An overview on test generation from

M.J. Escalona, J.J. Gutierrez, M. Mejías, G. Aragón, I. Ramos, J. Torres, F.J. Domínguez

Keywords:

Testing
Early testing
Functional test generation

ABSTRACT

Despite the fact that the test phase is described in the literature as one of the most relevant for quality assurance in software projects, this test phase is not usually developed, among others, with enough resources, time or suitable techniques.

To offer solutions which supply the test phase, with appropriate tools for the automation of tests generation, or even, for their self-execution, could become a suitable way to improve this phase and reduce the cost constraints in real projects.

This paper focuses on answering a concrete research question: is it possible to generate test cases from functional requirements described in an informal way? For this aim, it presents an overview of a set of relevant approaches that works in this field and offers a set of comparative analysis to determine which the state of the art is.

1. Introduction

Nowadays, there is an important research community focus-ing on the necessity of an efficient test phase in software project and there are interesting discussions and research works deal-ing with the way of getting this efficiency. The test phase is one of the most relevant to both assure the quality of a system and measure the correspondence of the system with the users' expectations.

The objective of software testing is the evaluation of a system or component through its execution, starting from given initial conditions and observing the results (IEEE, 2008). The establishment of such initial conditions consumes much of the effort spent in the development of test cases and testing techniques.

However, despite software community supports the importance of the test phase, it is not always scheduled, executed or faced up with enough resources in real projects. This fact has to do with some aspects such as delays in the development process that commonly generate resources scraps or lack of initial structure of the test in project planning.

In the wide range of tests that can be analyzed, this paper focuses on functional test cases. One of the most important aspects to consider in tests is both the degree of coverage of the initial require-ments defined at the beginning of the project and its validation with final users. Thus, the term *early testing* (Gutiérrez et al., 2006) is used to define a line in test research oriented to enhance the systematic
implementation of test cases based on system requirements or business models.

Information systems are designed to fulfill a set of requirements. These requirements describe, among other aspects, the functionality of the system. The set of requirements that describes the functional necessities of the system is named in this paper *Functional Requirements*. Probably, these functional requirements are based on system objectives or are in need of future users of the system.

For these reasons, a critical task is to assure the right implementation of the expected functionality in the final software. One of the main tools to achieve this goal is system testing, which is a mechanism to verify that the system performs the behavior expressed in its requirements. Thus, test cases must exercise the system under the scope of the requirements. This task makes the requirements, not only a right tool to validate the execution of test cases, but also a source of test cases. *Functional Test Cases* is the name given in this paper to the set of system tests cases that enables the validation of functional requirements, independently how they are obtained. When mentioning *test cases 2 from functional requirements*, we are referring to the set of functional test cases that is obtained in a systematic way from the functional requirement definition.

Therefore, some interesting questions to investigate are whether it would be possible to obtain the total functional test cases from functional requirements and how easy and automatic this process would be.

This systematic generation can be a suitable way to facilitate the execution of the test phase for it offers systematical processes that enable the test generation and test definition in a formal, economic and efficient way. Besides, it guarantees the traceability of the final system with the initial requirements. Thus, this paper

Table 1
Detailed name of the studied approaches.

Year	Title	Reference
1988	Category-Partition Method	Ostrand and Balcer (1988)
1997	Software Requirements and Acceptance Testing	Hsia (1994, 1997)
1999	Testing Object-Oriented Systems	Binder (1999)
1999	Test Cases Generation from UML State Diagrams	Kim et al. (1999)
2000	Automated Test Case Generation from Dynamic Models	Fröhlich and Link (1999, 2000)
2000	Writing Effective Use Cases	Cockburn (2000)
2002	Testing From Use Cases Using Path Analysis Technique	Naresh (2002)
2002	Generating Test Cases from Use Cases	Heumann (2002)
2002	A UML-Based Approach to System Testing (TOTEM)	Labiche and Briand (2002)
2002	Use Cased Derived Test Cases	Wood and Reis (2002)
2002	Quality Web Systems	Dustin (2002)
2003	Scent: A Method Employing Scenarios to Systematically Derive Test Cases for System Test	Ryser and Glinz (2003)
2003	What Is Requirements-Based Testing?	Mogyorodi (2001, 2002, 2003)
2003	Statistical Usage Testing Based on UML	Huebner et al. (2003), Riebisch et al. (2002)
2003	Traceability from Use Cases to Test Cases	Zielczynski (2006)
2004	RETNA: From Requirements to Testing in Natural Way	Boddu et al. (2004)
2004	PLUTO: A Test Methodology for Product Families	Bertolino and Gnesi (2004)
2004	A UML-based Test Generation and Execution	Ruder (2004)
2006	Automatic Test Generation: A Use Case Driven Approach	Nebut and Fleurey (2003), Nebut et al. (2006)
2007	An Automatic Tool for Generating Test Cases from the System's Requirements	Ibrahim et al. (2007), Ismail et al. (2007)
2008	REED. Requirement Editor	Oh et al. (2008)
2007	Test Cases Generation from Use Cases	Gutiérrez et al. (2008a,b)
2009	Software Product Line Testing	Pérez et al. (2009)
2009	Bridging Test and Model-driven Approaches in Web Engineering	Robles et al. (2009)

presents an overview to quantify the degree of maturity, systematization and automation of the existing works about the generation of functional test cases from functional requirements described in natural languages or, at least, in an informal language. In the literature, there are several approaches like (Liu and Yuting, 2008; Stocks and Carrington, 1996) or (TerMaat, 2001) that offer suitable results in the systematic generation of test cases. Still, they start with a formal definition of the requirements by using formal specification. The use of formal languages to define requirements probably provides the best solutions for both, test generation and requirements validation in general, as it is concluded in Ryser and Glinz (1999). Nevertheless, they are difficult to apply in the enterprise environment (Insfrán et al., 2002). For these reason, the study only considers approaches based on functional requirements described in natural or informal language.

Survey generation has been driven by using the guide proposed in SEG (2007). This process consists in three main activities:

1. *Planning*. In this phase, the survey is planned and delimited and a specific protocol to set it up must be defined. Both, the aims of the survey and the environment and sources to identify the approach of study, must be clearly and completely defined.
2. *Conducting the review*. In this phase, after the initial constraints have already been specified, the subject of study must be identified and reviewed with different sources. Then, the set of relevant approaches to be analyzed is detected. SEG (2007) proposes to establish a common characterization criterion to define each approach before the comparative study is carried out. It enables to obtain a uniform definition of each approach that may facilitate the comparative study.
3. *Reporting the review*. Finally, once the approaches have been studied and the research situation has already been analyzed, the process concludes with a report on the results of the review. In fact, this paper represents the result of our review. Previous phases were executed before writing this paper, as referenced in it.

In order to present the overview results, this paper is organized as follows: Section 2 introduces other related surveys about the test cases generation from functional requirements and Section 3 describes the method used to select approaches in the study, and it also introduces a characterization schema to normalize the information of each of the approaches found. This section involves the first and second phases of SEG.

In Section 4, by means of the characterization schema, the paper sets out the characterization schema of the approaches analyzed in this survey as well as other related approaches. From this global and homogeneous view of every approach of study, Section 5 analyzes the information obtained from the set of approaches and presents some relevant studies. Finally, Section 6 states the conclusions and ongoing work.

2. Related work

This paper is not the first one dealing with this subject. There are, at least, three previous comparative studies written with the same aim than this one. In this section, three previous surveys are related and related to this comparative study.

A. The first study presented by Ryser and Glinz (1999) introduce a division and characterization of approaches for validating and testing requirements. In the scope of approaches for testing functional requirements, this reference proposes a division among approaches which work with informal requirements (defined, for example, as text templates), approaches working with semi-formal requirements (defined, for example, as state-machines) and those which deals with formal requirements (defined, for example, with algebraic languages). Nevertheless, this paper puts forward that functional requirements defined in natural language and with text templates are not valid for a process of generating test cases.
B. The second study, developed by Denger and Medina (2003), explains and analyzes 12 approaches for generating test cases from functional requirements. In this comparative study, they conclude that most of the analyzed approaches work with functional requirements in natural language and the study, in turn, reveals the low uniformity and lack of standards in this area.
C. The third survey was conducted by Gutiérrez et al. (2006) in a previous research work. 8 out of 12 approaches are analyzed in a different line from Denger and Medinaïs. Again, the main conclusion is the lack of a common line and a global solution for functional test cases generation.

