ANALYSIS OF PROPOSALS TO GENERATION OF SYSTEM TEST CASES FROM SYSTEM REQUISITES

J. J. Gutiérrez, M. J. Escalona, M. Mejías, J. Torres

Department de Lenguajes y Sistemas Informáticos University of Sevilla (+34) 954552777, 954553867, 954552769, fax: 054557139 {javierj, escalona, risotto, jtorres}@lsi.us.es

ABSTRACT.

System test cases allow to verify the functionality of a software system. System testing is a basic technique to guarantee quality of software systems. This work describes, analyzes and compares five proposals to generate test cases from functional requirements in a systematic way. Test cases generated will verify the adequate implementation of those functional requirements. The objective of this analysis is to determine the grade of mature of those proposals, evaluating if they can be applied in real projects and identifying which aspects needs to me improved.

KEY WORDS.

Test case, system test, use cases, functional requirements.

1. INTRODUCTION.

System testing phase begins when the building of the software system is finished. The objectives of system testing phase are to test the system in depth and verify its global functionality and integrity, running the system in an environment as similar as the final production environment. This verification is based on observation of a controlled set of executions called testing cases. Test process can be expressed like a searching problem. Its main objective is to discover and to correct most bugs as soon as possible [Binder1999].

Nowadays, use cases are the most used tool to express functional requirements. Use cases are also a good artefact to generate system tests cases [Fröhlich2000]. Actually, there are several proposals that describe how to generate test cases from use cases. This work resumes a comparative analysis among five proposals.

This comparative analysis started in March of 2004 and it is still active. Today this comparative includes 12 proposals. The objective of this comparative is to evaluate in depth the state of the art in system test case generation and to build the basis of a new proposal.

Some results from this comparative analysis have been included in [Gutierrez2004]. This work includes a different set of proposals and more comparative characteristics. Conclusions in this work are more precise due this work includes more proposals.

This work is structured as follows. In section 2 the state of the art is described. In section three analyzed proposals are introduced. In section 4 results from the analysis are showed. In section 5 conclusions and future work are described.

2. STATE OF THE ART.

Briefly, process of system test generation from functional requirements consists of build a system model from functional requirements. From that system model, input data, events and expected results are generated [Jacobs2004]. This process is described in figure 1.

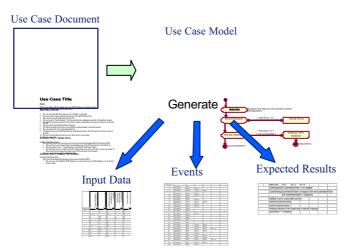


Figure 1. System test cases generation.

Through the time, evolution of proposals has been focused in changing how to express system model. First proposals were based on finite state machines to represent system behaviour. Later, with the apogee of diagrams proposed in UML notation, a new set of proposals and tools appeared to develop system-testing process from use scenarios described by UML diagrams.

Nowadays, there are a great number of techniques to represent functional specifications and derivate test from them. Some examples are model-based specification languages such as Z, algebraic specification languages o sate-based specifications. A complete description can be found in [Offutt1999].

Actually, UML diagrams are widely used to build system model [Bertolino2004]. Diagrams used more often are activity diagram, state diagrams and interaction diagram. A comparative study among these diagrams can be found in [Fröhlich2000].

3. BRIEF DESCRIPTION OF PROPOSALS.

Proposals included in this article are the most actual we have found. None of examined proposals is previous to year 2,002 Next paragraph describes briefly proposals analyzed in section 4.

SCENT [Ryser2003] is a methodological proposal divided in two blocks. In first block, SCENT describes a process to define use scenarios. In second block, SCENT describes how to systematically generate system-testing cases from scenarios obtained in block one. Generation of test cases accomplishes intervening a three steps process. In first step, each test case is defined, indicating what it goes to test. After that, test cases are generated from the distinct paths that can be gone over in the state diagram. Finally, test cases obtained are refined and completed with more test cases developed by classical methods, like stress tests, user interface tests, etc.

Test Cases from Use Cases (TCUC) [Heumann2002] develops a method to obtain a set of system test cases from use cases in three steps. First, all possible path of execution are generated from every use case. Every possible execution path generates a test case. Finally, test values for every test case are identified. Test values include valid and invalid values and outputs expected.

