THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, Series A,
Volume 13, Number 3/2012, pp. 251–260

PRACTICAL APPLICATION OF UML ACTIVITY DIAGRAMS FOR THE GENERATION OF TEST CASES

Luis FERNANDEZ-SANZ[1], Sanjay MISRA[2]

[1] Universidad de Alcalá, Dept. of Computer Science, Alcalá de Henares, Spain
[2] Atilim University, Department of Computer Engineering, Ankara, Turkey
E-mail: luis.fernandezs@uah.es

Software testing and debugging represents around one third of total effort in development projects. Different factors which have influence on poor practices of testing have been identified through specific surveys. Amongst several, one of the most important is the lack of efficient methods to exploit development models for generating test cases. This paper presents a new method for automatically generating a complete set of functional test cases from UML activity diagrams complementing specification of use cases. Test cases are prioritized according to software risk information. Results from experiences with more than 70 software professionals/experts validate benefits of the method. Participants also confirm its interest and effectiveness for testing needs of industry.

Key words: UML, activity diagrams, software testing, test cases, risk-based testing.

1. INTRODUCTION

Within software quality assurance (SQA) inventory, software testing is the commonest technique for verification and validation in development projects. Every project includes a specific phase for testing and debugging. This phase usually requires a large percentage, around one-third (ranging from 30 to 35%), of the total effort of the project according to different statistical studies of effort distribution throughout the life cycle [1, 2, 3]. Industry requires solutions for increasing both efficiency and effectiveness of testing methods given the fact that real life projects are strongly influenced by the need of reaching tangible productivity and efficiency goals. The most difficult activity within testing is the design of test cases so classic recommendations (e.g. [4]) highlight the need of careful design and good configuration management of cases. This avoids missing the investment of effort in designing cases for the different testing cycles needed along the development and maintenance of software products.

Specifications are considered the logical starting point to generate a set of test cases which covers most of the functional testing needed to validate a software product. One of the major objective of testing is to check whether the needs of user/customers (majority of which must be mentioned in requirements) is fulfilled or not. Obviously many approaches have been devised to diminish the impact of problems and to prevent errors and miscommunications during requirements. .

In software engineering, the Unified Modeling Language (UML) has become the de facto standard for software modelling. It has been accepted by far as the standard notation for object oriented development. In fact UML models have a widespread use in today's software practices. In the case of specifications, use case models and activity diagrams have been found most useful for verifying and validating requirements with customer representatives among various types of UML artefacts. It has been reported that use case diagrams and use case descriptions narratives (collectively called as use case models) are among the top four widely used UML components [5]. Use case models drive the development of other important UML artefacts along the development cycle according to the so-called "use case driven development". While use case descriptions capture detailed steps in main scenario and alternate scenarios, activity diagrams are often used to connect or weave through individual use cases as well as to offer a complementary view of use case internals. This combination leads to better and more complete specifications [6].

Software professionals require easy-to-use methods and tools oriented to maximizing productivity and efficiency. If they invest time and effort developing necessary UML models for specification, they also want to use them as efficiently as possible for additional activities like designing test cases. In fact, lack of connection between models and test design is one of the limiting factors which negatively influence real software testing practices [7]. Moreover specific surveys show that developers are not sure of using UML in testing and more specifically of the use of UML models for testability [8].

The above point becomes the motivation for our present work. In this paper we have developed a model to help developers to reach higher levels of productivity and efficiency. This model generates automatically functional test cases for acceptance test level taking a software specification with use cases and complementary activity diagrams (AD) as starting point. The method also allows systematic assignment of priority to generated test cases according to risk information entered by software analysts. A tool developed as plug-in for the Eclipse open environment allows easy application in real practice. Data collected from more than 70 software professionals during specific experiential application are included as evidence of its applicability and benefits.

The next section provides the functional test case generation from UML diagram. Our test case generation model is demonstrated and validated in section 3. In Section 4, results from a basic test case design experience with 71 software professionals are presented. The discussion on the results and validation of the model is in the same section. The conclusion drawn is included section 5.