Starting with these studies and with future work oriented towards finding a suitable solution for test generation, the follow up of this paper is an in-depth analysis of the present situation, focusing on approaches that describe requirements in an informal way, and tries to formalize the comparative study by means of characterization schemas (in the next section). It also enlarges on new approaches not studied in any of the previous surveys and compares the results of these previous surveys.

3. Planning and conducting the review

One of the most important tasks in carrying out a survey is to clearly delimit the scope of the approaches that are relevant for the study. However, one problem is how to present each approach in a homogeneous way to be compared. As introduced, according to SEG (2007), the characteristics that approaches should fulfill must be consistent with the thesis of this survey: *generation of test cases from functional requirements described in informal language*. Thus, following the terminology described in Brereton et al. (2007) for a concrete planning and development of our review, we define:

Context: The systematic, or even automatic generation of test cases from functional requirements, could be a solution to improve the test phase and reduce its cost. There are several approaches that offer different solutions in this area, so a critical overview will be introduced in order to quantify the degree of maturity, systematization and automation of the existing works about generation of functional test cases from functional requirements. This study will focus on approaches that deal with requirements described in informal languages since a previous comparative study, introduced in Section 2, concludes that the formal description of functional requirements makes easier test cases generation. Nevertheless, this notation is not commonly used in the enterprise environment. We want to analyze approaches that define functional requirements in informal languages, like scenarios, patterns, etc., but are widely used in real projects.

Thus, approaches presented in our study must conform to our research line of argumentation

1. The generation process must start from the functional specification of the tested system.
2. The functional specification must be written in non-formal languages.
3. The result must be system test cases.

The relevant approaches for this survey are those that start from the functional specification of the system, should be expressed in the form of functional requirement, use case or usage scenarios and are written in a natural language or similar. There are no constraints on either the structure or size of the approaches. In the survey, the only approaches that are included are the ones which describe how to obtain functional system test cases. This means that generated test cases allow verifying the right implementation of the functional specification in the tested system. Following these criteria, if an approach does not follow these constraints will not be included in the survey. In Section 4.14, an overview of some excluded approaches is presented and reasons for their exclusion are analyzed.

Objectives: After defining the context, the objectives of this study are listed and grouped attending to four points:

- Identify the solutions that have been proposed to address the systematic or automatic generation of functional test cases from functional requirements described in informal languages.
- Analyze, with a set of indicators, if they offer a suitable solution for real projects and improve the test phase with the reduction of cost and time and the coverage increase of functional test.
- Identify the gaps in current research.
- Propose future works about the automatic generation of functional test cases from functional requirements.

Methods: The search strategy for the review consists in three main stages:

- Find published surveys similar to the one used in this paper. Consequently, three previous surveys were found. The search started with approaches included in these papers and the analysis of those which covered the constraints defined.
- Design a Web-search to find other relevant and new approaches. Several sets of keywords were used by combining the start and end artifacts. Some examples are: “test, case, generation, requirement”, “test case, use case”, “system test case approach”, etc. These sets were used both in specific and general search engines such as Google, Google Scholar, Scopus, EI Compendex, ISI Web of Knowledge, IEEEXplore, ACM Digital Library and CiteSeerX.
- Look into references of papers included in previous reviews.

Results: The final result of searching approaches (including all the references from previous surveys and new approaches found) was grouped in 24 approaches which are presented in Table 1.

This table includes 11 approaches (in grayed way) which have a little mismatch with the characteristics set up in the survey. When looking for related works, it is realized that there are some approaches which set out similar ideas to other approaches without introducing a new idea of valuable content. These approaches are not included in the characterization study of Section 4, although the reasons for not analyzing them are specifically mentioned.

In the same way, some of the approaches included in Denger and Medina (2003), Gutiérrez et al. (2006) do not appear in this survey, either. Firstly, despite the reference it has been impossible to find a copy of some approaches included in Denger and Medina (2003), even asking their authors for them. In addition, some of them are no longer maintained. Secondly, there are some approaches in both surveys that do not match the criteria defined at the beginning of this section. For example, some approaches define test cases at a different level of detail than system test level, or some others use formal techniques to define requirements instead of natural languages.

3.1. A characterization schema

Although the approaches shown in Table 1 are described separately and offer a different level of depiction, they are, still, defined within the same context. In line with the definition of characterization schema, our next step in presenting the survey is to define a common pattern to depict in each approach. This schema enables the storage of information of every approach in a common pattern in order to make the comparison among them easier. This characterization schema solves the questions proposed in the survey in order to answer the main question of this research study: *is it possible the automatic generation of test cases from functional requirements described in an informal way?* These questions are

- Q1: Which are the main characteristics of the approach? We want to know if the approach offers a global vision in the survey, which its inputs are, how requirements are defined, etc.
- Q2: Does it offer suitable tools for support? Suitable tools for supporting the development are essential to demonstrate a real generation of test cases from functional requirements.
- Q3: How are test cases obtained? It is necessary to evaluate how test cases are obtained and presented as a result.

Table 2
Characterization Schema for approach description in the survey.

Elements	Attributes	Dominion
Technique	Inputs	{String+}
	Format of the inputs	{no, *}
	Coverage criterion	*
	Notation	*
	Systematization degree	{High, Medium, Low}
	Test values definition	{Yes, no}
Tool	Tool name	*
	License	*
	Available	{Yes, no}
	Automation degree	{total, partial}
	Environment	*
Test case	Amount of test cases generated	*
	Format of test cases generated	*
	Benefits	*
	Problems	*
Documentation	References to projects	{Yes, no}
	Format	{Paper, short paper, inner report, book chapter}, {journal, proceedings, journal, book}
	Size	* (number)
	References	{String+}

- Q4: Is the approach well-documented? Sometimes it is impossible to apply an appropriate approach because it is not well-documented.

Several indicators are selected to answer every question in each approach. They are described in Table 2 and the next paragraphs. Previous works, such as Denger and Medina (2003), Gutiérrez et al. (2006) (cited in Section 2), have been used as worthy sources of information to define elements, attributes and dominions of this characterization schema. The study of testing techniques cited in Vegas et al. (2009) has also been used as an additional source of information about characterization schemas.

In Table 2, each attribute is formalized by defining either the set of possible values (domain) or, in other cases, their format. Domains have been determined through a simplified and auto-explicative notation for regular expressions. Sometimes, it is necessary to allow a free text as a value for an attribute.

To delimit a domain for some fields has been an impossible task, due to the fact that they present a wide disparity of values. These attributes are marked with an "*" in Table 2. Domains between keys "{...}" define a set of options for the value of the attribute. There are some exceptions in the attributes *Inputs* and *References*. Both attributes are described with a list of strings. The domain for the attribute *size* is a number instead of a text string (as indicated in the table). Next paragraph provides a further description of the attributes in Table 2.

As seen in the preceding section and taking into consideration Table 2 above, we now turn to the depiction of each attribute and their corresponding elements. Firstly, we look at the Technique element which describes the test cases generation process which, in turn, answers to question Q1 and is composed of:

- *Input.* It offers the information needed for the process of generating test cases. Its domain consists in a list of strings with the needed artifacts.
- *Format of the input.* It determines how the input information must be defined. Its domain consists in a string which describes this format.
- *Coverage criterion.* It explains which is the testing technique used for generating test cases and so on, when the generation process is finished. This technique is highly based on the definition of the input requirements, for example, if one functional requirement is defined as a state-chart, the coverage criterion may be covering all states or all transitions, but, if the functional requirement is defined as natural text, the coverage criterion may be encompassing all the possible scenarios.
- *Notation.* It analyzes the format used during the generation process such as state-machines or UML activity diagrams.
- *Systematization degree.* It measures the degree of formalism when defining the transformation process. Thus, a high systematization degree will allow the process to be applied as appeared in the documentation. However, a medium and, even, a low automation degree will imply that additional support and human personal decisions are needed in the process. Systematization degree also affects the repeatability of the process. A low degree implies a relevant human support. For this reason, two different persons may obtain different results from the same functional requirement.
- *Test values definition.* It describes if the information required for the test case is widely indicated by the approach.

