International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 13 (2018) pp. 11096-11103

Model Based Testing – A Case Study Approach

Mettu Sumender Roy[1], Dr. G Samuel Varaprasad Raju[2]

M.Tech. (Ph.D)[1] , Ph.D[2]

[1]Assistant Professor, VIIT, Duvvada, Visakhapatnam, India.
[2]Professor, School of Distant Education, Andhra University, Waltair, India.

Abstract

Design or planning is a prominent stage in any software development life cycle. Such design idea is expressed using a modeling language. These models form the base for development and such developed application is then tested. But now the idea is to test functional units during the design phase and this is termed model based testing. In this review paper I summarize the up to date research on model based testing apart from discussing its add-ons and benefits. I also talk about the further possible study.

Keywords: Model testing, agile metamorphic method, path generation

INTRODUCTION

Software is a key component in many of our devices and products that we use every day. Most customers demand that their software should function as expected. The software should be of high quality, reliable, fault tolerant etc. the key aspects for succeeding in today's software industry is delivering high quality by rigorous testing and good quality assurance practices. One of the main challenges in software development is reducing the associated cost and time of software testing without sacrificing the quality of the developed software.

A software development life cycle comprises of feasibility study, requirement analysis, software design, development, testing, deployment and maintenance as a few common phases.

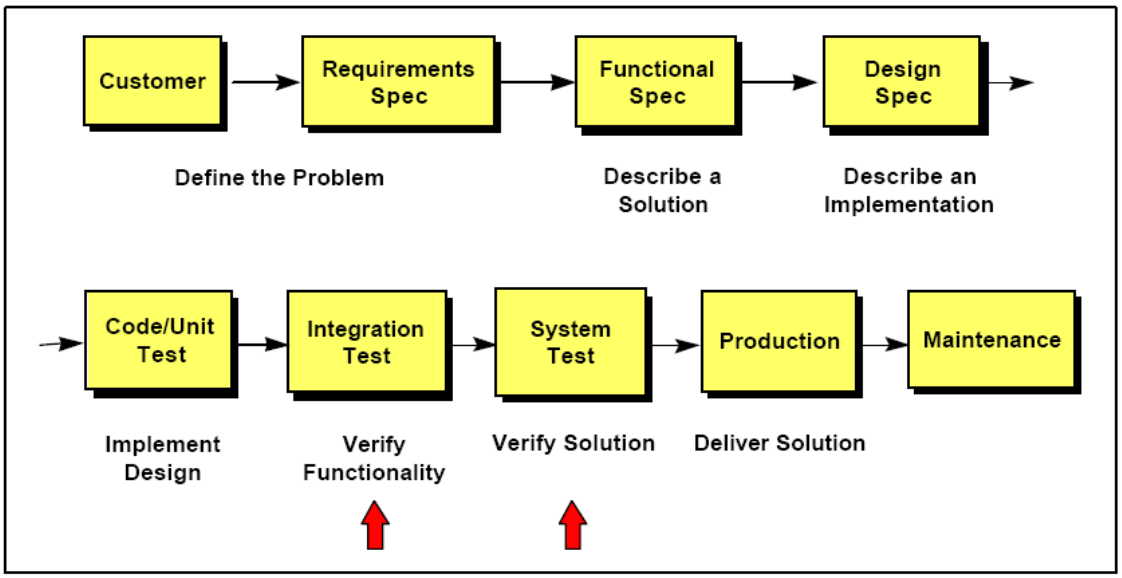

Figure 1: Over view of the phases of a project

Our approach is to automatically generate tests from behavioral models for solving some of these challenges. Software testing is the process of ensuring that the implementation exactly matches the requirements. The recent studies announce that almost 30% of software maintenance cost is spent for bug fixes and this accounts to developer, analysts and end users. An early stage for testing can definitely reduce this cost and thus the model based testing became a focus of this study.

Modeling is expressing the design of a proposed application or its modification. The model expresses the virtual solutions for functional and nonfunctional requirements. And if the model is flawless the forth going phase implementation will almost be bug free.

Model based testing apart from reducing the testing cost also enhances the chance to see the forth coming risk factors and help the project management more lucid and future ready.