2. GENERATION OF FUNCTIONAL TEST CASE FROM UML DIAGRAMS

Functional testing evaluates the system behaviour based on the requirements given in the specification. Further, use cases are an essential part of UML notation to explain the external behaviour of an information system. Main functional description is based on use cases in development project which use UML diagrams,. Use cases are even more useful not only for developers but also for client verification when combined with AD. The clients may choose the most preferred AD and use case descriptions [6]. An AD is a special case of a state diagram: a model showing the set of different states of an object during its life and the conditions to change from one state to another. In ADs most of states are action states and most of transitions are activated when action related to the previous state finishes.

Specification-based testing takes use cases as main reference. It enables acceptance testing by controlling both functionality and interface behaviour according to agreed descriptions. A number of different types of methods for generating test cases from use cases specifications have been proposed:

- Using non formal specifications of use cases: e.g. the proposal of Heumann [9].
- Scenario-based generation like SCENT [10].
- Statistical-based test generation transforming use case model into an usage model, e.g. [11, 12].
- State–driven test generation with a number of proposals using different strategies and basis for algorithms: e.g. [13, 14, 15, 16, 17].

In general, most important proposals for automatic or semiautomatic test case generation from UML models rely on using state-like diagrams both as reference and general guideline [18, 19]. This happens because they offer good information on dynamic behaviour. In fact, several surveys on test generation based on different models [20, 21, 22, 23] have been carried out in past years. Not all these surveys were concentrated in the problem of automatic generation of functional specification-based test cases from models but most complete ones [22, 23] show a variety of approaches. Additionally the most recent proposals follow this line of action of using state-like diagrams with special attention to the use of AD linked to use cases. Some authors have also investigated the use of other types of dynamic UML diagrams like interaction ones (sequence, collaboration and/or communication diagrams). They are not only used for functional testing but also sometimes for integration testing [24, 25]. However, AD have been included in a good number of proposals (e.g. [26, 27, 28, 29, 30]). Some of them rely on transforming UML AD to specific working formal models which enables easier manipulation of data and generation of test cases (e.g. [26]).

Notwithstanding this, the core idea for the majority of these proposals is the settlement of a generation criteria to be fulfilled by the set of test cases. If we use UML diagrams that describe the different functions and options of utilization of software by a user (like in AD describing use case details), the application of

traditional ideas of coverage (based on the graph theory of McCabe [31]) was considered a good strategy in the first proposals [30]. It been extended to the recent ones which use AD [26][30] where coverage of states and paths is included. In our proposal, a criterion is defined [32]: a minimum set of cases should ensure that the execution trace crosses at least once all the arrows and edges of the graph defined by an AD, i.e. they cover all possible test paths. It can be expressed as follows:

"A use case (as described by its activity diagram) will be tested enough if the execution trace covers all the arrows or transitions in the diagram at least once".

This includes loops (not covered by other proposals which use AD [26]): the concept of test path is used so two partial paths are included for each loop: no loop traversal and one cycle path.

One important feature of the proposed method is that it includes priority of test cases or some risk based strategy to rank generated test cases. Developers need to know which test cases are the most decisive to assure testing of most critical or important functions for acceptance level of quality as shown by empirical data collected from software professionals (see Section 4). Opinion collected from professionals [5] shows that testing is usually compressed between delayed previous stages of development and fixed delivery date to customer so it is essential to have information for contingency rearrangement of test plans.

Different approaches have been devised for risk analysis. One of them is historical analysis to guide testing activities (e.g., [40]). It uses defect analysis of software components to focus test intensity on parts of the software that have been revealed as fault-prone in earlier releases or earlier life cycle phases, such as development. This type of analysis can also exploit different metrics as indicators of potential problems (e.g.. [41]). However, this approach is not suitable for our approach of test case generation because it requires an early application from the very beginning of the project, when we are dealing with specifications and use cases. From the point of view of the test cases, it is preferable to exploit the philosophy of reliability-based risk. This approach explores the probability that the software product could fail in the operational environment. The work in [42] highlights the importance of previous estimations for the probability of occurrence of events that cause failures and the severity of the failure in terms of economic cost or even social or human consequences. Due to the inherent difficulty of this task, it is usual to apply qualitative analysis to estimate what are the factors associated to a risk and by quantitative analysis how this factor affect that risk. This leads to the determination of a risk factor for each option to be tested (e.g. paths in AD representing different scenarios of use cases). This risk factor can be used to negotiate or rearrange the test effort according to risk assumed by the customer.

