References

[1] M.J. Gallagher and V.L. Narasimhan, *ADTEST : A Test Data Generation Suite for Ada Software Systems*, IEEE Transactions on Software Engineering, Vol. 23, No. 8, August 1997

[2] A. Gottlieb, B. Botella and M. Reuher, *A CLP Framework for Computing Structural Test Data*, CL2000, LNAI 1891, Springer Verlag, July 2000, pp 399-413

[3] S-D Gouraud, A. Denise, M-C. Gaudel and B. Marre, *A New Way of Automating Statistical Testing Methods*, ASE 2001, Coronado Island, California, November 2001

[4] B. Jeng and E.J. Weyuker, *A Simplified Domain-Testing Strategy*, ACM Transactions on Software Engineering and Methodology, Vol. 3, No. 3, July 1994, pp 254-270

[5] B. Korel, *Automated Software Test Data Generation*, IEEE Transactions on Software Engineering, Vol. 16, No. 8, August 1990

[6] G.T. Leavens et al., *How the Design of JML Accomodates both Runtime Assertion Checking and Formal Verifications*, In Formal Methods for Components and Objects, LNCS Vol. 2852, Springer Verlag, 2003, pp 262-284

[7] B. Marre and A. Arnould, *Test sequences generation from Lustre descriptions: GATeL*, ASE 2000, Grenoble, pp 229--237, Sep. 2000

[8] B. Marre, P. Mouy and N. Williams, *On-the-fly Generation of K-Path Tests for C Functions*, ASE 2004, September 2004, Linz, Austria

[9] C. Michael and G. McGraw, *Automated Software Test Data Generation for Complex Programs*, ASE, Oct 1998, Honolulu

[10] C. Michel, M. Rueher and Y. Lebbah, *Solving Constraints over Floating-Point Numbers*, CP'2001, LNCS vol. 2239, pp 524-538, Springer Verlag, Berlin, 2001

[11] P. Mouy, *Vers une méthode de génération de tests boîte grise "à la volée"*, Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL 2004), June 2004, Besançon, France

[12] G.C. Necula, S. McPeak, S.P. Rahul and W. Weimer, *CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs*, Proc. Conference on Compiler Construction, 2002.

[13] M. Obayashi, H. Kubota, S.P. McCarron and L. Mallet, *The Assertion Based Testing Tool for OOP: ADL2*, In Proc. ICSE'98, Kyoto, Japan, 1998

[14] R.E. Prather and J.P. Myers, *The Path Prefix Testing Strategy*, IEEE Transactions on Software Engineering, Vol. 13, No. 7, July 1987

[15] N.T. Sy and Y. Deville, *Consistency Techniques for Interprocedural Test Data Generation*, ESEC/FSE'03, September 1-5, 2003, Helsinki, Finland

[16] M. Wallace, S. Novello and J. Schimpf, *ECLiPSe: A Platform for Constraint Logic Programming*, IC-Parc, Imperial College, London, August 1997

Systems Testing and Validation Workshop 2004

COMPARATIVE ANALYSIS OF METHODOLOGICAL PROPOSES TO SYSTEMATIC 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 tests verify functionality and system integrity of software system globally. System tests are made at the end of system construction. However, it is possible to begin to plan these tests in the first stages of development. This work describes the gaining to plan system tests soon in development process. This work presents a comparative analysis of four proposals to systematize the process of obtaining of system tests. Later, this work shows strong and weak points of every proposal. These points are useful to decide which proposal adopting and they are the bases to new investigations.

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

1. INTRODUCTION.

System testing phase begins when building of the software system is finished. The objectives of this 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.

System testing planning can begin as soon as first functional specifications of system begin to be available. This allows finding errors, omissions, inconsistencies and overspecifications in the functional specifications when it is easy and economic to correct them, because of the cost to correct errors increases at same time that time among the appearing of defect and his detection increases [7].

This work offers the results of a comparative analysis based on four proposals for generating system test cases from system specifications.

This analysis exposes their common elements and the strong and weak points of every proposal.