The second group of indicators oriented to answer question Q2 is *Tool*. The Tool field group attributes indicate the degree of tool development to support each approach. For this reason, these attributes have only been evaluated in the approaches which cite or define a tool. They are as follows:

- *Tool name.* It describes the name of the tool and is a free field.
- *Licenses.* It puts forward information about tool licenses policies.
- *Availability.* It indicates the possibility of obtaining a copy of the tool from either the Web or the author.
- *Automation degree.* It measures the part of the process that has been implemented in the tool. That is, if the whole process has been implemented in the supporting tool or only a piece of the process is executed automatically.
- *Environment.* It describes the execution environment of the tool: operative system, libraries, etc.

Attributes grouped in *Test Case* describe the results obtained after applying the generation process. They try to answer Q3 and consist of the following:

- *Amount of test cases generated.* It offers a reference to the amount of test cases generated by the approach, depending on the used technique.
- *Format of test cases generated.* It indicates how the test cases are defined.
- *Benefits.* It studies the advantages detected in the generation process according to our own experience.
- *Problems. It* describes the most important disadvantages detected in our evaluation.

Finally, attributes for the element *Documentation,* which answers to question Q4, describe the documentation found about every approach. They are as follows:

- *References to projects.* It indicates if the documentation includes either references to real projects or empirical experiences, in cases where the approach was used.
- *Format.* It describes the format of the available documentation for the approach.
- *Size.* It analyzes the amount of documentation measured by the number of pages about the approach.
- *References.* It lists the most relevant references about each approach.

As it was the aim of this section, with this pattern we offer a homogeneous way to describe the approaches of study. The evaluation of these four elements and their specific attributes offer a wide

Table 3
Characterization Schema for the Approach "Software Requirements and Acceptance Testing".

Attributes	Values
Inputs	Functional requirements
Format of the inputs	Scenario tree (as described in Hsia (1994))
Coverage criterion	State coverage in a state-machine
Notation	Scenario tree and state-machines, both with their own notation described in the references
Systematization degree	High
Test values definition	No
Tool name	Unknown
Amount of test cases generated	Proportional to the number of states in the state-machine
Format of test cases generated	Transitions in the state-machine
Benefits	None in particular or none specifically
Problems	No reference to real projects This approach uses its own notations instead of standard (or widely used) notations No reference about automation issues
References to projects	No
Format	Paper
Size	2 papers, 53 total pages
References	(Hsia, 1994; Hsia, 1997)

vision of each approach, essential for the conclusions presented at the end of the paper.

4. Characterization of approaches

Based on the characterization schema presented in Table 2, this section provides an instance of this schema for each approach numbered in Table 1. At the same time, each approach is presented with its most relevant references and a global description which is completed with its characterization in the common schema.

Nevertheless, a set of approaches of study numbered in Table 1 is not characterized in detail in this section, either because they do not include any new process or idea comparing them with the set of approaches characterized in next paragraphs or because they are not completely focused on the line of the paper. They are briefly presented at the end of this section.

4.1. *Software requirements and acceptance testing*

The approach for Software Requirements and Acceptance Testing, presented in Hsia (1994, 1997), focuses on the generation of acceptance test cases.

However, the scope of this acceptance test case is the same than the scope of functional system test case defined in previous sections, thus it was included in the survey.

The approach is divided into two parts: the first one describes a requirement elicitation process, mainly focused on functional requirements elicitation, with a specific model introduced by the approach based on Scenario tree. They are the input for the second part which describes the generation of test cases from these requirements. Table 3 shows the characterization schema for this approach.

4.2. *Testing object-oriented systems*

The approach presented in Binder (1999) adapts the Category-Partition Method (Ostrand and Balcer, 1988) to a use case in natural language context. The starting point is the group of functional

Table 4
Characterization Schema for the approach "Testing Object-Oriented Systems".

Attributes	Values
Inputs	Functional requirement
Format of the inputs	Templates (the structure of the template is not defined in the approach)
Coverage criterion	All combinations for the operational variables defined
Notation	Tabular text
Systematization degree	Low
Test values definition	Yes
Tool name	Unknown
Amount of test cases generated	Satisfactions of a coverage degree
Format of test cases generated	Description of the values for operational variables
Benefits	This approach offers a clear reference of the starting date of the testing process This approach includes a reference to the IEEE982.1 standard as a way of measuring the coverage of the generated test cases
Problems	The approach describes which information must appear in an extended requirement, although it does not describe any template for the requirements The approach is fully based on natural language, what makes its automation highly difficult The approach does not include references to real applications
References to projects	No
Format	Book
Size	9 pages
References	Binder (1999)

requirements written in natural language extended with operational variables. In this approach, an operational variable is a synonym of a category. For this reason, an operational variable is a variable factor whose value determines which scenario of a use case will be exercised in each moment. For example, if there is a use case with two scenarios for valid and invalid data, it is represented as an operational variable. At the end of the process, a table with all test cases for testing and operational variables is obtained.

Table 4 represents the characterization schema for this approach.

4.3. *Automated test case generation from dynamic models*

This approach, previously explained in Fröhlich and Link (1999, 2000), is divided into two blocks:

a. The first block describes how to translate a functional requirement in natural language into a state-chart diagram.
b. The second block describes how to generate a set of functional test cases from that state-chart diagram.

This second block is based on the language for problems planning called STRIPS (Fikes and Nilsson, 1971). The state-chart is translated into STRIPS operations, which change in the state of the system, and test cases are automatically generated from a planning tool, which supports STRIPS language. Table 5 displays the characterization schema for this approach.

Table 5
Characterization Schema for the approach "Automated Test Case Generation from Dynamic Models".

Attributes	Values
Inputs	Functional requirements
Format of the inputs	Template (this approach does not include the structure of the template, however, it suggests to follow the structure introduced in Cockburn (2000))
Coverage criterion	All states or all transitions
Notation	UML state diagram
Systematizations degree	High
Test values definition	No
Tool name	Any tool supporting STRIPS language
Amount of test cases generated	It depends on the coverage criterion chosen
Format of test cases generated	Test cases are defined with pre and post-conditions and sequence of transitions
Benefits	This process may be applied with the help of any tool supporting STRIPS It includes a guide on how to measure the coverage of the selected test cases The generation of the activity diagram is described in a very systematic way
Problems	The approach indicates how to manage related functional requirements. Nevertheless, this fact implies creating state machines with states which are, in turn, other state machines and this fact raises up the complexity Although it is a very systematic process, the state-machine is not automatically generated and must be performed by hand The translation from the state-chart into STRIPs operation is not automatic. It is not fully systematic due to the approach that allows the user to include supplementary operations, such as the ones related to test values
References to projects	No
Format	Paper
Size	2 papers, 40 total pages
References	Fröhlich and Link (1999, 2000)

4.4. *Testing from use cases using path analysis technique*

The starting point of this approach, proposed by Naresh (2002), is a functional requirement written in natural language and formatted in tabular text. However, this approach does not indicate a template format or rules to define functional requirements, although it uses an ad-hoc example. In its development process, this approach translates a functional requirement into a flow diagram and performs a path coverage technique to generate test cases. The result of this process is a table with all the possible paths of execution extracted from the diagram and expressed in natural language. Table 6 shows the characterization schema for this approach.

4.5. *Generating test cases from use cases*

This process, proposed by Heumann (2002), begins with the definition of functional requirements without defining any specific model to describe them. Nonetheless, the approach lists the minimum elements necessary for a functional requirement to perform the generation process. The result of this approach is a table with the system test cases represented as usage scenarios also in natural language. Table 7 shows the characterization schema for this approach.

Table 6
Characterization Schema for the approach "Testing from Use Cases using Path Analysis Techniques".