In the next section, we formally described a method for generating test cases from an AD with extended capabilities to describe use cases behaviour. This method is capable of processing all types of use cases (including the ones with loops) as well as of ranking test cases according to perceived importance of scenarios as indicated by software analysts [43].

3. TEST CASE GENERATION METHOD

AD have been formally defined in different publications even connecting them with the general underlying theory of Petri Nets (e.g. [44][38]). In order to include the information needed to work with the method, an extended formalism for AD si presented. The following elements will be used as basis for defining an AD:

- AD: Activity Diagram
- Ca: Path from the initial node to the final node.
- NA = {graph nodes representing Activities in the AD}
- ND = {graph nodes representing Decisions in the AD}
- NF = {graph nodes representing Forks in the AD}
- NJ = {graph nodes representing Joins in the AD}
- T = Transitions in the AD, i.e., every arrow between two different elements and the associated guard, if it exits.
- No = Origin of an arrow.
- Nd = Destination of an arrow.

An AD can be seen as:

G = (N, T) where

N= NA U ND U NF U NJ U {start point, end point} and T= (edge, guard)

The extended formal model of AD required for the method will be created with a previous three restrictions:

- A well-balanced and verified AD should be used as starting point.
- An acyclic AD is required because it is necessary to ensure that every node in the AD is visited at least once. Two types of loops appear in an AD: those where the probability of any number of cycles are the same and those where the probability of a new cycle is decreasing. First type of loops can be simplified as a new alternate path with fixed probability. Probability of the second one should follow a Poisson distribution.
- The probability of executing a transition should be independent from the previous one. Fault proneness study is omitted to simplify the algorithm.

The elements of traditional AD are the following:

- N'A = {graph nodes representing Activities in the DA plus the list of data to be considered}
- So: N'A = NA x data*, data = (id, value*)

where "*" represent the Kleen closure used in regular expressions

So, now an AD:

G = (N, T') where:

N= N'A U ND U NF U NJ U {start point, end point}

T'= (edge, guard, P, C)

$$edge \subseteq N_O^i \; x \; N_d^j \qquad i, j \in \mathrm{N}^+$$

where

$$i \cdot j = i \vee j$$

and , if

$$a)\; i > 1 \Leftrightarrow N_d \in N_J \wedge \forall k = 1..i \; N_o^k \notin N_F$$

$$b)\; j > 1 \Leftrightarrow N_o \in N_F \wedge \forall k = 1..j \; N_d^k \notin N_J$$

P = Probability of a transaction calculated as:

$$if \; (n_o, m) \notin N_D x N^j \Rightarrow P = 1$$

$$if \; (n_o, m) \in N_D x N^j \Rightarrow 0 < P < 1$$

$$\sum_{i=1}^{n} P_i = 1 \; si \; \{(n_o, m_{d_1})...(n_o, m_{d_n})\} \in edge$$

$$\wedge$$

$$(n_0, m_d) \in edge \Leftrightarrow \exists \, i \in \{1..n\} \;,, \; d = d_i$$

C = Cost of an error in the transition, calculated as:

$$if \; (n_o, m) \notin N_D x N^j \Rightarrow C = 0$$

$$if \; (n_o, m) \in N_D x N^j \Rightarrow 0 \leq C \leq 5$$

Now it is possible to write an algorithm for generation and ranking of paths in the AD (Test Cases). The algorithm is stated as follows:

- For each use case generate an AD compliant to restrictions where:

N' = N x (d1,…dn) x OT and n≠0

a. Assign values to attributes (P, C), using the techniques shown above.

b. Build a sub-diagram where each input data value is included (see fig. 2) and decision nodes referred to correctness of input data disappear. So,

$\forall\ (id, vi) \in data*,\ create\ :$

$a\ new\ node\ Ni$

$a\ new\ transition$

$t = ((N, Ni), (id = vi), P, C)$

and

$\forall\ t = ((No_t, Nd_t), g_t, Pt, Ct) \in T\ \ where$

$gt = (id = vi)\ and$

$(id, vi) \in data\ *$

$deleted\ \ t$

- Let be S the set of different paths existing in the new AD.

