Systems (CTE/ES) and Mobile testing tools. UML 2.0 models (e.g. HybridUML Profile) must be additionally tested with traditional approach. The testing methodology against UML models is the application of UML 2.0 Testing Profile (U2TP) being the Object Management Group (OMG) standard. Test artefacts retrieved from UML models and applied by UML 2.0 Testing Profile models serve as the base for further testing procedures. Moreover, the interfaces between Simulink models and UML models should be also tested, at least partly automatically. All the connections found, have to be tested. Further, at least two ways are possible – derivation of test code from the test model or derivation of test code from the system implementation code. The first one corresponds to Model Driven Testing concepts, that is why only this will be investigated. It is again a challenge, as the information must be derived not only from the UML 2.0 Testing Profile test models, but also from Simulink model and simulation. This can be done by using the formal methods. All the testing processes described till now correspond to black box testing. That is why the good candidate for test implementation code applied for real-time systems is extended version of Test and Testing Control Notation version 3 (TTCN-3). Finally, implementation code generated either manually or automatically from the models must be tested in white box testing. Although in the second case it has been proven that the code provides fewer failures than in the first case.

MDA prescribes certain model artefacts used along system development line, how those models may be prepared and their relationship. It is an approach to system development that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform. Main MDA artefacts are platform independent system models (PIMs), platform specific system models (PSMs) and system code. There is a clear distinction between PIM, PSM and system code although it depends on the context, the development process and the details of the system and target platform, where the border between PIM, PSM and system code is to be placed. Within these three abstraction levels, transformation techniques are used to translate model parts of one abstraction level into model parts on another abstraction level. These MDA abstraction levels can also be applied to test modelling as according to the philosophy of MDA, the same modelling mechanism can be reused for multiple targets. As shown in Figure 4, platform independent system design models (PIM) can be transformed into platform specific test models (PIT). While PIMs focus on describing the pure functioning of a system independently from potential platforms that may be used to realize and execute the system, the relating PITs contain the corresponding information about the test. In another transformation step, test code may be derived from the PIT. Certainly, the completeness of the code depends on the completeness of the system design model and test model.

Figure 5 shows the sequence diagram for MDT that is consists of 4 different classes to make it

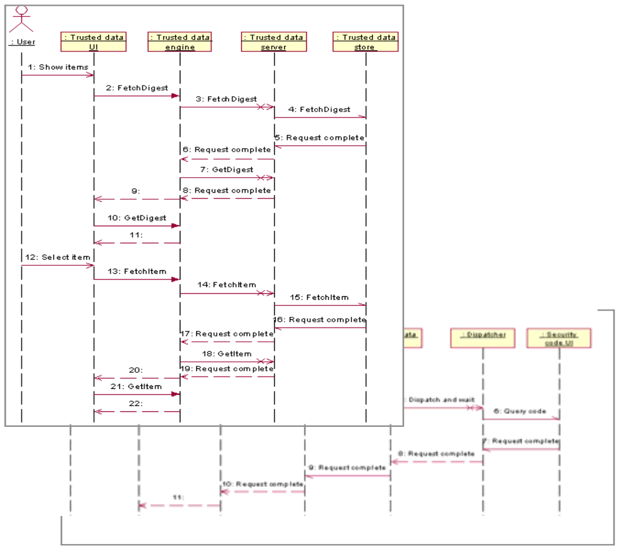

Fig. 5. Sequence diagram for MDT

The following code is Test conductor integration tests for interactions with Mobile applications hardware and model with MDT. The "_Expect" and"_Return" functions are automatically generated by MDT from the interfaces specified in header files.

```
static void testRunShouldNotDoAnythingIfItIsNotTime(void)
{
AdcModel_DoGetSample_Return(FALSE);
AdcConductor_Run();
}
static void
testRunShouldNotPassAdcResultToModelIfSampleIsNotComplete(v
oid)
{
AdcModel_DoGetSample_Return(TRUE);
AdcHardware_GetSampleComplete_Return(FALSE);
AdcConductor_Run();
```

