Generation of test cases from functional requirements. A survey

Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, Jesús Torres
Department of Computer Languages and Systems
University of Seville
{javierj, escalona, risoto, jtorres}@lsi.us.es

Abstract:
One of the major quality criteria of a software system is how well it fulfils the needs of users or customers. One technique to verify and improve the grade of fulfilment is system testing. System test cases might be derived from the requirements of the system under test. This paper presents the results of a survey among 13 approaches to drive the generation of test cases from functional requirements.

1. Introduction

Software testing is becoming more complex day by day. This complexity enforces using techniques and methods to assure software quality. One of this methods is system testing. The main goal of system testing is to verify that requirements are successfully implemented into system under test. In order words, system testing assures that software system does what it is expected to do.

The main artefacts to obtain system test cases are the own requirements of the system under test [Bertolino04]. This process might be driven by systematic methods and tools.

Since 2003, we have been studying and comparing existing approaches to derive test cases from the functional specification of the system under test. A preliminary abstract of our survey was presented in SV04 [Gutierrez04]. In this paper we present a more complete survey which validates conclusions exposed in [Gutierrez04] and extends them.

Nowadays, use cases are widely used to define functional requirements [Escalona04], so both terms are synonymous in this paper.

This work is organized as follows. Section 2 describes already analyzed approaches. Section 3 identifies the solved and unsolved aspects in the process of generation of test cases. Finally, section 4 exposes conclusions and future work.

2. State of the art

There are several approaches to derive test cases from functional requirements when those requirements are expressed in a formal and precise notation, like Z or algebraic specifications. However, most of the software industry works with requirements in natural languages. Template and tabular models, like the ones introduced by [Cockburn00] or [Escalona04], are widely used to develop all kind of systems.

This survey is focused on approaches that start from functional requirements expressed in natural language, generally as use cases. Actually, we only know two reports that analyse and compare approaches to generate system test cases from requirements. The first report [Denger03] (called Denger from here on) analyses 12 approaches. The second report is this survey itself (called Gutierrez from here on), which analyzes 13 approaches (listed in table 1). Five of those approaches are also analyzed in Denger report (Id 1, 3, 7, 8 and Category Partition in 11). Additional approaches included in Denger report are listed in table 2. Those approaches are quite similar to the ones in table1, so, they have been omitted.

Id	Year	Title	References
1	1997	Software Requirements and Acceptance Testing.	[Hsia95], [Hsia97]
2	2000	Automated Test Case Generation from Dynamic Models	[Fröhlich00]
3	2000	Extended Use Case Test Design Pattern	[Binder00]
4	2001	Requirement Base Testing	[Mogyorodi02], [Mogyorodi03]
5	2002	A UML-Based Approach to System Testing	[Labiche02]
6	2002	Test Cases from Use Cases	[Heumann02]
7	2002	Testing From Use Cases Using Path Analysis Technique	[Naresh02]
8	2002	Use Case Derived Test Cases.	[Wood02]
9	1999	Scenario-Based Validation and Test of Software	[Glinz99] , [Ryser03]
10	2003	Requirements by Contract	[Nebut03]
11	2003	PLUTO & Category Partition Method	[Ostrand88], [Bertolino03], [Bertolino04]
12	2004	Derivation of Domain Test Scenarios from Activity Diagrams.	[Ruder04], [Ruder04-2]
13	2004	Requirements to Testing in a Natural Way	[Boddu04]

Table 1: Approaches analyzed in Gutierrez Report.

Our previous paper includes four approaches. Only two of them (id 6 and id 9 in table 1) have been included in the final version of the survey. An approach omitted is UML-Based Statistical Test Case Generation due we have focused only on functional verification. Another omitted approach is AGEDIS [Hartman04], since a further investigation and conversation with their authors shows us that it is focused on design models and that it

requires the system built. However, AGEDIS papers are a good source of ideas.

Year	Authors	Title	Reference
1998	Meyer S. Sandfoss R.	Applying Use-Case Methodology to SRE and System Testing.	STAR West Conference.
1999	Carpenter P.B.	Verification of Requirements for Safety-Critical software.	SIGAda'99. Redondo Beach CA. USA.
1999	Collard R.	Test Design: Developing Test Cases from Use Cases.	Software Testing & Quality Engineering Magazine.
1999	Cunning S.J. Rozenblit J.W.	Test Scenario Generation From a Structured Requirement Specification.	IEEE Engineering of Computer Based Systems.
2000	Hindel B. Hehn U.	Constructing Test Cases from Derived Requirements.	EuroSPI.
2002	Blacburn M. Busser R. Nauman A.	Interface-Driven Model-Based Test Automation.	International Conference on Soft. Testing Analysis & Review
2002	Pudipeddi H.V.	Understanding, Designing and Testing Use Cases.	http://www.stickyminds.com

Table 2: Approaches analyzed in Denger Report.