$\forall\ \ p \in S$

$let\ \ T_p\ \ the\ \ set\ \ of\ \ transitions\ \ in\ \ p$

$let\ \ T_D \subseteq T_p,\ \ such\ \ as$

$t = (No_t, Nd_t, g_t, P_t, C_t) \in T_D\ :\Leftrightarrow\ No_t \in N_D$

$calculate\ \ the$ probability $\ of\ \ p\ \ as$

$$P(p) = \prod_{t \in T_D} P_t$$

$and\ \ calculate\ \ the\ \ \cos t\ \ of\ \ p\ \ as$

$$C(p) = \underset{t \in T_D}{Max}(C_t)$$

- Rank the paths (Test Cases): use the risk of every path as a function: f (P(p), C(p)).

3.1. Description of application of the method

The method starts with a typical AD (Fig.1) which describes the behaviour of a use case (e.g. deleting user data). At this point, the states where data are entered in the system are represented with activity states. It is also possible to include references to interfaces. These activity states are used to represent the typical testing values for each data input according to traditional boundary and equivalence partitioning techniques. For each arrow, there is a value of risk in terms of two factors: probability of use (P in algorithm) and importance of function (cost of error: C in algorithm).

In the use case described in Fig. 1, we are assuming a user ID of 6 digits. The *Request User ID* data input activity is expanded in the subAD shown in Fig. 2.

Each different path which could be traced across the AD could be considered as equivalent to a use case scenario. The total number of scenarios to be analysed tend to be extremely high if we consider two facts: a) the number of times that a loop is executed within a general path could create two different paths and b) each input action of each path can generate by itself one subpath for each combination of input values. This huge amount of testing possibilities is not practical so one should select a few cases considered as representatives of many others (the rest of options) which are considered as equivalent in terms of testing.

Accordingly, five paths can be found in the diagram of Figure 1 representing the interaction between the users and the system. They are summarized as follows:

- Init deletion, ask for ID to Cancel
- Init deletion, ask for ID, introduce an incorrect ID, introduce a correct ID, deletion is confirmed
- Init deletion, ask for ID, introduce a correct ID, deletion is cancelled
- Init deletion, ask for ID, introduce a correct ID, deletion is not confirmed.
- Init deletion, ask for ID, introduce a correct ID, deletion is confirmed

It is possible to get a whole list of different test paths applying the algorithm described above, each one corresponding to a use case scenario. Each path is identified by the chain of states traversed from initial state to the final one, including the ones corresponding to the specific values for input in the expanded activity states.

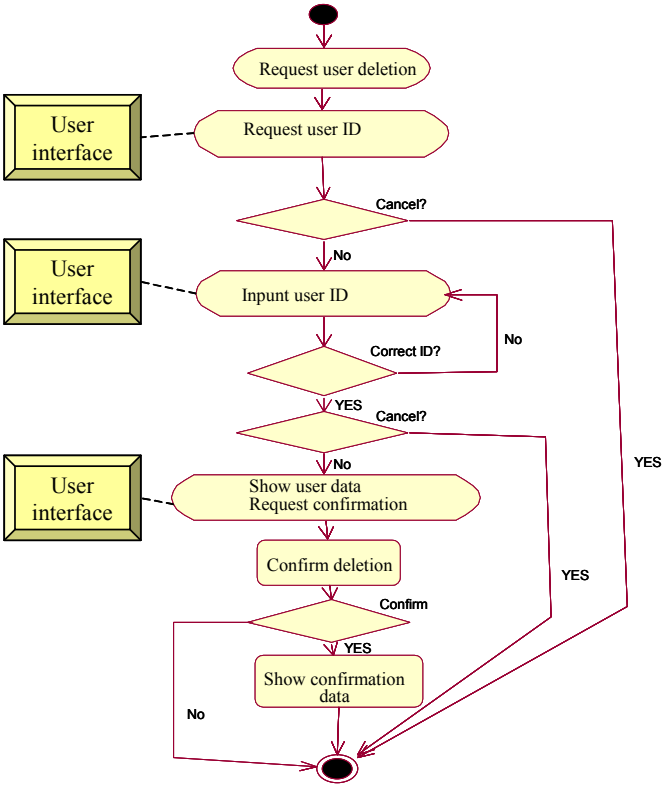

Fig. 1 – Starting AD of a use case for the method description.