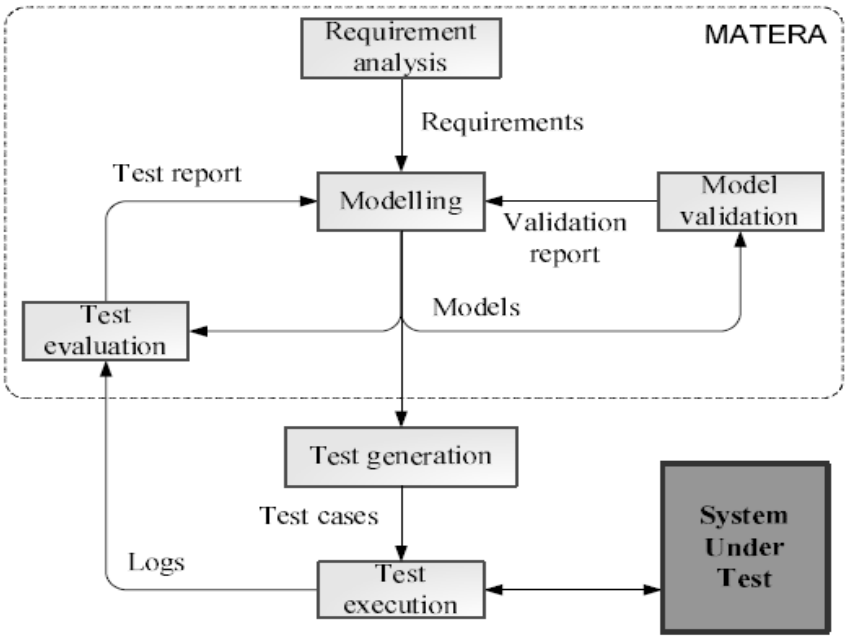

Figure 2: Model based testing techniques

From over a period of 5 to 6 years till now and in the forth-coming years, model based testing has been and will be an interesting and fast evolving idea for research. And in this paper I will discuss the different studies and their applications identified till date. This paper is organized as shown below. Section 2 quotes the inferences drawn from different papers and Section 3 will talk about the scope of future study and I conclude by a brief summary.

MODEL BASED TESTING – STUDIES

The first obstacle to overcome in developing tests is to determine the test target. While this may sound trivial, it is often the first place where things go wrong. A description of the product or application to be tested is essential. Applying a model at this level in a development process can dramatically reduce the ambiguity – and hence, the errors. Subsequent to construction of a model, either complete or partial, the issue of test generation can be addressed. The test objective is to verify that the system will behave properly when a sequence of user actions occurs.

Model Based Testing through Agile Metamorphic Method

Agile Metamorphic method[1] is about developing an automatic test infrastructure to detect design conflicts or flaws. The author proposed this method where DAT queries are automatically generated from a set of metamorphic testing models where each model encodes one or more of the identified equivalences. In addition, we use a scheme for generating time stamps that we use to automatically create time pairs that add another equivalent dimension.

Test Path generation for automated MBT

The author proposes an automated generation[2] of test suits for validating software systems. This is done from the artifacts of requirements and design. Testing using Models is a novel approach which utilizes the key concepts of black-box testing. While substantial part of testing process relies on the appropriateness and completeness of the Model to be used, an approach for designing/modelling a system under test (SUT) with more accuracy is discussed. It can then be combined to the traditional methods of Model-based testing to fill the loop holes and gaps which emerge during design and testing. The approach describes the functionality of the system in a much better way.

Verification-means of safety derivation

A safety case is a contextualized structured argument constituted of process and product-based sub-arguments to show that a system is acceptably safe. The creation of a safety case is an extremely time-consuming and costly activity needed for certification purposes. To reduce time and cost, reuse as well as automatic generation possibilities represent urgent research directions. In this paper, we study in brief the safety processes mandated by prescriptive standards and we identify process-related structures from which process-based arguments (those aimed at showing that a required development process has been applied according to the standard) can be generated and more easily reused.

Example case study of Automated Teller Machine (ATM).

There are many tools that support UML modeling in XML and XMI format. They include ArgoUML, BOUML,

Eclipse UML2 Tools, Enterprise Architect, GenMyModel, MagicDraw, Modelio, Papyrus, Poseidon, Rational Rhapsody, Rational Rose XDE, Rational Software Architect, Rational Software Modeler, StarUML, Visual Paradigm for UML. In order to select a suitable UML tool we opted for a tool that supports UML 2.0 IOD that supports saving diagrams in both XML and XMI format.

It revolves around three entities Customer, ATM machine, and the Bank responsible for the ATM. The ATM machine allows four transactions to the customer in which he/she can do the transactions like deposit funds, withdraw cash, transfer funds and balance enquiry. The operation of the ATM machine starts the validation process where a user inserts the ATM card for accessing the account. If the card passes certain validations, the user is asked to enter the pin number; on the other hand if the card doesn't pass the validations and the card is not valid the transaction is rejected by the machine.

