

UNIFIED TEST ENVIRONMENT-INTEGRATED PLATFORM FOR BRIDGING THE MODELING, TESTING AND CODE DEVELOPMENT FLOW

Jongpil Choi
Korea Polytechnic Univ
Jungwang-Dong 2121
Shihung City, 429-793
+82-31-496-8284
jpchoi@kpu.ac.kr

Sifat Islam
Florida Atlantic Univ
777 Glades Road
Boca Raton, FL 33431
561-297-3000
sislam3@fau.edu

Ravi Shankar
Florida Atlantic Univ
777 Glades Road
Boca Raton, FL 33431
561-297-3000
ravi@cse.fau.edu

Abstract – This study proposed the solutions of three problems. The first problem is that there is no unified platform to integrate various methodologies. The second problem is that UML tools need improvements for practical use. The third problem is that even if the developed code matches with requirement specs, there are bugs.

This study provides an unified and integrated test environment (UTE) that can be used to combine various methodologies. UTE can remove the dummy test cases and make the test cases practically usable. TDD or other methods only focus on meeting the requirement specs. This study provides test cases for the missing specs.

mechanisms. We call this test framework the Unified Test Environment (UTE). The purpose of this study is to provide an ideal Unified Test Environment for the OPP project.

In section 2, development methodologies are introduced and features that we wish to include in a test platform are given. In section 3, an overall description of UTE is given. Section 3 and 4 are descriptions of UTE functions, automatic test generation and TS mapping. Section 4 provides an internal mechanism to implement these functions using an XML processing engine. In section 5, we describe an integrated method of various technologies.

INTRODUCTION

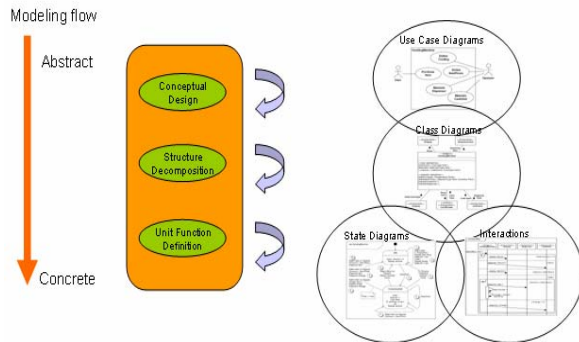
OPP (One Pass to Production) is a joint initiative of Motorola's iDEN group in Plantation, Florida and Florida Atlantic University. OPP is an ambitious project which has a goal of radically reducing the development time of mobile systems [1]. To achieve this goal, many advanced methodologies and technologies should be embedded in the final method and many tentative methodologies should be evaluated to find the final method. A test environment for a large project such as the OPP has to come up with various requirements from a test engineer, software development engineer, software architecture model designer, etc. In this paper, we extract the necessary features of the integrated test environment and provide conceptual-level framework architecture and functioning

METHODOLOGIES

Almost all software modeling processes are top-down and follow from an abstract to a concrete path. Modeling starts from requirements analysis and continues on to the conceptual design. Next we go to the structure decomposition and then to the unit function definition. Figure 1 shows a software modeling flow.

For conceptual design, UML Use Case diagrams or Class diagrams may be used. For the implementation level details, State and Interaction diagrams may be used at the lower level. Some UML tools may be used both at the high-level and middle-level for the convenience of the designer. The boundaries of the diagram usage are not very clear and for this reason the boundaries of the UML diagrams are overlapped in the figure.

Figure 1. Software Modeling Flow



UML can support the entire range of forms of representation, and it is similar to the Hardware Description Language (HDL) in hardware design in that sense. HDL has proven its effectiveness in hardware design and the testing area. Similarly, UML can be used as a modeling language and as a test language. At each modeling phase, we can also verify the designed model using UML.

Traditional software development process uses bottom-up design. After the unit level development, follows the subsystem integration, and then system integration. At each development stage, developers' roles are logically divided. Coding engineers or developers take charge of the unit component level. The Integrator is at the subsystem level and the system integrator is at the top system level.

In a software development flow, the bottom is in a decomposed form and the top is in a composed form. Figure 2 shows a conventional software development flow.

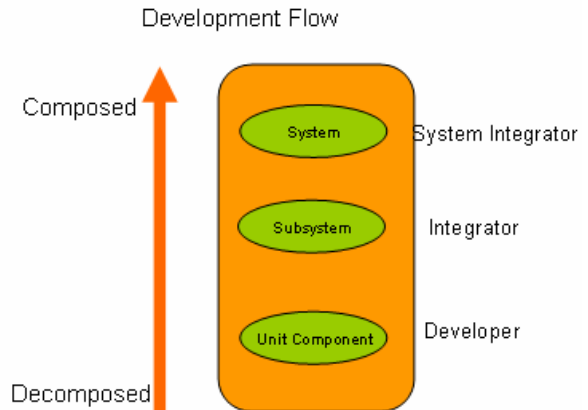
There are three state-of-the-art methodologies that we can use in software development. Figure 3 shows the three software development methodologies.