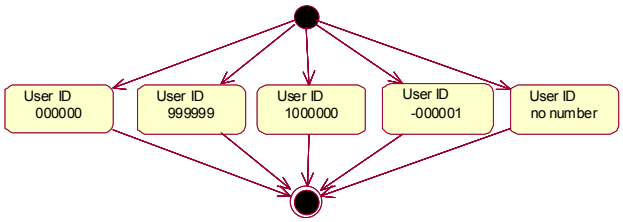

Fig. 2 – Expansion of data input state (activity) for Request user ID.

3.2. Eclipse plug-in for automation of the method

A tool for automating the application has been created. It takes the information of AD using XMI files and integrates in the Eclipse environment [45]. It allows the marking of risk (P, C) labels for each branch as well as the management of expanded branches for each input value. As main result, it enables the creation of a list of paths corresponding to each test case with the corresponding rank related to risk factor.

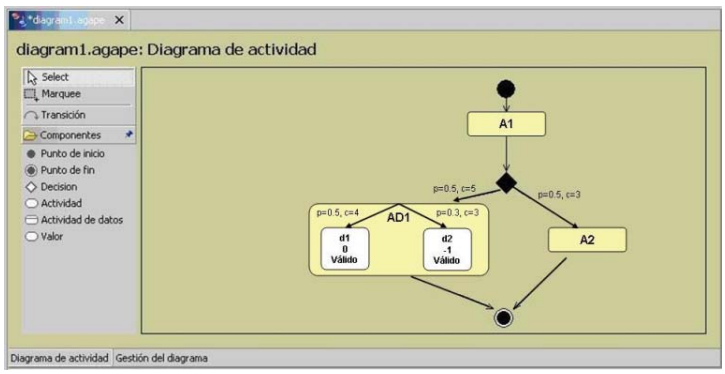

Fig. 3 – Eclipse plug-in.

4. RESULTS OF A TEST CASE DESIGN EXPERIENCE

A test case design experiment was devised for checking the validity and the benefits of the proposed method. More than one hundred IT professionals related to the area of software development (project leaders, analysts, programmers, testers, teachers, consultants, etc.) were involved in the experiment. The followings are the elements of our test case design:

- A simple data management software application for DVD collection management (create, update, delete, queries) was published on a restricted website where participants were specifically invited through direct contact. A brief specification of the application was also included as guideline for participants to carry out test case design. Four defects were intentionally injected in the application to check effectiveness of test cases of participants to detect them.
- Selected participants were encouraged to test the system by entering test cases, the ones they consider most appropriate to test the application and to detect the defects. They should also indicate if each test case reveals or not and a possible defect in the application (after getting results from the application once the test cases has been executed).
- Recorded test cases used by the participants were compared with the set of test cases generated with the method described in section 3. Participants had access to the results of their tests as well as to the comparison with the test cases generated by the method. They were asked to give their opinion using a questionnaire designed to check usefulness of the method and its possible implementation with a tool. They were also asked to give a priority mark to each test case according to their idea of importance for the tested system.

The system recorded the proposed tests and the time devoted to the task. The average devoted time was 21 minutes. Poor quality results (short time, few test cases) were discarded. We finally consider only 72 valid test design proposals (totalizing 1846 executed test cases) corresponding to participants with varied professional profiles (see Fig. 4). The average professional experience of the participants was 5.6 years.

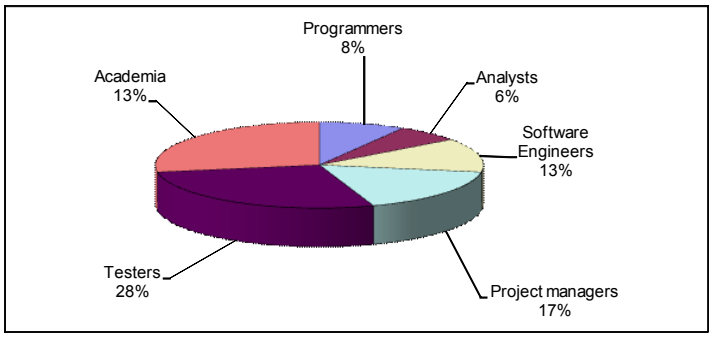

Fig. 4 – Positions of participants.