Attributes	Values
Inputs	Functional requirements
Format of the inputs	Template. This approach does not indicate any format
Coverage criterion	All paths from a flow diagram
Notation	Own notation for flow diagrams
Systematization degree	Low
Test values definition	No (it is cited in the approach, but it does not describe how to manage tests values)
Tool name	Unknown
Amount of test cases generated	Number of different paths from the flow diagram
Format of test cases generated	Descriptions of the steps from the test case
Benefits	It offers a guide on how to measure paths and how to detect low-value path that may be omitted Includes a step-by-step example
Problems	Many parts of the process are performed using natural language, what makes its automation highly difficult The approach does not indicate how to generate test cases from requirements that depend on other requirements It does not indicate how to select and quantify the attributes used for measuring the relevance of a path
References to projects	No
Format	Paper
Size	1 Paper, 20 total pages
References	Naresh (2002)

4.6. *Quality web systems*

As previous approaches, Dustin (2002) starts the process with the requirement elicitation process that includes functional requirements and some others like feasibility or accessibility requirements. Due to the fact that this approach focuses on Web systems, its process may be easily applied to other types of systems. As described in the approach, it sets out from the functional requirements written in some specific templates. The results are system functional test cases defined in natural language by using a template also specified in the proposal. Table 8 presents its characterization schema.

4.7. *Scenario-based validation and test of software*

SCENT (Ryser and Glinz, 2003) is a scenario-based approach to elicit, validate and verify functional requirements. It also includes a process for generating test cases from the scenarios. For this reason, this approach may be divided into two blocks:

a. The first one describes how to create and define scenarios.
b. The second block explains how to generate test cases.

The generation approach starts from a set of use case scenarios defined in natural language by using a specific template proposed in the approach. The results are a set of test cases to validate scenarios defined in tables and in natural language.

Table 7
Characterization Schema for the approach “Generating test cases from use cases”.

Attributes	Values
Inputs	Functional requirement
Format of the inputs	Text template
Coverage criterion	All scenarios
Notation	Ad-Hoc
Systematization degree	Low
Test values definition	No
Tool name	Unknown
Amount of test cases generated	Number of scenarios from the functional requirement
Format of test cases generated	Textual descriptions of the steps of a test case
Benefits	None specifically
Problems	It is an approach designed to treat the functional requirements independently. It cannot offer dependencies among functional requirements It is very difficult to automate because it is based on natural language To identify all its possible combinations may result a hard task in cases of complex functional requirements
References to projects	No
Format	Paper
Size	1 document, 10 pages
References	Heumann (2002)

This approach is illustrated with a real application in two real projects which are documented in Itschner et al. (1998). Table 9 represents the characterization schema for this approach.

Table 8
Characterization Schema for the approach “Quality Web Systems”.

Attributes	Values
Inputs	Functional requirements
Format of the inputs	Own templates defined by the approach
Coverage criterion	It is a heuristic criteria, an engineer should value if there are enough scenarios and test values
Notation	No
Systematization degree	Medium
Test values definition	Yes
Tool name	Unknown
Amount of test cases generated	It depends on the number of scenarios and test values available
Format of test cases generated	Test cases are defined as a text pattern
Benefits	It proposes a specific template for test cases There is no direct correspondence between a test pattern and a functional requirement. The same test pattern can include a test case referenced to several use cases
Problems	The approach is based on its own definition of functional requirements. It can only be applied if requirements are grouped according to the approach
References to projects	No
Format	Section in a book chapter
Size	13 pages
References	Dustin (2002)

Table 9
Characterization Schema for the approach “SCENT”.

Attributes	Values
Inputs	Functional requirements defined as scenarios according to Ryser and Glinz (2003)
Format of the inputs	Scenarios templates
Coverage criterion	Each node and transition
Notation	State diagrams and dependency diagrams with its own notation
Systematization degree	Low
Test values definition	No
Tool name	Unknown
Amount of test cases generated	Alternative states of a scenario
Format of test cases generated	They are rows in a table
Benefits	It offers practical examples and enterprise applications It enables the verification of the behavior of a functional requirement including its dependencies with other functional requirements
Problems	Although the approach offers a complete definition of scenarios, test generation description is not sufficiently detailed SCENT defines some steps in a very informal way without specific guidelines to apply them. It is quite negative for automatic generation Although it can be automatic, several elements depend on engineers’ experiences and decisions and require a manual process
References to projects	Yes. Itschner et al. (1998)
Format	Paper
Size	2 documents, 123 pages and a specific document with practical examples
References	Ryser and Glinz (2003)

4.8. *Requirement-based testing*

This approach, whose characterization schema is presented in Table 10, was proposed by Mogyorodi (2003) and it is also divided into two parts with 12 steps. The first part (steps from 1 to 4) starts with the formal revision of a set of requirements described in natural language. This revision is executed by a set of experts who check if requirements agree with general objectives. The second one explains how to generate test cases; all requirements are described with a cause–effect diagram and test cases are generated from this diagram (steps 5 and 6). A detailed example is presented in Mogyorodi (2001), although this paper does not offer a specific description about the automation of the processes, which depends on the expert’s experience. These diagrams are translated into a decision table with different combinations which are mixed to select each test case. Later on, from steps 7 to 12, they are reviewed and checked with users. This approach neither defines nor cites any format for the input functional requirements. It results in a set of functional test cases in natural language. A test case is composed of a set of causes and its subsequent effects.

This approach offers a tool named CaliberRBT, although it is not available for downloaded.

4.9. *Traceability from use cases to test cases*

This approach, proposed by IBM, is not in fact an approach. This paper presents a tool, named IBM Rational RequisitePro, which defines a set of functional requirements, presented as use cases and described by means of activity diagrams and natural languages. This tool also defines test cases and offers a set of traceability matrix

Table 10
Characterization Schema for the approach "Requirements-based testing".

Attributes	Values
Inputs	Functional Requirements in natural language
Format of the inputs	No
Coverage criterion	Other (coverage of a cause/effect diagram)
Notation	Cause/effect diagram
Systematization degree	Medium
Test values definition	No
Tool name	CaliberRBT
License	Proprietary
Availability	No
Automation degree	Partial
Environment	Unknown
Amount of test cases generated	Unknown
Format of test cases generated	Natural language
Benefits	It offers a support tool
Problems	This approach provides few details in the generation process. Only 2 out of the 12 steps of the process are related with the generation Its documentation is not specifically explained and its steps are not described in detail There are not references to real projects There is no clear justification about the use of cause/effect diagrams, because decision tables offer the same information The approach does not consider how dependencies among functional requirements must be treated
References to projects	No, but it includes a detailed example
Format	Paper
Size	3 work, 22 total pages
References	Mogyorodi (2001, 2002, 2003)

to connect each scenario of a functional requirement with its test cases. Despite the fact that it is not an approach (Zielczynski, 2006), it was included on the paper because it offers a detailed explanation about the definition of different scenarios from a use case. It also adds how to generate, manually, test cases from them. Table 11 presents its characterization schema.

4.10. RETNA: from requirements to testing in natural way

The approach "RETNA", proposed in Boddu et al. (2004), is divided into two blocks:

a. The first block describes a natural language analyzer. This analyzer is used for the study of functional requirements written in a paragraph no longer than 399 characters. After the analysis, the process continues with the elaboration of a state machine which is obtained in MONA language.
b. The second block of the approach describes how to generate functional test cases from the state machine. It introduces several constraints with the natural language analyzer. In fact, no tabular text or template may be used.

Table 12 presents its characterization schema.

4.11. A UML-based test generation and execution

Once again, this approach presented by Ruder (2004), starts with functional requirements written in natural language. The result is a set of functional test cases obtained from a coverage criterion based on combinations that support boolean propositions defined in Test Specification Language (TSL) (Balcer et al., 1990). TSL is a language used for writing format test specifications of the functions of a software system. This language defines the Category-Partition Method in Ostrand's paper.

Table 11
Characterization Schema for the approach "Traceability from Use Cases to Test Cases".

Attributes	Values
Inputs	Functional Requirements in RequisitePro
Format of the inputs	Scenarios in RequisitePro
Coverage criterion	Coverage of Scenarios
Notation	Activity diagrams
Systematization degree	Low
Test values definition	Yes
Tool name	RequisitePro
License	Proprietary
Availability	No
Automation degree	Partial
Environment	Microsoft Windows Operating System
Amount of test cases generated	Depends on the tester selection
Format of test cases generated	Scenarios in RequisitePro
Benefits	It offers a support tool It provides a commercial solution used in companies
Problems	It is not an approach It is completely closed to RequisitePro The test case generation depends on the tester and it is completely manual
References to projects	No
Format	Paper and handbooks of the tool
Size	A 21-page paper and handbooks
References	Zielczynski (2006)

Table 12
Characterization Schema for the approach "RETNA".