Model Driven Architecture (MDA) follows the modeling flow in a top-down direction and then goes through the development flow in a bottom-up direction. Most MDA suggest UML as a modeling tool that helps develop software schemes.

Test Driven Design (TDD) skips the modeling flow and emphasizes direct interpretation of user requirements. TDD has a short development path and good coding principles for the testing-coding-refactoring cycle. User requirement feedback is not easy in some large projects, and systematic

planning of a whole system is difficult in TDD. TDD depends on peoples' insights and engineers' capabilities to improve the quality of software. Originally TDD also did not take into consideration the use of component or reuse technology.

Figure 2. Software Development Flow

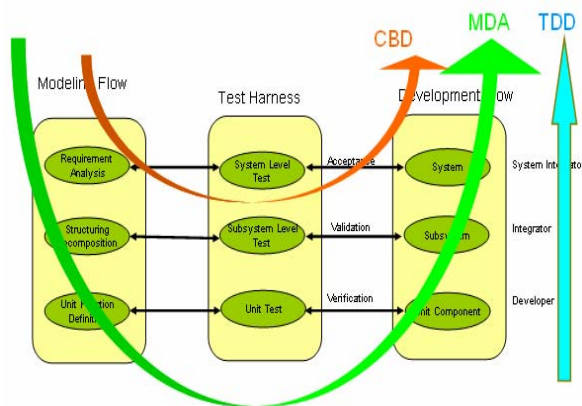


Component Based Design (CBD) can give fast turn around time since we do not delve deep into this process, as in the case of the MDA process. To use CBD successfully, the model in the flow should be well formed and the various performance parameters and characteristic features of the developed code should be well matched with the model. How well do the models and real codes match? We need to formalize the degree to which the model and the code match. Test Harness and Test-Source mapping mechanism, which are topics covered in sections 3 and 4, can be a solution that bridges the gap between the modeling flow side and the development flow side.

So far, many kinds of testing methodologies and technologies were introduced. We need a unified platform to test, compare, and integrate these methodologies. There are some problems faced in a software development site. There is no systematic method to compare the quality of requirement specifications and final products. There is no integrated platform to combine many methodologies and use these as testbeds. Projects start systematically at an early development phase, but we are cornered with the integration phase as time draws near the project deadline. A development schedule allows for only a short testing time in the integration phase. This is a main cause of bugs. Even if the software component is developed and passes the test

criteria, the component deployment environment may be totally different from the original one. This emphasizes the need for tighter coupling of test and design.

Figure 3. Software Development Methodologies



To cope with these problems, we have some means: Use UML from the conceptual level modeling to implementation level modeling and design. Model verification is possible in UML via the UML execution technique. Still we need to go farther and tie the three sides, viz., model design, real code development, and test harness, under the UML umbrella.

In this paper, we propose a framework to be used in integration and prototyping various technologies. We extracted necessary functions for Unified component Test Environment. It has to support the top-down model refining process, the bottom-up traditional software development process, and the component-based development process. Also we need bridging mechanisms to connect the separate UML modeling flow and SW development flow.

These are the required functions to meet the previous objectives:

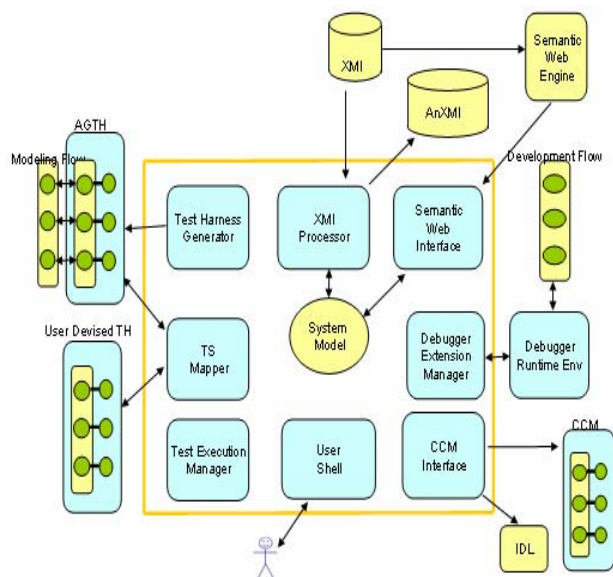
1. Improved automatic test harness generation
2. Tabular formalization for UML model semantic interpretation and source code matching
3. Use of formalized check points as a contract assurance means between UML model and source code
4. UML model execution and source code execution synchronization
5. Debugging and run-time environment supports for TDD coding discipline

6. Integrated framework to combine many different methodologies

Figure 4 shows the overview of the Unified Test Environment (UTE) platform. UML Modeling flow is on the left side, and software development flow is on the right side. The test environment is in the middle and provides functions to meet the previous objectives.