Approaches analyzed in Gutierrez report can be divided into three groups (shown in figure 1) due to their notation for modelling system behaviour. The first group derives test cases directly from requirements in natural language. The second group builds a model from the functionality and derives test cases from that model. Artefacts used to express the behaviour of the system are listed in table 4. The third group is based on the Category-Partition technique. Two approaches join techniques from the two groups. Approach [Boddu04] builds state machines from requirements in natural language as a first step to derive test cases. Approach [Ruder04] builds an activity diagram annotated with categories and partitions.

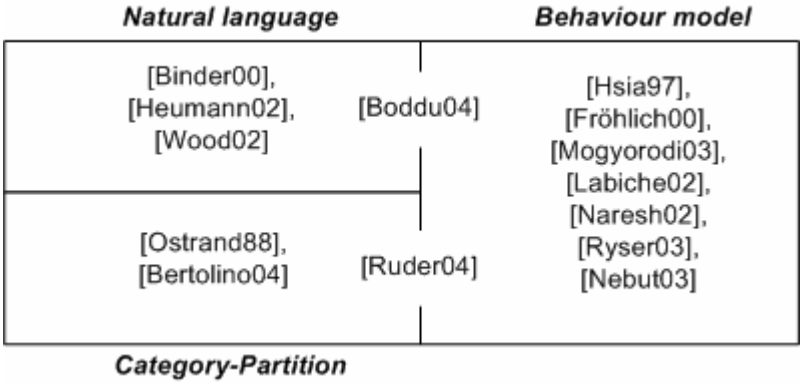

Figure 1: Techniques to modelling the behaviour of the system.

There are also differences among the artefacts needed to generate test cases. Some approaches start from use cases described with UML Use Case diagram [UML03] and completed with templates in natural

language. Other approaches include their own elicitation phase, so their starting point is a set of needs from the users. One approach needs use cases and additional information, like class diagrams and sequence diagrams. A classification is shown in figure 2.

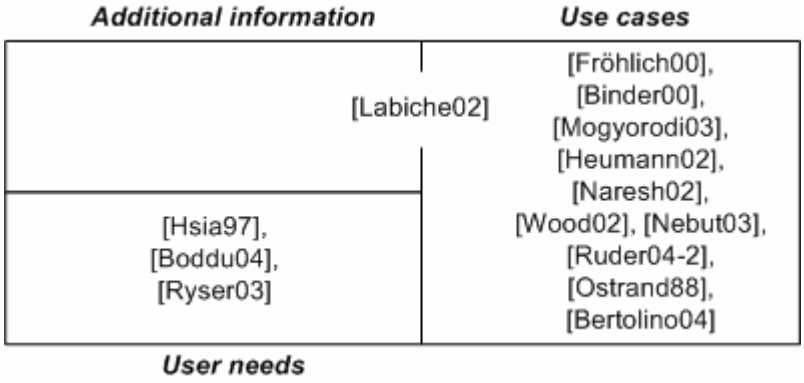

Figure 2: Artefacts needed to generate test cases.

The results obtained from the approaches have been classified into 2 groups (figure 3). The first group denotes that derived test cases are expressed like actions over the system in natural language or decision tables (like [Binder00]). Those test cases must be implemented by hand. The second group allows us to obtain test scripts. However, they used commercial or experimental tools, not available. In some cases, the Category-Partition method can generate executable results, as described in [Balcer89].

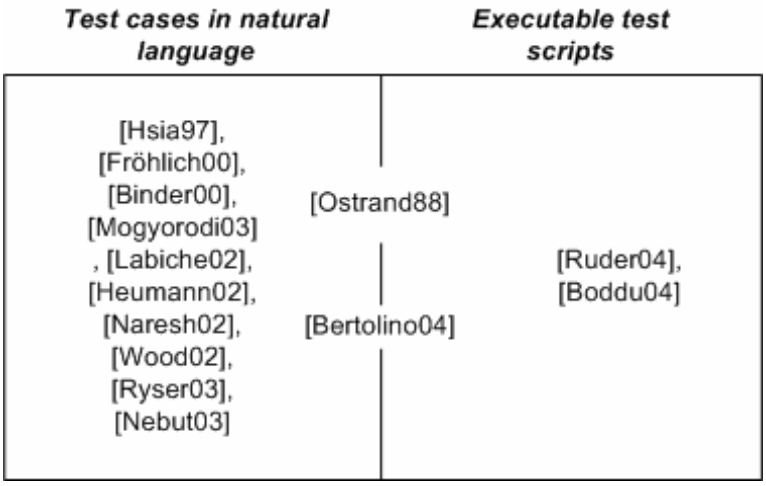

Figure 3: Results obtained from each approach.