A lot of interesting information was collected but the main results were the following ones:

- None of the participants included any test case or different scenario (path of the AD) which was not covered by the test design generated by the algorithm. Practical completeness of the proposed solution was achieved.
- Only one participant reached more than 75% of total existing paths; more than 50% of participants did not reach the 50% of the paths/scenarios. Test cases designed by 50 (70%) of participants hardly surpassed the threshold of 25% of total scenarios. Systematic approach for generating test cases from specifications is needed.
- Average number of different scenarios (corresponding to paths in the extended AD) was 39.6% and the average number of injected defects detected was 2. Participants with higher coverage detected more defects. Coverage criteria tend to offer good detection rates.
- At least an average 50% of the effort of testing devoted by a participant was dedicated to test something (a path or the equivalent input data) tested by his/her previous test cases. As an average, 33% of cases were repeating the same functional options. This stresses the need of efficient test design based on coverage criteria of paths/scenarios of use cases.

- Only one of the ten most tested paths by all participants (appearing in the eight place of the list) is also included in the list of the ten paths with most priority (as ranked by the same participants). Three of the most important paths for the participants were included in the list of the ten least tested ones. This confirms the importance of including formal priority criteria as part of the test case generation method.

Apart from data recorded in the system during test cases execution, the results referred to the opinion of participants were also remarkable:

- 56% of them include use cases as part of the specification of software systems, 18% use them depending on the project and only 28% say they never work with them. When dealing with other types of UML diagrams for specification the corresponding figures were 44% (yes), 21% (depends) and 35% (mo). Use case usage is similar to the ones presented in [5] so sample of this experience could be considered as standard in this sense.
- 55% of participants declare that they do not use any method or tool to design test cases while 30% says they follow a method.
- Regarding usefulness of the method and the plug-in, 77% of the testers tell that the algorithm would be really cost-effective for their organizations although it requires creating AD as part of software specifications with use cases. 70% of the participants consider that the need of allocating priority values to the different scenarios following the method would be cost effective in their industrial environments.

The opinion of professionals confirms the usefulness of a method with the following characteristics:

- It works with use cases: 56% of participants usually worked with them while other surveys [5] confirm 73% of professionals used them in 2/3 of projects.
- It offers perceived benefits that 77% of professionals consider they compensate the inconvenience of creating AD for use cases.
- It enables disciplined and systematic design of cases while 55% professionals are not using any method.

As a complementary check, this experience was repeated with 28 last-year students of a computing degree in Spain, all of them with specific training in software engineering and development but not in the proposed method. They received three types of specifications under the philosophy of use cases for the same application: one with textual descriptions (group A), one with use cases descriptions with standardized templates like the ones in [40] (group B) and another one also including AD (group C). Each group was created with a balanced mix of students according to results and grades in the specific training courses. The results were the following ones:

- Testers who used activity diagrams as reference (group C) covered 45% more test cases (scenarios) than those who only had use cases templates (B) and 70% more than group A.
- However, students from C group only reached 70% coverage while the proposed method always enables 100%. All the results were based only in general paths that are not equivalent to the detailed scenarios used with professionals so comparing exactly the performance of students and professionals is not possible,

The results of this complementary experience confirm that the use of AD improves the design of test cases and its inclusion in the proposed method.

5. CONCLUSIONS

Methods for test case generation from specifications (and in general from development models) are essential for an efficient and effective software development. This has been confirmed by specific surveys [7]. UML is considered as defacto standard for development models and its usage is really wide according to results of surveys [5] confirmed also by participants in the experience presented above. Creation of an efficient and effective method for generating functional test cases from the beginning of projects was needed to validate use cases specifications. According to data from the presented experiences, such a method should be based on use case descriptions with AD, and should include a priority allocation mechanism integrated to enable efficiency in testing. Moreover, the method should be supported by a tool which avoids low added value tasks to developers and testers.