These functions can be categorized into four groups. Test Harness Generation and Test Execution Manager are in one group. Test-Source Mapper and Debugger Extension Manager are in another group. XMI Processor and Semantic Web Interface are in the third group. The rest are in the fourth group. The detailed descriptions of each group are outlined in the following sections.

Figure 4. UTE Overview



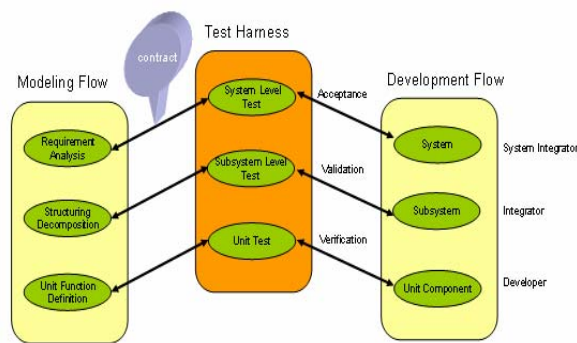
AUTOMATIC TEST GENERATION

Test Harnesses fill the role of real code verification and validation at the lower level of the development flow. It can also be used as criteria to decide code acceptability at the system level. At the modeling flow side, it can be a form of a contract which outlines the exact requirement specifications. Figure 5 shows the role of the test harness.

There are two important roles that help to bridge the modeling flow and the development flow. One of them is to match the model to the development flow as a contract form. If requirement specifications and acceptance criteria are formalized in executable form, it can provide a clear contract between the user and the developer. Test Harness performs part of the first role. The second role is that of the Test-Source Mapper, which can prove this matching in a real execution environment.

Test harness is a testing software that requires test cases for the software testing. The characteristics of the test cases vary according to the abstraction level. We can extract test information from UML diagrams and translate it into programming language. The extracted information can also be used in a TS mapping process.

Figure 5. Role of Test Harness



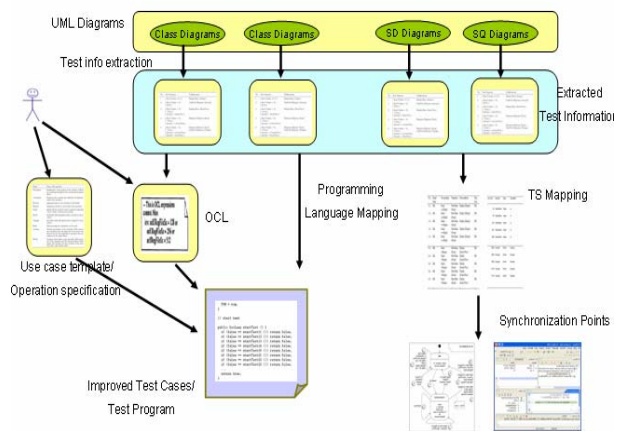
Automatic Test Harness Generator (ATHG) can generate test cases and tester components automatically. The efficiency of the test case generation can be improved using information from high-level conceptual diagrams and well-formed semantic annotations.

We can extract test information from high-level UML specification diagrams such as the Use Case diagrams or Class diagrams. Also we can reduce the test case domains using the Object Constraints Language (OCL). We can add constraints to all types of design objects with OCL. Another information source that we can request from the user is via the operational specification templates [2]. Using this information, we can narrow down possible test cases and improve their quality. Figure 6 shows automatic test case generation flow.

Test code can be packaged into the same component with real code, and this is called a Build-In component Test. Test codes can be fitted as a separate component and the test components may be called mirror components, since the inputs and the outputs are the opposite of the real components. We can use various test case generation algorithms for building test components [3].

Test execution control is a role of Test Execution Manager. It provides automatic test harness generation and controls the test execution. It supports user devised test harnesses as well as automatically generated test harnesses. It includes test definition and test configuration amongst its functionality. Rhapsody TestConductor performs similar functions to Test Execution Manager [4].