The Card Validation Code (CVC) is a three-digit security code that is printed on the back of cards. The number appears in reverse italic at the top of the signature panel at the end. This program helps *validate* that a genuine *card* is being used during a transaction. Every user has three chances to enter valid pin number when he/she is confused by handling 2 or more ATM cards of different Banks. In three attempts if the user does not enter a valid pin number, the card is automatically blocked and the customer is referred to the bank for card activation. Once correct pin number is entered user is allowed to perform a transaction. The customer is allowed to perform a new transaction if one is finished and a negative answer results in closing of the session and the machine ejects and returns the card to the customer. Activity diagram for the ATM machine operation is as follows.

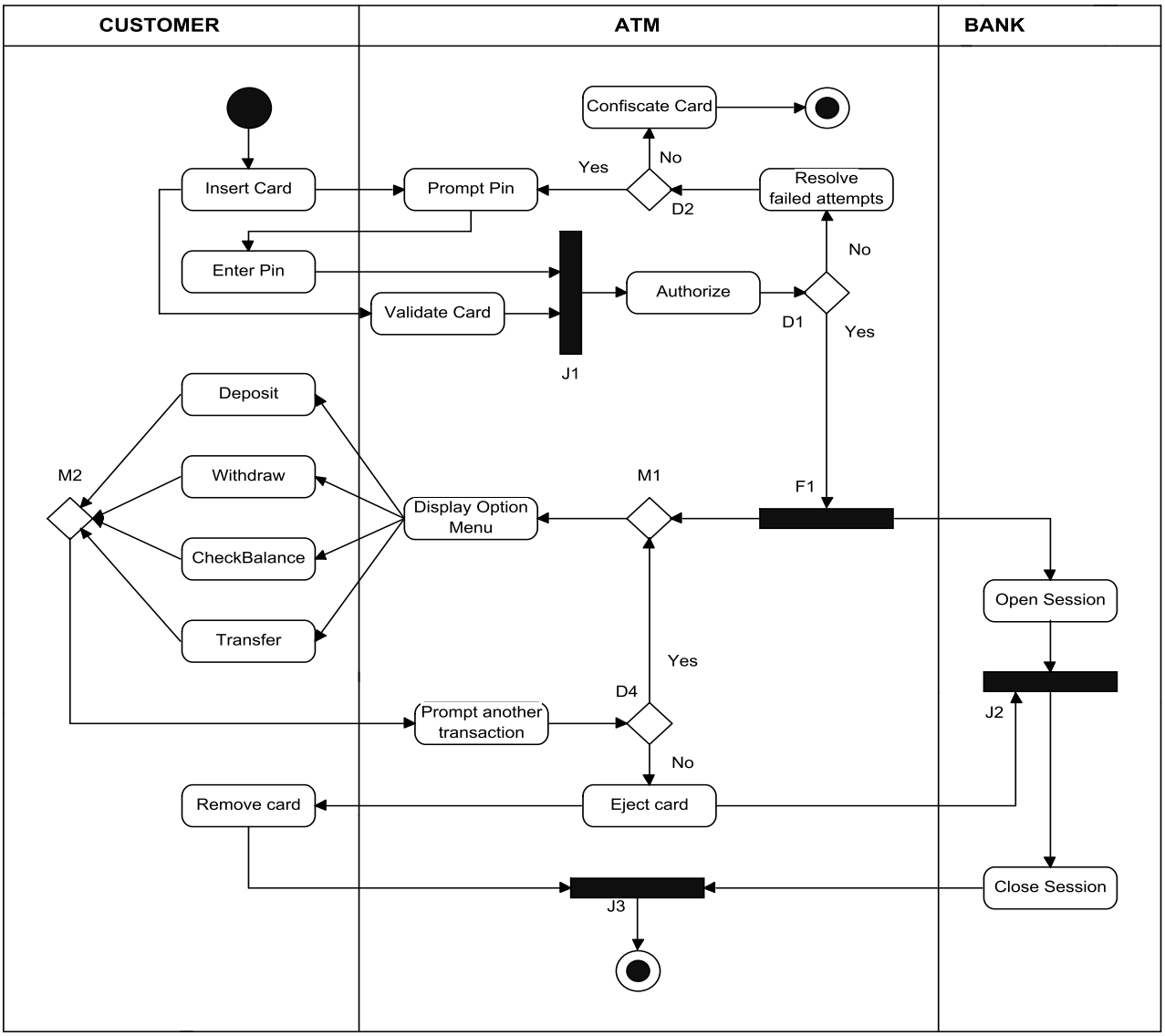

Figure 3: Use Case: ATM

Use case: Deposit Funds.