Many proposals have been published during the last years as we described in the first sections of this work. The method presented in this paper is capable to show a real improvement in practical experiences of test case design. Participants' opinion has confirmed the advantage of the proposed method over usual practice. It is also considered as cost-effective for real implementation in organizations. Obviously, improvement of the Eclipse plug-in would enable the direct integration of input and, even, expected output specification of test cases with scripting environments for test automation. Of course, it is expected that application of the method and the plug-in with more complex cases in real projects can provide more details of its ability to increase both effectiveness (i.e. covering much more options than manual design) and efficiency (i.e. avoiding non useful repetition of similar test cases).

REFERENCES

1. C., JONES, *Estimating software costs*, McGraw-Hill, 1998.
2. M. GRINDAL, J. OFFUTT AND J. MELLIN, *On the Testing Maturity of Software Producing Organizations: Detailed Data* Technical Report ISE-TR-06-03, Department of Information and Software Engineering, George Mason University, 2006.
3. F. MCGARRY, R. PAJERSKI, G. PAGE, S. WALIGORA, V. BASILI AND M. ZELKOWITZ, *Software Process Improvement in the NASA Software Engineering Laboratory*, Technical Report, CMU/SEI-94-TR-22, SEI Carnegie-Mellon University, 1994.
4. G.J. MYERS, *The Art of Software Testing*, New York, John Wiley, 1979.
5. B. DOBING AND J. PARSONS, *How UML is used*, Communications of the ACM, **49**, *5*, pp.109–113, May 2006.
6. N. BOLLOJU AND S. X. SUN, *Exploiting the Complementary Relationship between Use Case Models and Activity Diagrams for Developing Quality Requirements Specifications*, in ER Workshops 2008, pp. 144–153.
7. L. FERNÁNDEZ, M.T.VILLALBA, J.R.HILERA Y R. LACUESTA, *Factors with Negative Influence on Software Testing Practice in Spain: A Survey*, In: *Proceedings of the 16th International Conference EuroSPI 2009*, pp. 1–12.
8. A. NUGROHO AND M. R.V. CHAUDRON, *A survey into the rigor of UML use and its perceived impact on quality and productivity*, In Proceedings of the Second ACM-IEEE international symposium on *Empirical software engineering and measurement* (ESEM '08), pp. 90–99. 2008.
9. J. HEUMANN, *Generating Test Cases from Use Cases,* The Rational Edge, June, 2002.
10. J. RYSER AND M. GLINZ, *Scent: a Method Employing Scenarios to Systematically Derive Testcases for System Test*, Technical Report, University of Zurich, 2000.
11. G. WALTON, *Statistical testing of software based on a usage model,* Software, Practice and Experience, **25**, pp. 97–108, 1995.
12. J. A. WHITTAKER, *Stochastic software testing*, Annals of Software Engineering, **4**, *1*, pp. 115–131, 1997.
13. J. OFFUTT AND A. ABDURAZIK, *Generating Tests from UML Specifications*, Second International Conference UML'99 – The Unified Modeling Language, Beyond the Standard, 1999, pp. 416–129.
14. M. KANGASLUOMA, *Test Case Generation from UML state chart*, Master Thesis, Helsinki, University of Technology, 2000.
15. S. GNESI, D.LATELLA AND M. MASSINK, *Formal Test-case Generation for UML Statecharts*, 9th IEEE International Conference on Engineering Complex Computer Systems (ICECCS'04), 2004, pp. 75–84.
16. W. PIRES, J. BRUNET, AND F. RAMALHO. *UML-based design test generation*, Proceedings of the 2008 ACM symposium on Applied computing (SAC '08), 2008, pp. 735–740.
17. R. CAVARRA, C. CRICHTON, J. DAVIES, A. HARTMAN AND L. MOUNIER, *Using UML for automatic test generation*, International Symposium on Software Testing and Analysis (ISSTA), Vol. **2280**, 2002.
18. S. ROSARIA AND H. ROBINSON, *Applying models in your testing process*, Information and Software Technology, **42**, *12*, pp. 815–824, 2000.
19. L. C. BRIAND AND Y. LABICHE. *A UML-Based Approach to System Testing*, Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 2001, pp. 194–208.
20. B. AICHERNIG, W. KRENN, H. ERIKSSON AND J. VINTER, *State of the Art Survey- Part a: Model-based Test Case Generation*, Project Deliverable D1.2, 2008, www.mogentes.eu/public/MOGENTES_1-19a_1.1r_D1.2_Survey_Part-a.pdf
21. J. EDVARDSSON. *A survey on automatic test data generation*, Proceedings of the Second Conference on Computer Science and Engineering, 1999, pp. 21–28.
22. J.J GUTIÉRREZ, M.J. ESCALONA, M. MEJÍAS AND J. TORRES, *Generation of test cases from functional requirements. A survey*, 4th Workshop on System Testing and Validation, 2006.
23. C. ARILO, C. DIAS NETO, R. SUBRAMANYAN, M. VIEIRA AND G. H. TRAVASSOS. *A survey on model-based testing approaches: a systematic review*, Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007 (WEASELTech '07), 2007, pp. 31–36.
24. P. SAMUEL, R. MALL AND P. KANTH, *Automatic test case generation from UML communication diagrams,* Information and Software Technology, **49**, *2*, pp. 158–171, 2007.
25. Y. CHO, W. LEE AND K. CHONG, *The Technique of Test Case Design Based on the UML Sequence Diagram for the Development of Web Applications*, Computational Science And Its Applications – ICCSA 2005, pp. 1–10.
26. M. CHEN, P. MISHRA AND D. KALITA, *Coverage-driven automatic test generation for UML activity diagrams*, Proceedings of the 18th ACM Great Lakes symposium on VLSI (GLSVLSI '08), 2008, pp.139–142.
27. L. WANG. J., YUAN, X. YU, J. HU, X. LI AND G. ZHENG, *Generating Test Cases from UML Activity Diagram based on Gray-Box Method*, National Natural Science Foundation of China, 2005.