Attributes	Values
Inputs	Functional requirements described with a single paragraph no longer than 399 characters
Format of the inputs	No
Coverage criterion	Scenarios (coverage of a state diagram)
Notation	State-diagram (MONA notation)
Systematization degree	High
Test values definition	No
Tool name	No (it is a set of scripts in different languages that using different libraries
License	Proprietary
Availability	No
Automation degree	Total
Environment	Unknown
Amount of test cases generated	Coverage criteria up in the state diagram
Format of test cases generated	It is not detailed in the approach
Benefits	The process to obtain a set of test cases from functional requirements in natural language is completely automatic
Problems	Obtained results are not detailed enough and there are no examples on how state diagrams are obtained It is based on a natural language analyzer, thus, it can only be used with requirements written in English
References to projects	No
Format	Paper
Size	9 pages
References	Boddu et al. (2004)

Table 13
Characterization Schema for the approach "A UML-based test generation and execution".

Attributes	Values
Inputs	Requirements written in natural language
Format of the inputs	Functional requirements with tabular structure
Coverage criterion	Other (combination that support boolean propositions defined by TSL Language)
Notation	Activity diagrams described with stereotypes, categories and partitions
Systematization degree	Medium
Test values definition	Yes
Tool name	TDE-UML
License	Proprietary
Availability	No
Automation degree	Medium
Environment	Unknown
Amount of test cases generated	Depends on different alternatives and paths in the activity diagram
Format of test cases generated	Paths and a set of partitions (they can be defined with a set of test scripts)
Benefits	The use of stereotypes permits to add information to automate the test generation process under UML rules It groups two techniques to test definition
Problems	It focuses on graphical interfaces. Consequently, it does not permit test generation for other kind of external interfaces of a system, for example, two systems communication It does not describe the in-depth process of building activity diagrams It does not clarify whether each functional requirement needs an activity diagram or only one activity diagram is required for the whole system We could not find an available version of the tool to test it
References to projects	No
Format	Paper
Size	1 papers, 20 pages
References	Ruder (2004)

The approach claims that these test cases may be executed in tools that interact with graphical interfaces, which allows to automate test cases execution. They use a set of heterogeneous tools and scripts ad-hoc that are not now available on the Web. Some of them are: JTrek to test oracles definition, MONA languages and a tool to translate formulae into state-machines.

Table 13 shows the characterization schema for this approach.

4.12. An automatic tool for generating test cases from the system's requirements

One of the most relevant aspects of this approach included in Ibrahim et al. (2007), Ismail et al. (2007) is that it introduces a tool, called GenTCase, for generating test cases automatically. The process starts with the definition of use case diagram which is supported by the tool. Every use case may be completed with some event flow that contains a tabular description of the steps of a use case in natural language, including preconditions, post-conditions, alternative steps and a UML sequence diagram. After the analysis of different paths, the generation results in a text file with the generated test cases. Due to the references found, the tool is described instead of the process, thus it may be concluded that the process is the tool in itself. Table 14 shows the characterization schema for this approach.

Table 14
Characterization Schema for the approach "An automatic tool for generation test cases from the system's requirements".

Attributes	Values
Inputs	None. Requirements are defined with the own tool
Format of the inputs	Not applicable
Coverage criterion	A combination of main steps and alternative steps
Notation	UML functional requirements diagrams like use cases, sequences diagrams and tabular text
Systematization degree	High
Test values definition	No
Tool name	GenTCase
License	Unknown
Availability	No
Automation degree	Total
Environment	Microsoft Windows Operating System
Amount of test cases generated	Depends on alterative steps
Format of test cases generated	Textual description of test steps
Benefits	A tool to support the process
Problems	It does not offer a theoretical base, the approach is the tool It could be redundant because an execution flow and a sequence diagram have to be defined at the same time. In references, the use of this sequence diagram for test generation is not clear
References to projects	No
Format	2 Papers
Size	20 Total pages
References	Ibrahim et al. (2007), Ismail et al. (2007)

4.13. Test cases generation from use cases

This approach represents in our study a general tendency to the use of models in the test phase. This tendency, named model-based testing (MBT) (Legeard and Model-based Testing:, 2010), has been widely used in different areas of software testing. For instance, the approach MODEST (Rutherford and Wolf, 2003) starts with a domain model, described in XML, and generates a test code. Our survey includes three approaches on MBT environment quite close to the survey aims. Nevertheless, the only study in detail is the one presented by Gutierrez et al.; the two left are described in the next section.

Gutierrez et al.'s approach enriches the MBT with the use of the model-driven paradigm for the systematic generation of test cases from functional requirements represented as use cases. The approach is based on the UML Testing profile for this aim and is supported by a set of tools. The metamodels used are described in Gutiérrez et al. (2008a), but transformations are not included in edited papers, with the exception of a part of them that appears in Gutiérrez et al. (2008b). However, the approach is included in a free-available tool named NDT-Suite. NDT-Suite is a tool case designed to support a methodology, named NDT (Navigational Development Techniques) (Escalona and Aragón, 2008), which is led to Web Engineering. This methodology is being widely used in real projects. Besides, the approach presents an automatic generation of test cases from use cases described by means of scenarios or activity diagrams. In www.iwt2.org a video with its use is available to download. Table 15 introduces its characterization schema.

Table 15
Characterization Schema for the approach "Test cases generation from use cases".

Attributes	Values
Inputs	Requirements described either as a specific linguistic pattern or an activity diagram
Format of the inputs	It offers its own linguistic patterns or a specific extension of UML activity diagrams to describe functional requirements
Coverage criterion	A combination of main steps and alternative steps
Notation	UML functional requirements diagrams like use cases, sequences diagrams and tabular text
Systematization degree	High
Test values definition	Yes
Tool name	NDT-Suite
License	Free, although the Website defines the profile over Enterprise Architect, which is a non-free tool
Availability	Yes
Automation degree	Total
Environment	Microsoft Windows Operating System
Amount of test cases generated	Depends on alternative steps
Format of test cases generated	Textual description of test steps or graphical presentation as an UML diagram
Benefits	It is a tool to support the process It is used by several companies It is integrated in a framework for the development of Web systems It offers a large number of examples, videos, etc
Problems	Transformations are not completely published Its documentation is mainly in Spanish Data to be put to the test must be included manually
References to projects	Yes
Format	2 Papers and a Website to support
Size	12 Total pages
References	Gutiérrez et al. (2008a,b)

4.14. *Other related approaches*

As introduced at the beginning, there are several approaches in Table 1 that are not described in detail in previous sections. It mainly happened for two reasons:a. On one hand, some approaches are either quite similar to the newest one or they are extended by the newest approaches. This is the case of the following three:

- First, the book *Writing Effective Use Cases* (Cockburn, 2000) includes a chapter about generation of functional system test cases from functional requirements. This chapter proposes identifying every possible scenario from the use case and using them as test cases. This strategy is widely applied in the characterized approaches (for example in Heumann (2002) and Naresh (2002)) and the inclusion of this approach does not offer any new visions with regard to the newest one, which offers other new ideas apart from the scenario identification.
- Second, the approach named in Table 1 as *Test Cases Generation from UML State Diagrams* included in Kim et al. (1999) applied ideas quite similar to other approaches in the survey. From state diagrams, it generates a set of test cases that cover a set of paths of the state diagram. There are some current approaches that also offer this solution.
- Finally, the work by Wood and Reis (*Use Cased derived Test Cases*) (Wood and Reis, 2002) also sets out the same strategy to generate test cases and, once again, it does not encompass more information than other similar approaches.

b. On the other hand, another reason for not describing an approach deeply is that it does not completely match with the aims of the paper. Some of them are listed below:

- The Category-Partition Method (Ostrand and Balcer, 1988) does not match with the aims of this paper, even though it was included because several of the presented approaches (as it was previously introduced in Section 4) use it for its generation methods.
- The next approach chronologically presented is *TOTEM* (Labiche and Briand, 2002) which is organized into two parts:

a. The first part describes how to generate test cases with exercise sequences of functional requirements by using parameters for the functional requirements in a similar manner to Requirement Based Contract.
b. The second part generates test cases for each functional requirement. However, this approach starts from UML sequence diagrams instead of text templates.