1.1. State of the art.

The first proposals to systematize and automatize generation of system test cases from system requirements were based on finite state machines to represent system behaviour. Later, with the apogee of diagrams proposed in UML annotation, a new set of proposals and tools appeared to develop system-testing process from use scenarios described by UML diagrams.

Nowadays, a new group of proposals attempting to integrate those two approximations are rising, whereas approximation based in finite status machine, as approximation based at use scenarios. This group adopts the best ones of both [9]. Within this group of proposal, SCENT [2] and AGEDIS [5] are two of their more representatives ones.

2. DESCRIPTION OF PROPOSALS.

The criterions for proposals selection have been, in first place, to select the more moderns. In this way, none of examined proposals is previous to year 2,002, and even one of them, AGEDIS, has been finalized at the beginning of the year 2,004, so it is the most modern proposal existent at the time to write this work.

In second place, we have looked for proposals based on the actual work to join the finite states machine and use scenarios approximations. We have selected the two most important proposals: SCENT and AGEDIS.

In third place, we have selected a little and fast to apply proposal called Test Cases From Use Cases [1], as an alternative to the two previous proposals.

These three proposals obtain a group of functional tests case. However, there are more kinds of tests like security tests, stress tests, performance tests, etc. For this reason we have selected, as last one, a

Systems Testing and Validation Workshop 2004

proposal that allows deriving test cases to accomplish reliability tests, called UML Based Statistical Test Case Generation [4].

2.1. Method Employing Scenarios to Systematically Derive Test Cases for System Test (SCENT).

SCENT [2], [3] is a methodological proposal divided in two blocks. In first block, SCENT describes a process to define use scenarios, to refine them and to organize them, starting from system functional requirements. In second block, starting from scenarios obtained in block one, SCENT describes how to systematically generate system-testing cases.

In first block, SCENT defines use scenario as an ordered group of interactions normally among a system and a set of actors. A scenario can include a concrete sequence of interactions or a group of possible interactions or execution paths.

The first step to obtain use scenarios is the identification and the wording in natural language of system scenarios. All actors that interact with the system, its roles and the relevant events for the system are identified. With these elements, generic use scenarios are building and priorities are assigned in terms of importance of each use case. From now on, the use scenarios are described in detail and use scenario diagram, including dependences among use scenarios is built. Use scenario diagrams reflect which use scenario must be executed before, or which ones can be executed independently.

Next, alternative execution flows are modelled; indicating how the system must reacts executing these flows. Later, execution flows are added to scenarios. Next, non-functional requisites, as descriptions of user interfaces or performance notes, are added to use scenarios. Finally, use scenarios obtained are validated to verify that scenarios and all generated products reflect suitable and completely the needs of the users of the system. Once validated, use scenarios are translated to state diagrams. These state diagrams are completed with information necessary to be able to generate test cases from them, like preconditions postconditions, data inputs, data outputs and more non-functional requisites.

A set of use scenarios and state diagrams contains information necessary for generation of test cases are obtained at the end of this block.

In second block, 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.

2,2. Test cases from use cases.

This proposal [1] 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. After that, test cases are identified from those use cases. 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.

Starting from use cases and their not formal description, a list of test cases, with their test values and the expected output is obtained.

2.3. UML-Based Statistical Test Case Generation.

This proposal [4], unlike the other ones, is centred on statistical use tests or reliability tests. These tests verify that system fulfils a determined reliability level. The main idea is that different parts of a program do not have to be tested with the same meticulousness because the program spends 90% of its time executing only a 10% of its code. Statistical use tests identify this 10% and checks the reliability of that code.

The process of obtaining reliability tests consists of five steps. In first step, use cases are refined with preconditions and

postcondiciones, alternatives to the main execution path and dependencies to another use cases. After that, use cases are translated into state diagrams. Next, usage model is built from state diagrams. Usage model indicates the probability that a transition occur (illustration 1).

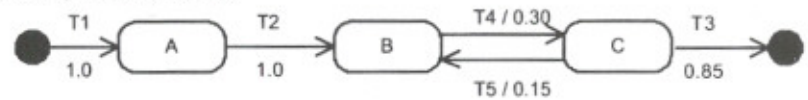

Illustration 1. Example of status diagram with the probability that each transition occurs.