When you need to deposit cash or checks, your easiest option might be to use a deposit-enabled ATM. There's no need to visit a branch or wait for the mail to get your checks to the bank. Some people don't know that depositing funds at an ATM is possible, while others are uncomfortable with the idea or they just don't know how to do it. For doing this verify that the ATM accepts deposits (and that the ATM works with your bank account). Later the debit card is inserted and asked to choose the on-screen option for deposits. Enter the amount of your deposit if necessary. Some ATMs can figure out how much you're depositing by reading from the check or counting bills – they won't ask for an amount. Confirm that the deposit was completed correctly. Review the amount of the deposit, and, if possible, make any corrections.

The process of depositing through ATM starts with two threads executing concurrently, getting the account number, amount to be deposited and depositing the amount. This transaction tries to commit when another concurrent execution begins by updating balance in the bank and other thread prints the receipt if the customer requires it.

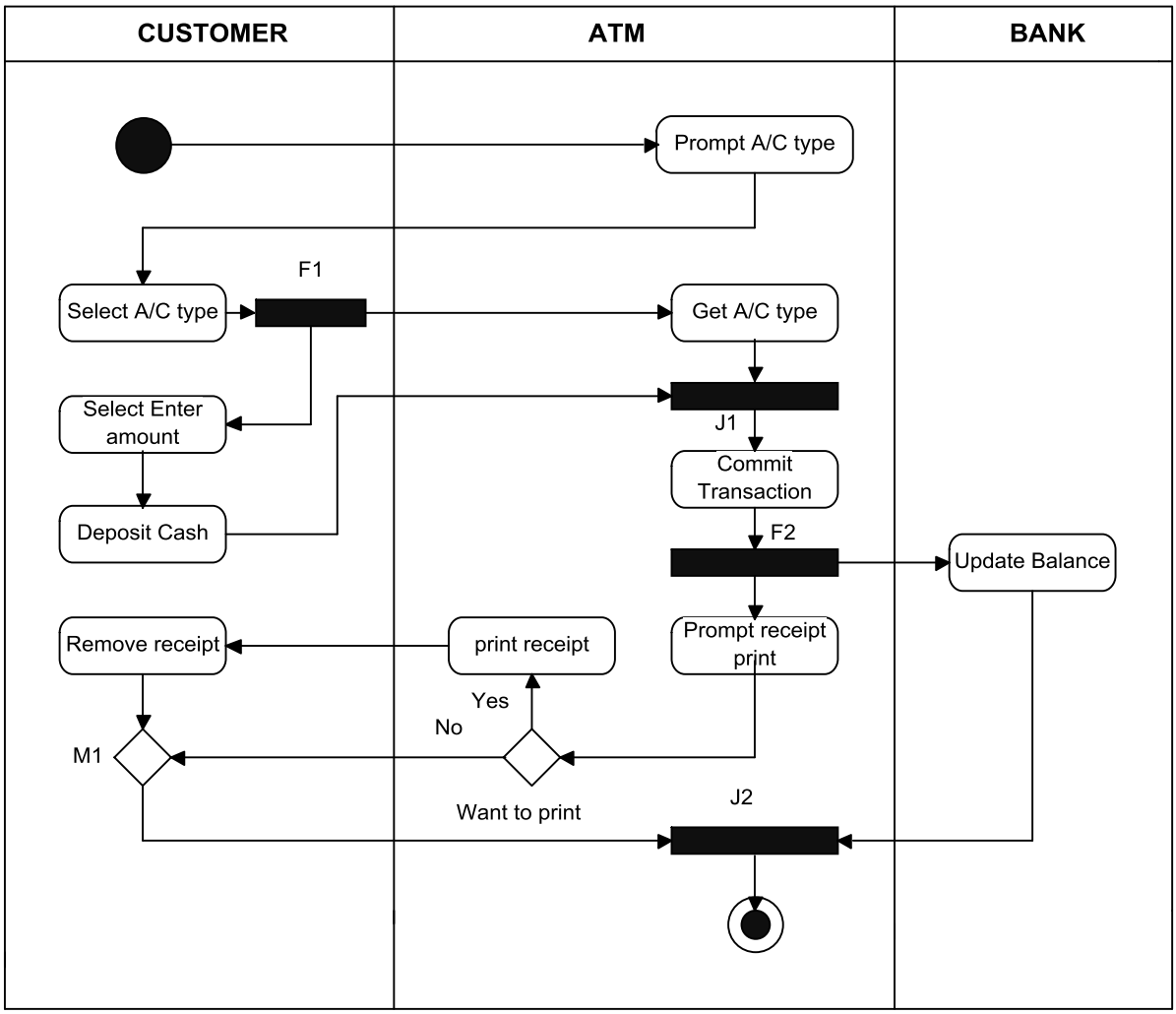

Figure 4: Use case: Deposit funds

Check balance Use case.

When you opened your account with a debit card by inserting into ATM, look for the menu that gives you options. Scroll through these options until you find one that includes the word "balance." You are prompted to Select the option to view your balance. Then, you will be able to go back and make a withdrawal or print that balance. Opt to withdraw money from the ATM. Ask for a receipt. Your checking balance will be printed onto your receipt for your records.

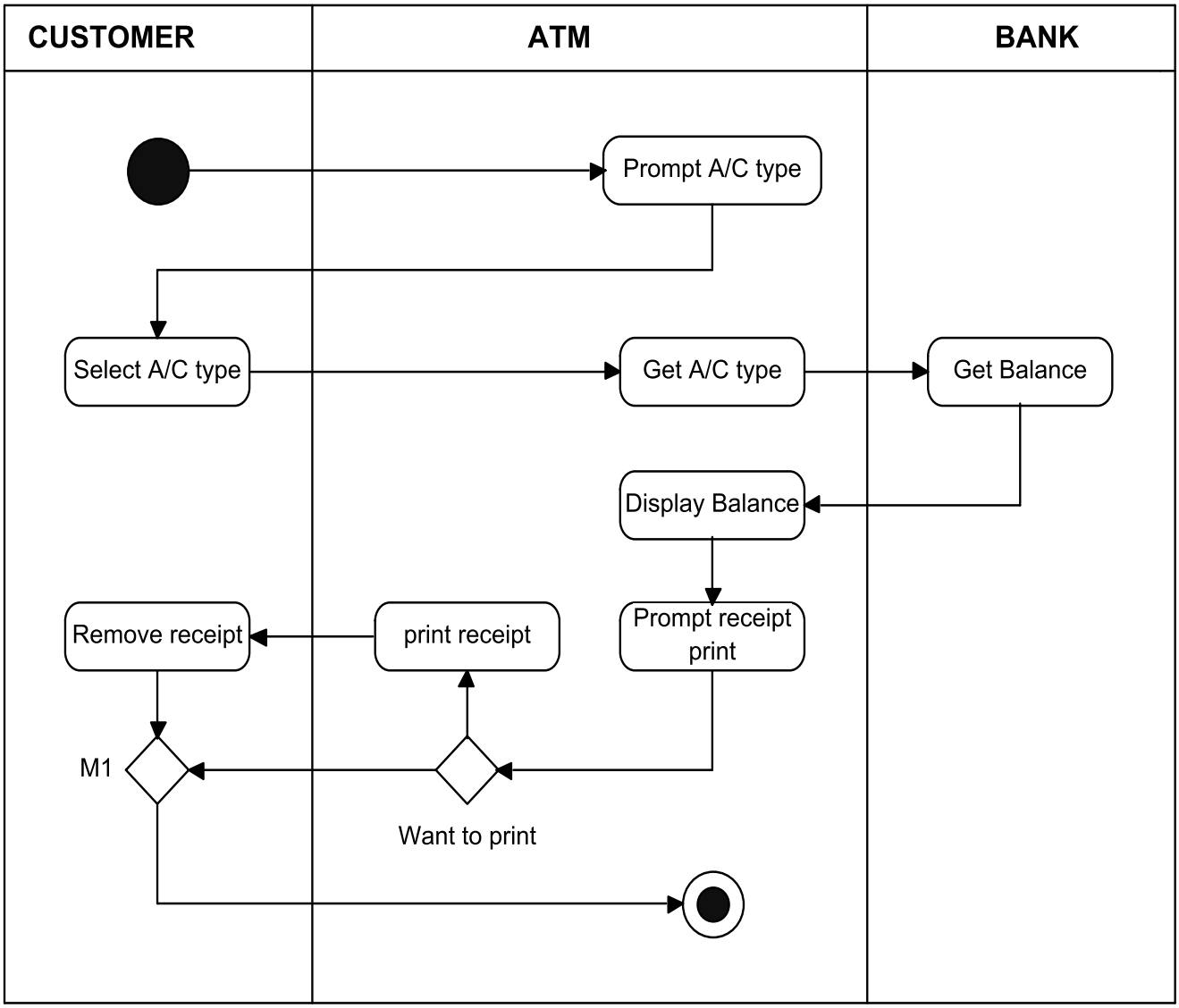

Figure 5: Use case: Check balance

Transfer funds Use case.