AGEDIS [Hartman2004] is an investigation project financed by the European Union concluded at the beginning of 2,004. AGEDIS main objective has been the development of a set of tools for the automatic generation and execution of tests to verify systems based in distributed components. Although AGEDIS can be applied to any kind of system, better results are obtained applying AGEDIS to control systems, as communication protocols, than to information transformation systems, like compilers. AGEDIS focuses in two products: A system model written in a modelling language called IF, and a set of UML class and state diagrams. These products allow automatic generation of sets of tests and groups of test case objects to link system model and its implementation. This one allows executing tests with system model and system implementation and comparing outputs from model with outputs from implementation.

Use Case Path Analysis (UCPA) [Ahlowalia2002] describes a process composed by 5 steps. The starting point is a textual description of a use case. A flow chart is built from the use case. Using path analysis process a set of test cases is generated.

Requirements by Contract (RBC) [Nebut2003] is divided in two blocks. In first block, this proposal shows how to extend UML use case diagrams adding preconditions, post-conditions and parameters. In second block this proposal describes several algorithms to generate test cases from extended use case diagrams. At the end of this process, a set of test objectives is obtained. A test objective is a sequence of instantiated use cases. A test case generator has to be used in order to produce concrete test cases from those test objectives.

3. ANALYSIS OF PROPOSAL.

3.1. Comparative analysis.

Eleven factors were evaluated for each proposal. Table 1 shows the most relevant factors and next paragraphs describe those factors.

	SCENT	TCUC	UCPA	AGEDIS	RBC
New	Yes	No	Yes	Yes	Yes
notation					
Full	No	No	No	Yes	No
systematized					
Practical	Yes	No	No	Yes	No
cases		-			
Automated	Medium	Low	Medium	Full	Medium
level	N .T	27	N T		
Use of	No	No	No	Yes	Yes
standards	N-	N-	N	Yes	Yes
Supporting	No	No	No	res	res
tools	Middle	Low	Low	High	Middle
Difficulty of	Middle	LOW	LOW	High	Ivitable
implantation Application	Yes	Yes	Yes	Yes	No
Examples	103	103	103	103	110
Coverage	Path Analysis	Path Analysis	Path Analysis	Several	Several
criterion	1 atti 7 titaty 515	i atti / titary 515	1 atti 7 titary 515	Several	Several
Test values	No	No	No	Yes	No
Test case	No	No	Yes	Yes	No
optimization					
Multiple use	Yes	No	No	Yes	Yes
cases					
Test case	Yes	No	No	Yes	Yes
order					
Steps	16+3 (1)	3	5	6	4

Table 1. Comparative analysis.

(1) - 16 steps to obtain scenarios and 3 steps to generate test cases.

New notation indicates if a proposal proposes its own notation or diagrams. SCENT introduces a proprietary usage diagram and UCPA a proprietary notation for flow diagram. RBC introduces a proprietary notation to extend use case diagrams. AGESIS uses IF Language to model the system. TCUC uses natural language only.

Full systematized indicates if a proposal describe how to perform all steps indicates. If proposal is not fully systematized some steps, like build system model form requirements, are not systematically detailed. Only AGEDIS if fully systematized and has a complete set of tools to perform the whole generation process.

Practical cases indicate if there are real project reports in which a proposal has been applied. Only SCENT and AGEDIS includes references to real projects.

Automated level measures the grade in which that proposal can be implanted in software tool. AGEDIS is the only proposal with full level.

Use of standards indicates if a proposal is based in diagrams largely used like UML diagrams. Only AGEDIS and RBC use UML diagrams.

Supporting tools indicate if, nowadays, there are tools to support the proposal.

Difficulty of implantation is based in quantity and difficulty of transformations to realize in each step. A low difficulty indicates a simple proposal to realize, without specific preparation. A medium difficulty indicates that there is new notation or some process that needs a previous preparation. A high difficulty indicates that a proposal cannot be applied without a depth study of its elements.

Application examples indicate if a proposal includes examples, aside from practical cases. All proposals, except RBT, include a practical case.

Coverage criterion indicates the method to generate test cases from system model. Several means that proposal exposes different coverage criterions.

Test values indicate if proposal exposes how to select test values for test cases generated.

Test case optimization indicates which proposal describes how to select some of test cases generated without losing of quality or coverage.

Multiple use cases indicate if proposal can generate test cases that involve more than one use case or proposal can only generate test cases from one use case in isolation.

Test case order indicates if proposal describes the order to execute test cases generated.

3.2. Strong and weak points.

This section describes briefly main strong and weak point of each proposal.