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
}
static void
testRunShouldGetLatestSampleFromAdcAndPassItToModelAndStartNewConversionWhenItIsTime(void)
{
AdcModel_DoGetSample_Return(TRUE);
AdcHardware_GetSampleComplete_Return(TRUE);
AdcHardware_GetSample_Return(MDTU);
AdcModel_ProcessInput_Expect(MDTU);
AdcHardware_StartConversion_Expect();
AdcConductor_Run();
}
```

Figure 6 shows the Execution Environment for Mobile applications on MDT. It consist of system model and test model as development and execution

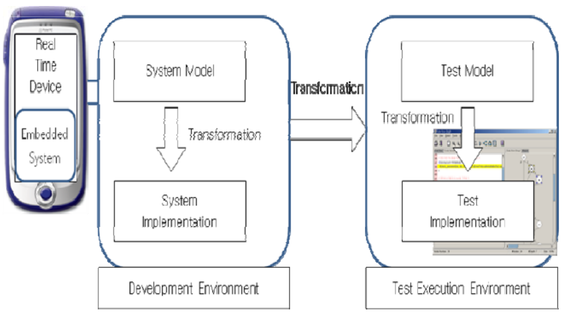

Fig. 6. Execution Environment for Mobile applications on MDT

IV. Conclusion and Future Work

In this paper, we design and implement the automation of testing development parallel to system development. The innovation in relation to standard software engineering is the analysis of this development considering the input signal type (continuous, discrete) and real time constraints. Object oriented and function oriented approaches are compared and combined. Finally, Model Driven Testing ideas are used so as to elaborate the full overview on testing automation process for Mobile applications systems.

Future work requires further investigation on test information retrieval from MDT models. It implies a lot of effort in the context of transformation, mapping rules and technical possibilities Such transformations follow the principles of MDA-based testing, which differentiates between platform-independent tests (PIT), platform-specific tests (PSTs), test code and the relations to the corresponding model artifacts. PIM, PSM) for the system. Transformation from PIM to PIT is discussed in the presented approach. However particular concepts are still to be completed. Also focus is to be put on input/output signal specific information on the system model as well as on the test model side. Test implementation code (e.g. TTCN-3) for time continuous behaviour is to be developed. Research on this will allow us to investigate the relation between object and function oriented design on system model and test model side.

In Mobile applications software development, testing and quality evaluation is one of the most important factors and can affect the entire Mobile applications software development process. These factors require lot of time, so reducing the testing and evaluating time is an effective factor to release the product early.

In this paper, we proposed manual testing as way to make up for problems of automated testing. We proposed model based on existing problem of manual based testing areas and analysis the requirements. Using this technique, we can achieve more effective testing on hardware related software areas. Test results can be used as resource for other testers for sharing of experience.

Acknowledgments.

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the CITRC(Convergence Information Technology Research Center) support program **(NIPA-2013-H0401-13-2008)** supervised by the NIPA(National IT Industry Promotion Agency)"

References

[1] Design of Mobile applications Controllers for Safety Critical Systems, http://www.columbus.gr /innovation.htm

[2] Z. R. Dai: Model-Driven Testing with UML 2.0, Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA'04), Canterbury, England, September 2004.

[3] U. Brockmeyer, Automatic Model Validation and Automatic Test Generation in a Model based Development Process, OSC – Mobile applications Systems, Model-Based Design Conference, Munich 2005

[4] U. Brockmeyer, Automatic Model Validation and Automatic Test Generation in a Model based Development Process, OSC – Mobile applications Systems, Model-Based Design Conference, Munich 2005

[5] B.Dobing, J.Parsons. How UML is Used. *Communications of the ACM*, 49(5), 109-113.

[6] OMG. Introduction to OMG's Unified Modeling Language, Version 2.0. Object Management Group, http://www.omg.org/gettingstarted/what_is_uml.htm

[7] J.Arlow, I Neustad. UML2 and the Unified Process: Practical Object-Oriented Analysis and design. Pearson Education, Inc, 2005.

[8] V.Garousi, L.Briand, Y.Labiche. Control flow analysis on UML 2.0 Sequence diagram. *In Model Driven Architecture:Foundations and Applications*, Lecture Notes in Computer Science, vol.3748, Springer, Berlin, 2005.

[9] A. Cavarra, C.Crichton, J.Davies. A method for the automatic generation of test suites from object models. Information and Software Technology vol.46 no.5, pp. 309-314, 2004.

[10] Y. G. Kim, H. S. Hong, D. H. Bae, S. D. Cha. Test cases generation from UML state diagrams. Software vol.146, no4, pp. 187-192, 1999.

[11] J. Offutt, A. Abdurazik. Generating tests from UML specifications. *Proceeding of the second International Conference on UML*. Lecture Notes in Computer Science, 1723, Springer-Verlag , GmgH, Fort Collins, TX, pp. 416-429, 1999.

[12] S.Kansomkeat, W.Rivepiboon. Automated-generating test case using UML statechart diagrams. *Proceedings of SAICSIT 2003*, ACM, pp.296-300, 2003.

[14] L. Briand, Y. Labiche. A UML-based approach to system testing. *Journal of Software and Systems modeling 1 (1)*, pp. 10-42, 2002.