Swipe your debit card of State Bank of India. Choose Transfer option (from the various options listed on the screen after swiping your ATM card. Transfer option could be seen at the right bottom side of the ATM screen.) Enter the PIN (after selecting Transfer option you will be asked to Enter the PIN. You have to enter your ATM Card PIN Number there.) Choose Card to Card Transfer option (from the various option listed on the screen after entering the PIN.) Re-Enter Beneficiary's Debit Card Number (you will be asked twice to enter the debit card number of the beneficiary for confirmation). Enter Transferable Amount (you can now enter the amount of money which you want to transfer to the beneficiary's account). Select account type (you will be asked to choose your account type from either Savings or Current). Transaction Completed (your transaction is completed now and the amount of money is credited instantly to the recipient's account).

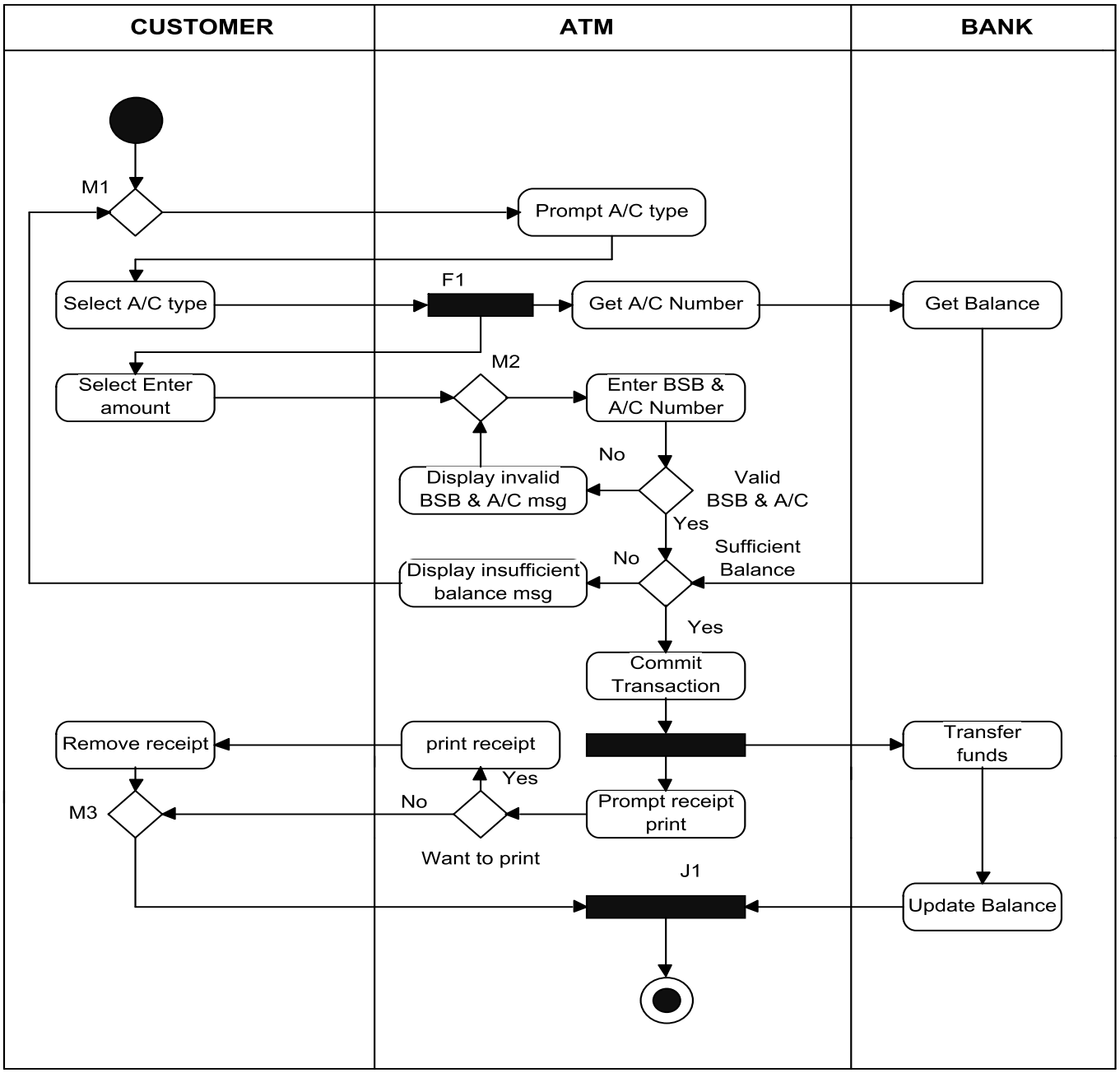

Figure 6: Use case: Transfer funds

With Draw Cash Use case.

Once you select the option, the screen will ask you to enter your four digit ATM pin. Type the ATM pin and press enter. After this, there will be another window giving you options like withdrawal, balance enquiry, deposit etc. In case you want to withdraw, choose the withdrawal option, select the account type (whether Savings A/c or Current A/c), enter the amount to be withdrawn and press enter. You will get the cash coming out of the slot on the lower side of the machine. Collect your cash. An option will appear as to whether you want to do another transaction. If yes, go ahead with another transaction. If no, then click the no option. After this, an option as to whether you want the printed receipt of the transaction done will appear on the screen. In case you want the printed receipt, click yes and close the transaction.