SCENT offers a detailed method to manipulate and organize use scenarios. It includes two references to real projects where it has been successfully applied. However, t is necessary to make a very drawn-out job, 16 steps, with scenarios before generating test cases.

TCUC works with use cases written in natural language, instead of formal use cases. This makes suitable to rapidly obtain test cases, but difficult the automated of process by tools.

UCPA proposes a technique to determine which execution paths are most frequent and critical and which execution path are useless for testing purposes. This technique allows decreasing number of test without affect quality. Weak points are: flow chart notation is simple and hard to apply over complex requirements, and UCPA does not detail how to build test cases once identifies execution paths.

AGEDIS is the most complete proposal. AGEDIS includes generation and execution of test cases. This proposal provides references to five real successful projects. It has a complete tool kit to support all steps of the process. However, AGEDIS cannot be applied to all kind of projects, just only projects that flow controls is more important that information transforming. Tool kit is free for educational purposes only. AGEDIS obligates to adapt to AGEDIS tools.

RBC is supported by a prototype open-source tool called UCTSystem [Generating2003]. RBC generates test cases that include several use cases and describe several coverage criterions to concrete types of system. However, RBC extends use cases ignoring extension rules proposed by UML Consortium. RBC treats use case as black boxes. This means that this proposal cannot generate test cases to verify the whole set of interactions in a use case in isolation.

4. CONCLUSIONS.

Comparative analysis shows that AGEDIS is the most modern and complete proposal but, due its weak points, it is not the definitive solution. None of proposals are definitive; all proposals have some advantage over the rest and some weak points. There are main elements which are not described with enough detail to be applied in practice. Examples are: coverage criterions, or how to use storage requirement to derive test values. None of analyzed proposals describe clearly the result of the generation process. None of analyzed proposals describe de grade of detail of results. All proposals generate test but: how test cases are expressed? can they be directly implemented?. Examples included into proposals describe test cases as tables, but none of proposals defines clearly how to express a test case. Except AGEDIS, none of proposals defines detail level of generated test cases. They also do not generate test cases, only test descriptions that must be refined an implemented.

Actually we are working in answer these questions. Our goal is not to define a new proposal, but takes all elements in exiting proposals and complete their lacks, like several coverage criterions, a formal specification to a test cases, how to generate test code from that specification, etc.

6. REFERENCES.

- [Ahlowalia2002] Ahlowalia, Naresh. 2002. Testing From Use Cases Using Path Analysis Technique. *International Conference on Software Testing Analysis & Review*.
- [Bertolino2004] A. Bertolino, E. Marchetti, H. Muccini. 2004. Introducing a Reasonably Complete and Coherent Approach for Model-based. *Electronic Notes in Theoretical Computer Science*.
- [Binder1999] Binder, Rober V. 1999. *Testing Object-Oriented Systems*. Addison Wesley.
- [Fröhlich2000] Fröhlich P., Link J. 2000. Automated Test Case Generation from Dynamic Models. *ECOOP'00*. Sophia Antipolis and Cannes, France.
- [Generating2003] Generating tests from requirements tool. http://www.irisa.fr/triskell/results/ISSRE03/UCTSystem/
- [Gutierrez2004] Gutiérrez, J.J., Escalona, M.J., Mejías, M., Torres, J. 2004. Comparative Analysis of Methodological proposes to systematic generation of system test cases from systems requirements. *Proceeding of the 3rd workshop on System Testing and Validation*. pp 151-160. Paris, France.
- [Hartman2004] Hartman A., Nagin A. 2004. The AGEDIS Tools for Model Based Testing. *ISSTA04*. Boston, Massachusetts.
- [Heumann2002] Heumann, Jim, 2002. Generating Test Cases from Use Cases. Journal of Software Testing Professionals.
- [Jacobs2004] Jacobs, F. 2004. Automatic generation of test cases from use cases. *ICSTEST'04*. Bilbao. Spain.
- [Nebut2003] Nebut, C.F., et-al. 2003. Requirements by contract allow automated system testing. Proceedings of the 14th International symposium of Software Reliability Engineering (ISSRE'03). Denver, Colorado. EEUU.
- [Offutt1999] Offut, A., et-al. 1999. Criteria for Generating Specification-based Tests. *ICECCS '99*. Las Vegas, Nevada.
- [Ryser2003] J. Ryser, M. Glinz 2003. SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test. Technical Report 2000/03, Institut für Informatik, Universität Zürich.