There are several approaches that mention a supporting tool. A classification is shown in figure 4. The only tool freely available that we have found is a supporting tool for [Nebut03] approach.

Many of the analyzed approaches include the generation of expected results. However, in all of the approaches, this activity is developed by hand, using the criterion of the testers.

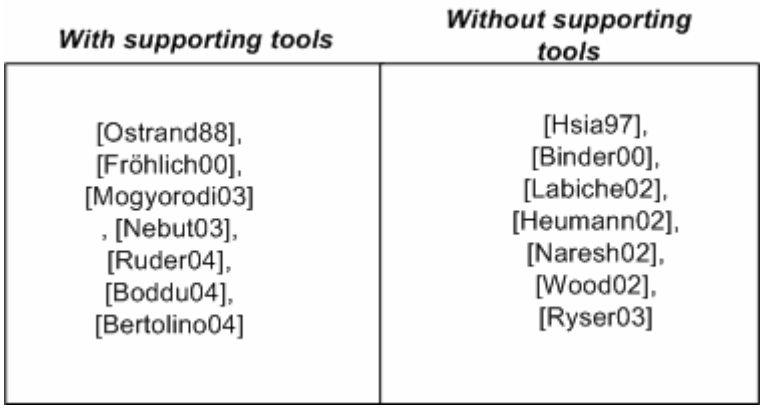

Figure 4: Supporting tools.

Denger and Gutierrez analyses have similar conclusions although Gutierrez report is two years older than the Denger report. The main conclusion of Denger report is that approaches work at a theoretical level without describing how to generate executable test cases. Denger report exposes that there is room for more approaches to solve the troubles and the lacks detected in the exiting ones. The conclusions at Gutierrez report exposes that new approaches have the same lacks than the ones analyzed in Denger. There is a lack of documentation, case studies, and available supporting tools. These facts make hard the application of theses approaches over real projects.

3. Resolved and unresolved aspects

After analyzing and comparing approaches listed in table 1, we have extracted which aspects are treated satisfactorily and which aspects need a further development. Those aspects are listed in table 3 and described in the following paragraphs.

Resolved aspects	Unresolved aspects
• Building behavioural mode. • Supporting tools. • Two levels in test cases derivation. • Definition of requirements	• Documentation and practical cases. • No measure of effectiveness and quality. • Lack of systematization and automation. • Test case implementation. • Lack of empirical studies.

Table 3: Resolved and unresolved aspect identified.

As seen in figure 1, ten approaches develop a behavioural model to express the functionality expected in the system under test. The notations used are listed in table 4.

Reference	Behavioural model
[Hsia97]	Scenario trees and finite state machines.
[Fröhlich00]	Finite state machines.
[Mogyorodi03]	Cause-effect diagrams.
[Labiche02]	UML activities and sequence diagrams.
[Naresh02]	Action-flow diagram.
[Nebur03]	Use cases annotated with precondition, post-conditions and invariants.
[Ryser03]	Own Use cases execution model (similar to FSM).
[Ostrand88], [Bertolino04]	Categories, choices and restriction.
[Ruder04]	UML activity diagram annotated with categories, choices and restriction.
[Boddu04]	Finite state machine.

Table 4: Techniques to generate behavioural model.

The use of behavioural models allows systematize and automate the process. A widely used technique in testing is called model-based testing [Bertolino04-2]. Testing approaches based on model-based testing might also be applied to derive system test cases from behavioural models.

The existence of supporting tools can be classified as a resolved but also as an unresolved aspect. Seven of the approaches analyzed have a supporting tool. This fact is a grade of their maturity and improves their automation. However, an unresolved aspect is to allow the access of these tools by free or open-source licences or, at least, investigation licences, like AGEDIS project.