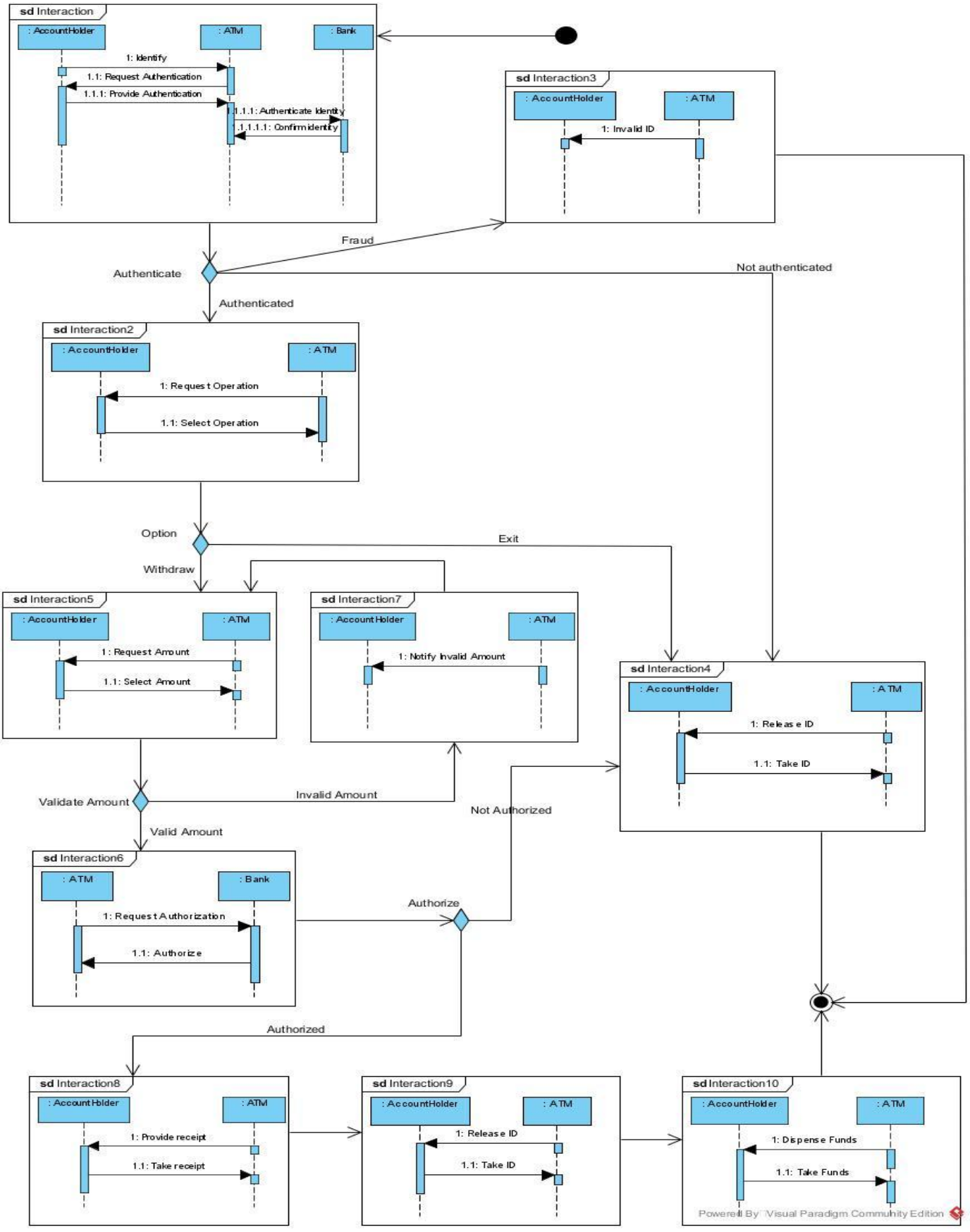

Figure 7. Use case: Withdraw Cash

FUTURE SCOPE

Model based testing still has a lot of open frontiers. Issues like design metaphors based generation of test cases. Auto generation of test data and boundary-level test cases are still to be addressed. Apart from this the test cases are tightly coupled with the design and any change in the design effects the test suite altogether. This issue can also be addressed making test case generation from requirements artifacts and making them independent of design documents and then validating design documents against the generated test cases.

To ensure that the design changes make minimum changes in test suits an xml or any other platform independent semi-structured requirement artifact framework based auto generated test scenarios can be addressed.

CONCLUSION

In this paper I spoke about the recent distinct issues addressed in regard to model based testing. I also observed that the model based testing apart from reducing the testing cost and cost spent on bug fixes , it also unlocks the scope for risk assessment well in advance for the project management team.

Though a lot of work is found addressing variety of issues related to model based testing, we still find a lot of scope as discussed in the future scope. And my future papers are about addressing a few of those refinements.

REFERENCES

[1] Agile Metamorphic Model-Based Testing, Mikael Lindvall; DharmalingamGanesan; SigurthorBjorgvinsson; Kristjan Jonsson; HaukurSteinnLogason; Frederik Dietrich; Robert E. Wiegand; 2016 IEEE/ACM 1st International Workshop on Metamorphic Testing (MET); Year: 2016

[2] An approach to improve test path generation: Inclination towards automated model-based software design and testing; Parampreet Kaur; Ashish Kr. Luhach; 2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO); Year: 2016

[3] Deriving verification-related means of compliance for a model-basedtesting processBarbara Gallina; Anneliese Andrews2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC); Year: 2016