28. H. LI, AND C. P. LAM, *Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams*, TestCom 2005, pp. 69–80.
29. R. CHANDLER, C. P. LAM AND H. LI, *An Automated Approach to Generating Usage Scenarios from UML Activity Diagrams*, Proceedings of the 12th Asia-Pacific Software Engineering Conference, 2005, pp.134–139.
30. M. CHEN, X. QIU, AND X. LI, *Automatic Test Case Generation for UML Activity Diagrams*, Proceedings of the 2006 international workshop on Automation of software test (AST '06), 2006, pp. 2–8.
32. T. MCCABE, *A Software Complexity Measure*, IEEE Transactions on Software Engineering, **2**, *6*, pp. 308–320, 1976.
32. A.M. MEMON, M.L. SOFFA AND M.E. POLLACK, *Coverage criteria for GUI testing*, 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, 2001, pp. 256–267,.
33. L. FERNANDEZ AND P. J. LARA, *Generación de casos de prueba a partir de especificaciones UML*, VIII Jornadas de Innovación y Calidad del Software, 2003, pp. 48–58.
34. C. STRINGFELLOW AND A. ANDREWS, *Deriving a Fault Architecture to Guide Testing.* Software Quality Journal, **10**, pp. 299–330, 2002.
35. A. VON MAYRHAUSER, J. WANG, M. OHLSSON AND C. WOHLIN, *Deriving a fault architecture from defect history,* International Conference on Software Reliability Engineering, 1999, pp. 295–298.
36. M.L., HUTCHESON, *Software Testing Fundamentals*, Indianapolis, John Wiley, 2003.
37. L. FERNANDEZ, P.J. LARA, J.J. ESCRIBANO AND M.T. VILLALBA, *Use Cases for enhancing IS requirements management*, IADIS International Conference: e-Society, 2004, pp. 541–548.
38. I. TRICKOVIC, *Formalizing activity diagrams of UML by Petri Nets*, Novi Sad Journal of Mathemetics, **30**, *3*, pp. 161–171, 2000.
39. L.FERNANDEZ Y P. LARA, *Test Case Generation, UML and Eclipse*, Dr. Dobbs Journal, **33**, *11*, pp. 49–52, 2008,
40. G. SCHNEIDER AND J.P. WINTERS, J., *Applying Use Cases. A Practical Guide*, Addison-Wesley, 2001.

Received July 13, 2011