- After TOTEM, the next approach is defined in Riebisch et al. (2002) and Huebner et al. (2003) (*Statistical Usage Testing Based on UML*). Both works describe a process to generate statistical test cases from functional requirements defined as tabular text. Although statistical test cases may also be included in a system test level, they are different from functional test cases. The main objective of a statistical test case is not to identify the right implementation of a functional requirement, but to verify the feasibility defined in the requirements. The process begins with creating a state-chart diagram annotating each transition with a percentage that indicates the probability of execution of that transition. For example, if an S state has two outcoming transitions, S1 with 30% and S2 with 70%, this means that 3 out of 10 times that this state is reached, the transition S1 will be fired, and 7 out of 10 times, the transition S2 will be fired. Finally, test cases are defined with random paths over the state machine. According to this representative example, if 100 test cases are randomly generated, about 30 of them should execute the transition S1 and 70 of them should execute the transition S2 approximately.
- The approach *Product Lines Use case Test Optimization* (PLUTO), proposed by (Bertolino and Gnesi, 2003) and (Hsia, 1994), sets out a process to generate test cases for the functional requirements of product lines. This approach starts with functional requirements for products lines as described in Bertolino (2002). The notation for the functional requirements is the approach stated in the book (Cockburn, 2000) plus a new element called variability point, which describes the concrete characteristics of each product in the family. The result of this approach is both, a set of test cases common to all the products in the family and a different set of test cases specific for each product in the family. These test cases are described through templates in natural language and defined within the approach.
- PLUTO is another application of the Category-Partition Method, similar to other characterized approaches like (Binder, 1999) in Section 4.2. The main difference of PLUTO is that the categories are the variability point of the product family.
- A quite younger approach considered in this list is *Automatic Test Generation: A Use Case Driven Approach*, proposed by Nebut and Fleurey (2003), Nebut et al. (2006). This approach is divided into two parts:

1. The first one describes how to extend the UML use cases with contracts which are defined in a logic propositional language

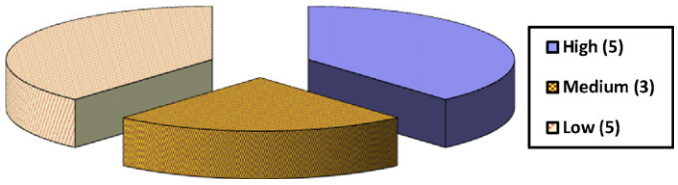

Fig. 1. Systematization degree.

determined in the own approach. The language is used to define preconditions, post-conditions and input parameters for the use case.
2. The second part describes the way to generate test cases from the extended UML use cases in an automatic way. Reference (Nebut and Fleurey, 2003) describes the application of this approach to product families and in Nebut et al. (2006) this approach is extended to generate test cases based on the behavior of each functional requirement.

It is similar to TOTEM because neither TOTEM nor Nebut's approach include objectives oriented to test cases generation from functional requirements. Both generate test cases from correct execution sequences in use cases, and, even though both of them are interesting, they are not close to the studied approaches.

- Another relevant approach within this group is REED (Oh et al., 2008) which generates test cases from a single requirement. Nevertheless, it is not included on the paper because, in fact, it is not an approach, but a tool to represent requirements in a specific notation and generate test case. The tool is close to this own representation and it does not focus on functional requirements.

In the environment of the model-driven testing, two approaches were referenced in Fig. 1, but they are not included in the paper either:

- The first one, by Pérez et al. (2009), is not widely described because it completely focuses on product lines. It uses the model-driven paradigm for the systematic generation of test cases in product lines described with a dynamic model. Despite the fact that some ideas of the approach, like the systematic generation of test cases, are closely linked to the final goal of this paper, it is not focused on functional requirements.
- The second one, by Robles et al. (2009), is a model-driven approach that begins with a set of requirements, described by means of some mockups, generates user interfaces from them and, simultaneously, a set of tests to prove them. The approach uses XML to describe the system.

The use of the model-driven engineering in test generation is a new tendency that is solving the automatic, or, at least, systematic generation of test cases, as it is presented in final conclusions.

5. Analysis

With the previous overview of the state of art and with the definition of the characterization schema for each approach, this section points out the level of maturity in the generation of system test cases from functional requirements.

5.1. Test generation strategy

A first aspect to be taken into consideration to group approaches that may be extracted from the existing ones is the test generation strategy. There are two clear strategies: scenario analysis and test value analysis. With a total of 13 characterized approaches, 11 of them work with test value strategy, mainly based on the Category-Partition method. There is only 1 that is dealing with scenario analysis, and another one that supports both strategies. Table 16 provides that information.

Table 16
Test generation strategy in each approach.

Approach	Category-Partition method	Scenario Analysis
Software Requirements and Acceptance Testing (1997)		1
Testing Object-Oriented Systems (1999)	1	
Automated Test Case Generation from Dynamic Models (2000)	1	
Testing From Use Cases Using Path Analysis Technique (2002)	1	
Generating Test Cases From Use Cases (2002)	1	
Quality Web Systems (2002)	1	
SCENT (2003)	1	
Requirement Based Testing (2003)	1	
Traceability from Use Case to Test Cases (2003)	1	
RETNA (2004)	1	
A UML Based Test Generation and Execution (2004)	1	1
An Automatic Tool for Generating Test Cases from the System's Requirements (2007)	1	
Test Cases Generation from Use Cases (2007)	1	
Total	12	2

As observed in the table, most of the approaches work with the identification of usage scenarios from a functional requirement. This identification may be performed either directly over the functional requirement or by representing the functional requirement on a diagram (for example a state-chart diagram) firstly and, then, by using a coverage criterion over the diagram, like all states, all transitions, etc. There are few of them that use the analysis of test cases (based on the Category-Partition method, aforementioned). Only one of the approaches (*A UML Based test generation and Execution*) uses both strategies.

5.2. *The grade of systematization*

Another relevant attribute is the systematization degree of the approaches. Fig. 1 shows an abstract of the automation degree.

Functional requirements in natural languages are generic and informal artifacts. However, Fig. 1 confirms the possibility of developing a systematic and precise set of steps or tasks to generate functional test cases. Specifically, five out of all the analyzed approaches have become highly systematized. It may also be relevant when comparing the degree of systematization with the existence of a supporting tool for the approaches analyzed. Table 17 enumerates a list of the systematization degree and the inclusion of a tool obtained from these fields in the characterization schema.

As can be concluded, columns 2 and 3 are a mismatch within Table 17. Some of the approaches with a high degree of systematization do not offer a tool support. They are analyzed with the next factor.

It is important to stick out the lack of free and wide availability of the supporting tools in general in the approaches, as it can be deduced from their characterization tables. As aforementioned in Section 5.3, this tendency of changing approaches oriented to MBT was based on UML profiles. The use of UML-based tool is opening

Table 17
Systematization degree and tools.

Approach	Systematization	Tool
Software Requirements and Acceptance Testing	High	No
Testing Object-Oriented Systems	Low	No
Automated Test Case Generation from Dynamic Models	High	Yes (Partial)
Testing From Use Cases Using Path Analysis Technique	Low	No
Generating Test Cases From Use Cases	Low	No
Quality Web Systems	Medium	No
SCENT	Low	No
Requirement Based Testing	Medium	Yes (partial)
Traceability from Use Case to Test Cases	Low	Yes (partial)
RETNA	High	Yes (total)
A UML Based Test Generation and Execution	Medium	Yes (partial)
An Automatic Tool for Generating Test Cases from the System's Requirements	High	Yes (total)
Test Cases Generation from Use Cases	High	Yes (total)

a new path to offer suitable tools in order to support test cases definition and connection.

5.3. *Functional requirements models*

Closer to the systematization degree is the fact that some approaches work directly with functional requirement defined as use cases in natural language, but other approaches build a more formal model, as a first phase for generating functional test cases. A brief list on these approaches may be found in Table 18. The first row presents approaches that deal with requirements in natural languages and the second one includes approaches that generate an intermediate model. This information is directly obtained from "*format of the inputs*" in the characterization schema.