[4] Automated model-based Android GUI testing using multi-level GUI comparison criteriaYoung-Min Baek; Doo-Hwan Bae2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE); Year: 2016

[5] Model-Based Software Product Line Testing by Coupling Feature Modelswith Hierarchical Markov Chain Usage ModelsCerenSahinGebizli; Hasan Sözer2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C); Year: 2016

[6] Model-Based Testing Directed by Structural Coverage and Functional RequirementsYanjun Sun; Gérard Memmi; Sylvie Vignes2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C); Year: 2016

[7] Testing Context-Aware Applications Based on Bigraphical ModelingLian Yu; Wei-Tek Tsai; GianPerroneIEEE Transactions on Reliability; Year: 2016, Volume: 65, Issue: 3

[8] Techniques to generate UTP-based test cases from sequence diagrams using M2M (Model-to-Model) transformation; YongjinSeo; Eun Young Cheon; Jin-A Kim; Hyeon Soo Kim

[9] 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS); Year: 2016

[10] Successive Refinement of Models for Model-Based Testing to Increase System Test EffectivenessCerenSahinGebizli; Hasan Sözer; Ali ÖzerErcan2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW); Year: 2016

[11] Industrial Evaluation of Test Suite Generation Strategies for Model-BasedTestingJohan Blom; Bengt Jonsson; Sven-OlofNyström2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW); Year: 2016

[12] Property-Based Testing with FsCheck by Deriving Properties from Business Rule ModelsBernhard K. Aichernig; Richard Schumi; 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW); Year: 2016

[13] Timed-Model-Based Method for Security Analysis and Testing of Smart Grid SystemsGabriel Pedroza; Pascale Le Gall; Christophe Gaston; Fabrice Bersey; 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC); Year: 2016

[14] A model-driven testing framework based on requirement for embedded softwareHaishen Lei; Yichen Wang; 2016 11th International Conference on Reliability, Maintainability and Safety (ICRMS)