An aspect discovered, is that it is necessary to derive two levels of test cases. It is well known in code testing that test separate components or fragments of code tested with unit test cases do not guarantee the proper operation of the system [Cohen04]. Thus, an additional integration testing phase which verifies that each component works successfully in collaboration with other components is needed. This philosophy can be also applied to system testing. System testing can be divided into two stages. The first stage verifies the behaviour of each use case in isolation (similar to code unit testing). The second stage verifies that use cases work properly together and satisfy functional requirements (similar to code integration testing). Both stages can be combined in order to obtain test cases that combine several requirements and several execution paths for each requirement. However, there are only 2 approaches (id 6 and 10 in table 1) that include dependences of use cases and derive test cases that involve several use cases. So, this stage has to be studied deeply for new approaches.

An important aspect is how to define a use case to derive a set of test cases form it. Most of the approaches indicate the information that a use case has to include to be useful in test case derivation. As there is not a standard widely accept to define use cases, this aspect is resolved enough.

There are also several aspects that we estimate that they are still unresolved. Those aspects are listed in Table 3. The first aspect is the lack of documentation. All approaches have little or incomplete technical documentation. Many approaches do not refer to any practical application or realistic case study. Both facts obstruct real application of those approaches.

An aspect already forgotten by all approaches is to propose metrics and tools to evaluate the quality of generated test cases. We assume that all approaches start from the assumption that, if their steps are successfully applied, a set of test cases with the maximum coverage are obtained. Since there are several decisions that testing team has to adopt, a metric to evaluate how those decisions affect to coverage is needed. This aspect is also related to the absence of empirical studies about effectiveness of each approach. None of the authors demonstrated with experiments that their approach is better than random testing or than using common sense.

Coverage criterion in the approaches analyzed (except in [Binder00]) is mainly based in a combinational exploration of all possible scenarios from a use case. Some approaches, like [Ruder04], [Nebut03] or [Naresh02], include several coverage approaches (like all-transition or all-states) to minimize the combinational explosion.

The implementation of test cases is also an aspect briefly (or omitted) described in the analyzed approaches. The two approaches ([Ruder04] and [Boddu04]) that derive executable test scripts need tools not available. As seen in figure 3, many approaches just only generate textual description of test cases in natural language. Additional and non-trivial steps have to be performed to implement those test cases into executable scripts or code.

4. Conclusions

The main conclusion is that there are enough approaches to acquire precise ideas on how to derive test cases. However, there is still not a complete and integrated approach that describes the whole process.

From this survey, it is concluded that the most-used technique is model-based testing. This technique is being used more and more every day in the software industry. Model-based testing is successfully applied and

documented when it is applied to design models or when models are extracted from code. Our future work is to develop a model-based testing approach from functional requirement. This approach has to be able to derive test cases from early development phases, allowing improve quality requirements and minimizing time and effort used in system testing [Dahlstedt05].

The increasing number of analyzed approaches, from 4 to 13, has confirmed the preliminary conclusions presented in [Gutierrez04]. None of the approaches is the definite one. With their study as a whole set, we have exposed its resolved and unresolved aspects, so there is a gasp for a new approach which uses the resolved aspect and corrects the unresolved ones.

Number	Task
1	Build behavioural model.
2	Derive test scenarios from one use case.
3	Derive test scenarios from several use cases.
4	Generate test values.
5	Obtain test scenarios.
6	Reduction the number of tests cases without lost of coverage.
7	Measure of coverage.
8	Generate expected results.
9	Sort test cases to maximize a selected criterion (test scenario prioritize).
10	Build test scripts or executable test code.

Table 5: Tasks to generate test cases from use cases.

In paper [Gutierrez04], we exposed, as future work, the development of a new methodology. Nowadays, we have defined the scope of the new approach from the conclusions obtained from Gutierrez report. The scope of our approach is listed in table 5.

Activity	Approach
1	Approaches listed in table 4.
2	All approaches
3	[Labiche02], [Nebut03]
4	Approaches based in CP method [Bertolino04], [Ruder04], [Ostrand88]
5	[Labiche02]
6	[Naresh02]
7	[Binder00]
8	No approaches
9	[Naresh02]
10	No approaches

Table 6: Approaches that mentioned steps listed in table 5.

Comparing the aspects exposed in section 3 with tasks listed in Table 5, we have defined the grade of maturity and coverage of each task from exiting approaches. Results are listed in table 6.

Some preliminary papers and several cases of study have been developed. However, none of them have been published yet.

References

[Balcer89] Balcer M. J., Hasling W. M., Ostrand T. J. 1989. Automatic Generation of Test Scripts from Formal Test Specifications. ACM, pp 210- 218. USA.

[Bertolino03] Bertolino, A., Gnesi, S. 2003. Use Case-based Testing of Product Lines. ESEC/FSE'03. Helsinki, Finland.