Looking at the previous table, it may be concluded that a high number of approaches develop a model instead of directly working with the functional requirements in natural language. Even more, all the approaches which have a high degree of systematization (Fig. 1) include a formalization of the requirements into some kind of more formal model. As it has been checked in characterization schemas, the step for generating a more formal model from a functional requirement in natural text may also be described with a high degree of systematization. As a complement to Table 18, Table 19 lists the notations and languages used in the analyzed approaches for generating a more formal representation of the requirements. This detail is not implicitly included in the characterization schema but it is mentioned in the description of each approach in the survey.

It may be observed that there is a wide range of notations. However, state-machines, activity diagrams and flow diagrams are quite similar. In fact, the whole UML introduces the activity diagram as a specialization of a state-chart and they both share many elements and semantics. Thus, the state diagram notation is then the notation frequently used, since seven of the approaches make use of it or of a similar one, for that matter. The three remaining approaches use different notations systems among them and only one, out of these three, uses a notation defined by UML.

Table 18
The use of formal models for requirements in test generation process.

	Totals
Approaches that deal with requirements without an intermediate formal model	3
Approaches that deal with requirements of an intermediate formal model	10

Table 19
Notation used for formal models in requirements.

Approach	Notation
Software Requirements and Acceptance Testing	Tree scenarios
Automated Test Case Generation from Dynamic Models	State diagrams
Testing From Use Cases Using Path Analysis Technique	Execution flow diagrams
SCENT	State diagrams
Requirement Based Testing	Cause/Effect diagrams
Traceability from Use Case to Test Cases	Activity diagrams
RETNA	State diagrams
A UML Based Test Generation and Execution	Activity diagrams
An Automatic Tool for Generating Test Cases from the System's Requirements	Sequence diagrams
Test Cases Generation from Use Cases	Activity diagrams

The tendency to the use of UML diagrams, like activity diagrams, is increasing in the last years, due, mainly, to the use of model-driven paradigm. The possibility of defining profiles, like the UML-Testing profile, is making the new approach use formal extensions of UML to define their solutions. For instance, the last approach shown in Table 19, defines an extension of UML included in an UML-based tool to support the development. Besides, the use of standard models, like UML activity diagrams, assures that the approach will be more understandable for the development team because they are well-known by the software community.

5.4. *Notation for functional requirements inputs and test cases*

Another relevant aspect to consider is the disparity at the time of defining the functional requirements which are the source for the generation process. Table 20 shows the analyzed approaches and the notation used for the input requirements.

If the inputs (functional requirements) have a wide notation variety in the analyzed approaches, the outputs (functional test cases) are also defined in a wide range of formats and notations. Table 21 represents a brief summary on how each approach defines a functional system test case.

With regard to Table 21, it can be concluded that few of the approaches define a test case as a set of test steps, plus their input values and a set of constraints to validate the test case. This means that an additional work to include all lost information into the test cases is needed.

6. Conclusions and future work

This paper offers an overview about the situation of test cases generation from functional requirements. It starts with the procedure defined by SEG (2007) to follow a defined process when facing up this overview. This paper uses the terminology proposed by Brereton et al. (2007) in order to specify the strategy used in the development of the survey.

The research begins with the analysis of previous surveys and comparatives studies as well as the definition of the scope of the work.

After selecting the approaches of study, Table 2 represents a characterization common schema to describe each of them. The overview is presented with a briefly introduction of every approaches and their characterization schema samples. Section 5 puts forward a set of comparative analysis with relevant conclusions about the present day situation.

A set of conclusions of this survey may be summarized from the previous works analyzed in Section 2.

Table 20
Notation for functional requirements input.

Approach	Tree Scenarios	Test patterns			Text without structure
		Without specific notation	With a specific notation	Implemented on the tool	
Software Requirements and Acceptance Testing	X				
Testing Object-Oriented Systems		X			
Automated Test Case Generation from Dynamic Models		X			
Testing From Use Cases by Using Path Analysis Technique		X			
Generating Test Cases From Use Cases		X			
Quality Web Systems			X		
SCENT			X		
Requirement Based Testing					X
Traceability from Use Case to Test Cases				X	
RETNA					X
A UML Based Test Generation and Execution		X			
An Automatic Tool for Generating Test Cases from the System's Requirements				X	
Test Cases Generation from Use Cases			X	X	

First, the conclusions from Denger and Medina's survey indicate that the authors of the approaches and processes do not follow any standards for the definition of templates (for functional requirements). On the contrary, each approach uses its own templates and format. Therefore, this conclusion is still valid, as it was mentioned in the previous section.

Another idea from Denger and Medina's report is that none of the approaches uses path analysis techniques and, as a previous step, the approaches build a more formal representation of the functional requirements. Once again, this conclusion is still appropriate.

According to Gutiérrez et al. (2006) one of the conclusions from the previous survey is that many of the existing approaches have to formalize the requirements as a first step to generate functional test cases. Again, this conclusion is still valid. In fact, this is a mandatory step in the approaches which offer a high degree of systematization and have supporting tools. However, it can be pointed out that some approaches, as it is presented in Gutiérrez et al. (2008b) offer a systematic way or even automatic ways (for instance with NDT-Suite) to generate more formal models to automate the process. In this case, this is possible because, although requirements are described in natural language, they are metamodels and some transformations from this description permit to translate requirements in natural language into activity diagrams.

As a conclusion, we can observe that three know surveys have similar results. This state of the art indicates that, there is not a definitive approach that closes the problem of generating functional text cases automatically in a satisfactory way. In fact, it implies a lack of evolution among the existing approaches. Thus, there are some aspects that may be improved, like the use of standards for the inputs and outputs, the application of the standards and more formal methods to describe the process itself, the need for empirical results, the measurement of the possible automation and a profitable tool supporting, etc.

One of the most recent ideas carried out in the last years by several authors is the use of the model-driven paradigm (MDE) in test generation (Brambilla et al., 2009; Hartman and Nagin, 2003). MDE is a new paradigm that focuses on defining a set of metamodels, which are instanced in the development process, and a set of transformations among them. Consequently, in the last year, some related works are emerging in this line. Thus, MDT (Model-Driven Testing) is providing important results in the research (Heckel and Lohmann, 2003; Kuliamin et al., 2003).

When reviewing the literature about MDT as a solution for our research question, we found a high number of papers, nevertheless only one is included with a detailed description and two others are briefly mentioned. The reason is that, despite the high number of MDT approaches, they do not match with the aim of the sur-

Table 21
Notation for test case.

Approach	Test case notation
Software Requirements and Acceptance Testing	Path in a state diagram
Testing Object-Oriented Systems	Rows in a table (a set of values in operational variables)
Automated Test Case Generation from Dynamic Models	Transition Sequence
Testing From Use Cases Using Path Analysis Technique	Flow diagram paths
Generating Test Cases From Use Cases	Text patterns
Quality Web Systems	Text patterns
SCENT	Path in a state diagram
Requirement Based Testing	Natural language
Traceability from Use Case to Test Cases (2003)	Path in a state diagram
RETNA	Path in a state diagram
A UML Based Test Generation and Execution	Path in a activity diagram
An Automatic Tool for Generating Test Cases from the System's Requirements	Text patterns
Test Cases Generation from Use Cases	Text patterns

vey. Thus, we only include several general references in this area because they are offering quite suitable results.

In fact, an added conclusion of the survey that guides to future works is the use of this paradigm when improving test phase. As commented in Section 5.3, this paradigm could also be useful in the coverage of other aspect that need to be improved; the definition of tools in order to support the application of each method.

Finally, we highly recommend as a basic need for future works the real application of the approaches. There are very few practical references, thus, a final feeling could be that, in the research area, we are promoting new advances in the improvement of testing but, are we sure that they are useful in the enterprise environment? If we try to apply some of the approaches presented in this paper, mainly those that do not offer any tool support, we could conclude that it is impossible when we have a high number of functional requirements, which is the most common situation in the enterprise environment.

Research groups must direct their approach to the enterprise in order to offer a suitable answer to test software.

Acknowledgments

This research has been supported by the project QSimTest (TIN2007-67843-C06_03) and by the Tempros project (TIN2010-20057-C03-02) of the Ministry of Education and Science, Spain.

References

Balcer, M., Hasling, W., Ostrand, T., 1990. Automatic generation of test scripts from formal test specifications. In: Proceedings of ACM SIGSOFT'89 – Third Symposium on Software Testing, Verification, and Analysis (TAVS-3) ,. ACM Press, pp. 257–271.