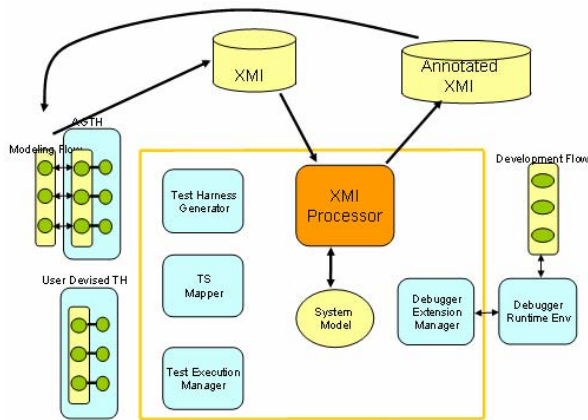
Figure 6. Automatic Test Case Generation Flow



TS MAPPING

The idea of Test-Source Mapper is to connect the modeling side and development side and provide the user with a contract form in a tabular style. It gives a meaningful association between UML models and source code by providing links between certain code structure or execution activities with the corresponding UML representation. It also defines the portions of the real code that we want to verify in high level modeling context in UML. It gives us visible

Figure 9. XMI Processor



This processor reads in the UML model in XMI format and saves it into an internal system model representation. This process also adds annotated descriptions into the system model and writes it into annotated XMI files. For XMI processing, the following tools can be used: JAX (Java API for XMI), SAX (Simple API for XMI) parser, DOM Parser, Xquery/Xpath XML processing tool, Object Constraint Language (OCL), and Eclipse EMF (Eclipse Modeling Framework).

XMI parsers are used for input processing and XML and OCL tools are used for annotation of the system model. For building internal representation of component model and component model library, we have to use the following representations [2, 6].

- Use the package symbol for specifying a component
- Represent a grouping of all descriptive documents and models
- Collectively define a component in terms of functionality, behavior, structure, and external and internal quality attributes
- UML diagrams
- XMI model database
- Annotated XMI description
- OCL description
- Test harnesses and Test cases

Annotated XMI component model contains the representation of the processed information for testing. The OCL description or UML Schedulability Performance and Time (SPT) profile extension can be used for this XMI expansion. The annotated XMI is fed back to the

original UML modeling flow for the back annotation.

The OPP team has another task group studying the semantic web. The purpose of semantic web is to create knowledge bases using ontology methods. These methods can find out hidden requirement specs or inferred spec facts. These techniques can be used to select components by cost function evaluation and find optimal combination of component composition. We can use the information that is generated from semantic web for testing. By utilizing the Semantic Web interface, we can add more test information into the System Model and reach a final verdict about test acceptability or suitability with an associated probability.

TECHNOLOGY INTEGRATION

CORBA Component Model (CCM) is a type of component implementation technique that covers heterogeneous distributed dynamic environment. There are two paths to build the implementation level test harness. The first path is to Build-In-Testing style test harness embedding. The second path is to separate the target code from test harness as different component. CCM is a good candidate for component implementation. We can generate CCM IDL (Interface Description Language) files and CCM skeleton and stub code from System Model. CCM Tool sets and IDL parsers are used for this feature [7].

User Shell is used for completeness. It can provide various user interfaces in a command prompt or a simple GUI. By using this shell we can query about the system info, get statistics and perform controls.

The UTE functions can be implemented with many existing tools or methods. These functions can be integrated using the Eclipse plug-in framework. The three functional groups of UTE are implemented as Eclipse plug-ins and can be integrated to a single platform [5].

CONCLUSION

The purpose of this study was to extract the necessary features and identify the existing tool sets, technologies, and methods to provide

Unified Test Environment. UTE can be used to integrate the OPP verification environment, support model refining process, SW development process, and Built-In Test components or separated test components. This environment has check points to match the model with the source, perform run-time synchronization of high-level models with their source, and annotate XML component model representation. UTE can also generate CORBA Component Model interface and use semantic web inferred information.

There are many benefits of using these techniques. UTE can be used as an OPP component integration test platform. These techniques can improve ATHG, which can relieve the burden of test engineers. Coverage of the missing specs with automatically generated test cases can reduce the possibility of bugs.

ACKNOWLEDGEMENTS

The authors would like to thank the Center for Systems Integration at Florida Atlantic University, specifically the One Pass to Production Project funded by Motorola.

REFERENCES

- [1] OPP project, <http://www.csi.fau.edu>
- [2] Hans-Gerhard Gross, *Component-Based Software Testing with UML*, 2005, Springer
- [3] R. Binder, *Testing Object-Oriented Systems: Models, Patterns and Tools*, Addison-Wesley, 2000
- [4] *Rhapsody TestConductor User Guide*
- [5] TPTP Project Homepage, www.eclipse.org/tptp
- [6] Vijay Madiseti and Chonlameth Arpnikanondt, *A Platform-Centric Approach to System-On-Chip (SOC) Design*, 2005, Springer
- [7] Egon Teiniker, "A Test-Driven Component Development Framework based on the CORBA Component Model", Proc. 27th Annual International Computer Software and Applications Conference, 2003 IEEE