[Bertolino04] Bertolino, A., Gnesi, S. 2004. PLUTO: A Test Methodology for Product Families. Lecture Notes in Computer Science. Springer-Verlag Heidelberg. 3014 / 2004. pp 181-197.

[Bertolino04-2] Bertolino A., Marchetti E., Faedo A. 2004. Introducing a Reasonably Complete and Coherent Approach for Model-based Testing. TACoS'04 Preliminary Version.

[Bertolino04] Bertolino, A., Gnesi, S. 2004. PLUTO: A Test Methodology for Product Families. Lecture Notes in Computer Science. Springer-Verlag Heidelberg. 3014 / 2004. pp 181-197.

[Binder00] Binder R. V. 2000. Testing Object-Oriented Systems. Addison-Wesley. USA.

[Boddu04] Boddu R., Guo L., Mukhopadhyay S. 2004. RETNA: From Requirements to Testing in Natural Way. 12th IEEE International Requirements Engineering RE'04.

[Cockburn00] Cockburn, A. 2000. Writing Effective Use Cases. Addison-Wesley 1st edition. USA.

[Cohen04] Cohen, F. 2004. Java Testing and Design. From Unit Testing to Automated Web Tests. Prentice Hall. USA.

[Dahlstedt05] Dahlstedt, Å. (2005) Guidelines Regarding Requirements Engineering Practices in order to Facilitate System Testing. 11th International Workshop on Requirements Engineering. Porto, Portugal.

[Denger03] Denger C., Medina M. 2003. Test Case Derived from Requirement Specifications. Fraunhofer IESE Report.

[Escalona04] Escalona M.J. 2004. Modelos y técnicas para la especificación y el análisis de la Navegación en Sistemas Software. Ph. European Thesis. Department of Computer Language and Systems. University of Seville. Seville, Spain.

[Fröhlich00] Fröhlich, P, Link, J. 2000. Automated Test Case Generation from Dynamic Models. ECOOP 2000. pp. 472-491.

[Glinz99] Glinz M., Ryser J. 1999. A Practical Approach to Validating and Testing Software Systems Using Scenarios Quality. Week Europe QWE'99 in Brussels, Institut für Informatik, Universität Zürich.

[Gutierrez04] Gutierrez J.J., Escalona M.J., Mejías M., Torres J., Álvarez J.A. 2004. Comparative Analysis of Methodological Proposes to Systematic Generation of System Test Cases from System Requirements. Proceedings of the 3rd International Workshop on System Testing and Validation.(SV'2004). pp. 151-160. Paris, France. December.

[Gutierrez05] Gutierrez J.J. 2005. Resolved and unresolved aspects in system test cases generation. Inner Report www.lsi.us.es/~escalona/

[Hartman04] Hartman, A. 2004 AGEDIS Final Project Report AGEDIS Consortium Internal Report. http://www.agedis.de/

[Heumann02] Heumann, J. 2002. Generating Test Cases from Use Cases. Journal of Software Testing Professionals.

[Labiche02] Labiche Y., Briand, L.C. 2002. A UML-Based Approach to System Testing, Journal of Software and Systems Modelling (SoSyM) Vol. 1 No.1 pp. 10-42.

[Mogyorodi02] Mogyorodi G. E. 2002. Requirements-Based Testing: Ambiguity Reviews. Journal of Software Testing Professionals. p 21-24.

[Mogyorodi03] Mogyorodi G. E. What Is Requirements-Based Testing?. 15th Annual Software Technology Conference. Apr. 28-May 1. Salt Lake City, USA

[Naresh02] Naresh A. 2002. Testing From Use Cases Using Path Analysis Technique. International Conference On Software Testing Analysis & Review.

[Nebut03] 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. USA.

[Nebut04] Nebut, C. F., et-al. 2004. A Requirement-Based Approach to Test Product Families. LNCS. pp 198-210.

[Ostrand88] Ostrand, T.Jj, Balcer, M.J. 1988. Category-Partition Method. Communications of the ACM. 676-686.

[Ruder04] Ruder A., et-al. 2004. A Model-based Approach to Improve System Testing of Interactive Applications. ISSTA'04. Boston, USA.

[Ryser03] Ryser J., Glinz M. 2003. Scent: A Method Employing Scenarios to Systematically Derive Test Cases for System Test. Technical Report 2000/03, Institut für Informatik, Universität Zürich.

[UML03] Object Management Group, 2003, Unified Modelling Language 2.0