Usage model allows identifying most frequent execution paths. Finally, test models are extracted from usage models and random executions are defined from usage models. Each random execution will be a test case. On execution, these test cases select what path will be taken in every transition in terms of the probability; hence, the majority of test cases will execute the transitions with bigger probability.

2.4. AGEDIS.

AGEDIS [5], [6], [8] 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 methodology with the same name and a set of tools for the automatic generation and execution of tests to verify systems based in distributed components. Although AGEDIS methodology and tools 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.

This methodology exposes an iterative process of six steps (illustration 2). In first step, a behavioural system model is building

from system specifications. UML class diagrams compose a behaviour model where each class has allotted an UML state diagram that describes the behaviour of the objects of that class. From now on, tests objectives are elaborated (use cases tests with concrete data, system charging test, etc.). Test objectives are translated to a generation and execution set of test directives. In next step, a tool generates automatically tests that satisfied these objectives.

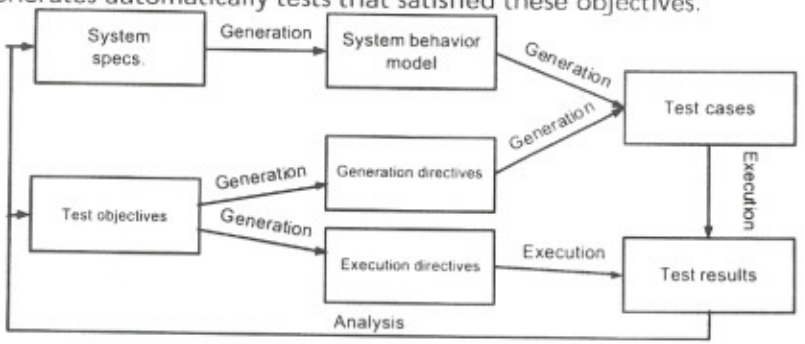

Illustration 2. Description of process to generation and execution of test cases.

At last, test outputs are examined and these six steps are repeated until they attain desired objectives.

3. ANALYSIS OF PROPOSAL.

3,1. Common points and proposals comparison.

A set of common points has been found in analyzed proposal.

1. All proposals start from functional system specifications and allow developing system test cases as soon as first functional specifications are available.

2. All proposals use analysis of all possible patches, from textual description of use case or scenario steps or from state diagrams.

3. Functional specifications do not have to fulfil any formal requirement. It is possible to start working with a brief natural language description.

4. Derivation of system test cases from functional specifications can be made automatically and systematically. All proposals can be automatizated by software tools.

5. The application of these proposals helps validation of functional specifications, checking if they are right and complete at first phases of development process.

Eleven factors were evaluated for each proposal. Table 1 shows the most relevant factors and table 2 describe those factors.

	SCENT	Test cases generation from use cases.	UML-Based Statistical Test Case Generation	AGEDIS
New notation	Yes (1)	No	No	Yes (2)
Practical cases	Yes	No	No	Yes
Kind of tests	Faults	Faults	Reliability	Faults
Use of standards	Yes	Yes	Yes	Yes
Supporting tools	No	No	Yes	Yes
Difficulty of implantation	Middle	Low	Middle	High
Examples	Yes	Yes	Yes	Yes
Steps	16+3 (3)	3	5	6

Table 1. Comparison of analyzed proposals.

(1) - Scenario dependences diagrams.
(2) - IF Language to model the system.
(3) - 16 steps to obtain the scenarios and 3 steps to derive test cases.

New notation	Indicates if a proposal proposes its own notation or diagrams.
Practical cases	Indicates if there are real project reports in which methodology has been applied.
Kind of tests	Indicates if test cases generated are orientated to search system failures or they are oriented to charging tests.
Use of standards	Indicates if a proposal is based in diagrams largely used like UML diagrams.
Supporting tools	Indicates if, nowadays, there are tools to support methodology or automatize their steps.
Difficulty of implantation	It is based in quantity and difficulty of transformations to realize in each step. A low difficulty indicates a simple proposal to realize, for the one that almost requires previous preparation. A medium difficulty indicates there are 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.
Examples	Indicates if a proposal includes application examples, aside from practical cases.
Steps	Indicates the number of steps to obtain a set of system tests from functional specifications.