Bertolino, A., Fantechi, A., Gnesi, S., Lami, G., Maccari, A., 2002. Use Case Description of Requirements for Product Lines, REPL'02, Essen, Germany, Avaya Labs Tech. Re ALR-2002-033.

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

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

Binder, R.V., 1999. Testing Object-Oriented Systems. Addison Wesley.

Boddu, R., Guo, L., Mukhopadhyay, S., 2004. RETNA: from requirements to testing in natural way. In: 12th IEEE International Requirements Engineering RE'04.

Brambilla, M., Fraternali, P., Tisi, M., 2009. A transformation framework to bridge domain specific languages to MDA. In: 4th Workshop on Model-Driven Web Engineering. LNCS, vol. 5421 , France, pp. 167–181.

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, 2007. Lessons from applying the systematic literature review process within the software engineering domain. The Journal of System and Software 80, 271–583.

Cockburn, A., 2000. Writing Effective Use Cases, 1st ed. Addison-Wesley, USA.

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

Dustin, E., 2002. Quality Web Systems, 1st ed. Addison-Wesley, USA.

Escalona, M.J., Aragón, G., 2008. NDT: a model-driven approach for web requirements. IEEE Transactions on Software Engineering 34 (3), 370–390.

Fikes, R.E., Nilsson, N.J., 1971. STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence 2.

Fröhlich, P., Link, J., 1999. Modelling dynamic behaviour based on use cases. In: Proceedings of Quality Week Europe , Brussels.

Fröhlich, P., Link, J., 2000. Automated test case generation from dynamic models. ECOOP, 472–491, 2000.

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

Gutiérrez, J.J., Escalona, M.J., Mejías, M., Torres, J., 2006. Generation of test cases from functional requirements A survey. In: 4th Workshop on System Testing and Validation , Potsdam, Germany.

Gutiérrez, J.J., Escalona, M.J., Mejías, M., Torres, J., Torres-Zenteno, A.H., 2008a. A Case Study for generating test cases from use cases. In: Proceedings of RCIS , Morocco, pp. 223–228.

Gutiérrez, J.J., Nebut, C., Escalona, M.J., Mejías, M., Ramos, I., 2008b. Visualization of use cases through automatically generated activity diagrams. In: ACM/IEEE 11th International Conference on Model-Driven Engineering Languages and Systems. LNCS, vol. 5301 , pp. 83–96.

Hartman, A., Nagin, K., 2003. Model driven testing – AGEDIS architecture interfaces and tools. In: 1st European Conference on Model Driven Software Engineering , Nuremberg, Germany, December, pp. 1–11.

Heckel, R., Lohmann, M., 2003. Towards model-driven testing. Electronic Notes in Theoretical Computer Science 82 (6), 1–11.

Heumann, J., 2002. Generating test cases from use cases. Journal of Software Testing Professionals.

Huebner, M., Philippow, I., Riebisch, M., 2003. Statistical usage testing based on UML. In: Proc. of the 7th World Multi Conference on Systemics, Cybernetics and Informatics , pp. 290–295.

Hsia, P., 1994. A usage-model based approach to test the Therac-25. Safety and Reliability in Emerging Control Technologies. IFAC Workshop. Florida, USA, pp. 55–63.

Hsia, P., 1997. Software requirements and acceptance testing. Annals of software Engineering, 291–317.

IEEE 829-2008, 2008. IEEE Standard for Software and System Test Documentation.

Ibrahim, R., Saringat, M., Ibrahim, Z., Ismail, N.N., 2007. An automatic tool for generating test cases from the system's requirements. In: IEEE 7th International Conference on Computer and Information Technology CIT2007 , Fukushima, Japan.

Ismail, N., Ibrahim, R., Ibrahim, N., 2007. Automatic generation of test cases from use-case diagram. In: International Conference on Electrical Engineering and Informatics. Institut Teknologi, Bandung, Indonesia, June 17–19.

Itschner, R., Pommerell, C., Rutishauser, M., 1998. GLAS.S. Remote monitoring of embedded systems in power engineering. IEEE Internet Computing 2 (3), 46–52.

Insfrán, E., Pastor, O., Wieringa, R., 2002. Requirements engineering-based conceptual modelling. Requirements Engineering Journal Vol7 (1.).

Kim, Y.G., Hong, H.S., Cho, S.M., Bae, D.H., Cha, S.D., 1999. Test cases generation from UML state diagrams. IEEE Proceedings-Software 146 (4), 187–192.

Kuliamin, V.V., Petrenko, A.K., Kossatchev, A.S., Bourdonov, I.B., 2003. UniTesK: Model based testing in industrial practice. In: 1st European Conference on Model Driven Software Engineering , Nuremberg, Germany, December, pp. 55–63.

Labiche, Y., Briand, L.C., 2002. A UML-based approach to system testing. Journal of Software and Systems Modelling (SoSyM) 1 (1), 10–42.

Legeard, B., 2010. Model-based testing: next generation functional software testing. In: Dagstuhl Seminar Proceedings 10111. Practical Software Testing: Tool Automation and Human Factors.

Liu, S., Yuting, C., 2008. A relation-based method combining functional and structural testing for test case generation. Journal of Systems and Software 81 (2), 234–248.

Mogyorodi, G., 2001. Requirements-based testing: an overview. In: 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39) , p. 0286.

Mogyorodi, G., 2002. Requirements-based testing: ambiguity reviews. Journal of Software Testing Professionals, 21–24.

Mogyorodi, G., 2003. What is requirements-based testing? In: 15th Annual Software Technology Conference , Salt Lake City, USA, April 28–May 1.

Naresh, A., 2002. Testing from use cases using path analysis technique. In: International Conference On Software Testing Analysis & Review.

Nebut, C., Fleurey, 2003. Requirements by contract allow automated system testing. In: Proceedings of the 14th International Symposium of Software Reliability Engineering (ISSRE'03) , Denver, Colorado, EEUU.

Nebut, C., Fleury, F., Le Traon, Y., Jézéquel, J.M., 2006. Automatic test generation: a use case driven approach. IEEE Transactions on Software Engineering 32 (3).

NDT-Suite, 2008. Available from: www.iwt2.org/ndt (Last visit 2/2011).

Oh, J., Lee, H., Park, H., Kim, J., Choi, K., Jung, K., 2008. A single requirement modelling with graphical language for embedded system. The KIPS Transactions: Part D 15-D (4), 505–512, 2008.

Ostrand, T.J., Balcer, M.J., 1988. Category-partition method. Communications of the ACM, 676–686.

Pérez, B., Polo, M., Piatini, M., 2009. Software product line testing – a systematic review. In: 4th International Conference on Software and Data Technologies (ICSoft 2009).

Riebisch, M., Philippow, I., Götze, Marco, 2002. UML-based Statistical Test Case Generation. Net Object Days, Erfurt, Germany, October 7–10.

Robles, E., Grigera, J., Rossi, G., 2009. Bridging Test and model-driven approaches in web engineering. In: 9th International Conference on Web Engineering. LNCS, vol. 5648 , pp. 130–150.

Ruder, A., 2004. UML-based test generation and execution. In: Rückblick Meeting , Berlin.

Rutherford, M.J., Wolf, A.L., 2003. A case for test-code generation in model-driven systems. In: Proceedings of the Second International Conference on Generative Programming and Component Engineering , pp. 377–396.

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.

SEG (Software Engineering Group), 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering Version 2.3. EBSE Technical Report. EBSE-2007-01. School of Computer Science and Mathematics, Keel University and Department of Computer Science, University of Durham, United Kingdom.

Stocks, P., Carrington, D., 1996. A framework for specification-based testing. IEEE Transaction on Software Engineering 22 (11).

TerMaat, P., 2001. Adventures in automated testing! The Software Testing and Quality Engineering Magazine, May/June.

Vegas, S., Juristo, N., Brasili, V.R., 2009. Maturing software engineering knowledge through classifications: a case study on unit testing techniques. IEEE Transactions on Software Engineering 35 (4), 551–565.

Wood, D., Reis, J., 2002. Use Cased Derived Test Cases. StickyMind. www.stickymind.com.

Zielczynski, P., 2006. Traceability from use cases to test cases. Available from: www.ibm.com/developerworks/rational/library/04/r-3217 (Last visit 02/2011).