Table 2. Description of characteristics evaluated in table 1.

3.2. Strong and weak points of each proposal.

Table 3 shows strong and weak points of each proposal.

	Strong points	Weak points
SCENT	This proposal offers a detailed method to manipulate and organize use scenarios. It includes two references to real projects where it has been successfully applied.	It is necessary to made a very drawn-out job, 16 steps, with scenarios before generating test cases.
Test cases generation from use cases	Its strong point is also its weak point. Working with use cases written in natural language, instead of formal use cases, does it suitable to rapidly obtain test cases, but it difficult the automatization of process by tools.	
UML-Based Statistical Test Case Generation	It is the only proposal that allows deriving reliability test cases from functional requirements.	It does not include references to practical cases. It does not provide a way to determine probability. Each probability has to be calculated studying state diagrams and specifications supplied by the customer.
AGEDIS	This proposal is the most complete, including generation, execution and automatization 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.	It can not be applied to all kind of projects, just only projects that flow controls is more important that information transforming. Tool kit is free only fro educational purposes.

Systems Testing and Validation Workshop 2004

4. CONCLUSIONS.

Generation of system test cases at first phases of development process offers an additional validation of system requirements. Generation must begin as soon as requirements begin to be available. Generation of system test cases can be systematized and integrated into development process, in tandem with requirements elicitation phase and can be automatized by software tools.

There are many differences in four proposals examined. This differences offers variety at the time to select one according to the investment in time and resources available. Thus, for example, we can start with a simple method for little projects, like [1] or [8], and evolving gradually to more complete and complex proposal.

AGEDIS is the most modern and complete proposal but it is not the definitive solution. None of proposals are definitive; all proposals have some advantage over the rest and some weak points. So, we think that there is a gasp for a new proposal that joins all strong points of proposal examined and avoid all their weak points. Some initials key ideas about this proposal are: starting from use cases, detecting redundant test cases and calculating the minimum set of test to cover a use case.

5. FUTURE WORKS.

A line of investigation based in this work is the development of a new methodology for systematic generation of system test cases,

which includes strong points of analyzed methodologies, and correcting weak points.

Another line of investigation is to make a study of the implantation process of these proposals in production environments. Applying a system test methodology into an industrial environment imposes additional requirements not exiting in an academic study, like return of inversion.

6. REFERENCES.

[1] Heumann , Jim, 2002. Generating Test Cases from Use Cases. *Journal of Software Testing Professionals.*

[2] Johannes Ryser, Martin Glinz 2003. Scent: A Method Employing Scenarios to Systematically Derive Test Cases for System Test. *Technical Report 2000/03,* Institut für Informatik, Universität Zürich.

[3] Johannes Ryser, Martin Glinz 2003. A Practical Approach to Validating and Testing Software Systems Using Scenarios *Quality Week Europe QWE'99 in Brussels.* Institut für Informatik, Universität Zürich.

[4] Matthias Riebisch, Ilka Philippow, Marco Götze Ilmenau. UML-Based Statistical Test Case Generation. *Technical University,* Ilmenau, Germany

[5] Hartman, Alan 2004 AGEDIS Final Project Report *AGEDIS Consortium Internal Report.* http://www.agedis.de/

[6] Cavarra, Alessandra, Davies, Jim 2004 Modelling Language Specification. *AGEDIS Consortium Internal Report.* http://www.agedis.de/

[7] J. J. Gutiérrez, M. J. Escalona, M. Mejías, J. Torres, 2004. Un studio comparative de propuestas metodológicas para generaciónd e pruebas del sistema. Inner report waiting publication. http://www.lsi.us.es

[8] Ian Craggs, Manolis Sardis, Thierry Heuillard. 2003. AGEDIS Case Studies: Model-Based Testing in Industry. *AGEDIS Consortium Internal Report* . http://www.agedis.de/

[9] A. Bertolino, E. Marchetti,H. Muccini. 2004. Introducing a Reasonably Complete and Coherent Approach for Model-based. *Electronic Notes in Theoretical Computer Science.*