Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 79 (2006) 1606–1618
The Journal of Systems and Software
www.elsevier.com/locate/jss

Packaging experiences for improving testing technique selection

Sira Vegas [a,*], Natalia Juristo [a], Victor Basili [b]

[a] Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo, 28660, Boadilla del Monte, Madrid, Spain
[b] Department of Computer Science, University of Maryland, College Park, MD 20742, USA

Received 22 December 2004; received in revised form 24 February 2006; accepted 28 February 2006
Available online 21 April 2006

Abstract

One of the major problems within the software testing area is how to get a suitable set of cases to test a software system. A good set of test cases should assure maximum effectiveness with as few cases as possible. There are now numerous testing techniques available for generating test cases. However, many are never used, while just a few are used over and over again. Testers use little (if any) information about the available techniques, their usefulness and, generally, how suited they are to the project at hand, upon which to base their decision on which testing techniques to use. Using a characterisation schema is one solution for improving testing techniques selection. The schema helps to choose the best-suited techniques for a given project based on relevant information for the purpose of selection, assuring that testers' selections are systematic. However, a characterisation schema is only part of the solution. We have found that a critical aspect for making a good selection is the availability of the necessary information and the sources of information that have to be consulted to access this information. Any organisation wishing to use characterisation schemas to select SE techniques needs to first address the issue of packaging the information that the schema contains.

Keywords: Software testing; Testing techniques; Technique selection; Packaging experiences

1. Introduction

As Harrold (2000) claims, evaluation is a highly important process, as it is directed at assuring software quality. According to Beizer (1990), testing is considered as one of the most costly development processes, sometimes exceeding fifty per cent of total development costs. Software consumers and organisations incur approximately US$50B in losses from defective software each year (RTI, 2000). This estimate suggests an industry-wide deficiency in testing.

One of the factors that influence the quality of testing is the set of test cases used. The generation of test cases is closely linked to the selection of testing techniques. When applied, each technique generates a set of test cases, which will differ from one technique to another since different techniques follow different strategies to generate test cases. Accordingly, a sound selection of testing techniques will mean that a good set of test cases is generated to properly test the software. Therefore, the selection of testing techniques is a critical activity, as the selection of the right techniques for a given project implies carrying out effective testing at a reasonable cost.

The question of which are the right techniques to get the best set of test cases for a given system is a question testers face every time they test a system. How is it answered at present? Testers actually make the selection on the basis of their particular perception of the techniques and situations, which is not necessarily incorrect, but partial (and therefore incomplete). In other engineering disciplines, however, any selection is accompanied by a detailed and complete analysis of the fitness of candidate techniques for the situation in which they are to be applied. For example, the technique to be used to surface a road will be chosen depending on parameters such as the climate in the region

* Corresponding author. Tel.: +34 91 336 6929; fax: +34 91 336 6917.
E-mail addresses: svegas@fi.upm.es (S. Vegas), natalia@fi.upm.es (N. Juristo), basili@cs.umd.edu (V. Basili).

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2006.02.049

or what type of and how much traffic will use it. Alternatively, the technique to be used to lay the foundations for a house will be chosen depending on what type of land it is sited and how many floors the building is to have, among others. This, unfortunately, does not apply in software engineering (SE) in general, and software testing in particular.

In Vegas et al. (2003), we proposed a solution to improve testers' selections. The proposed approach consists of a list of the relevant parameters (a characterisation schema) to match the testing techniques to the testing situation. The approach is based on the hypothesis that formalising the parameters for testing technique selection would help testers to take into account all the relevant parameters for their selections, as well as a wide variety of techniques and the information concerning the selection parameters for this set of techniques.

Nevertheless, the use of the proposed schema in a real environment has revealed that the selection problem in SE is more complex than we actually thought. Whereas at first the problem appeared to be that the relevant parameters for testing techniques selection were not known, experiences in using the characterisation schema have shown that this problem masked another: the availability of the information for some of the selection parameters.

This article deals with the problem that surfaced after several organisations adopted the characterisation schema as a practice for selecting testing techniques. These problems revolve around the search for the information needed to be able to select testing techniques using the proposed schema; the generation of this information, when it is not available; and as its storage. We also deal with other issues related to the generation and storage of information for a characterisation schema: the changes that we had to make to the schema that we proposed originally to tailor it to the setting where information is missing, and the establishment of formal procedures for schema use and maintenance.

The article has been organised as follows. Section 2 presents testing techniques selection in the ideal case where all the information is available. Sections 3–5 discuss testing techniques selection where there are limitations on the information, as is the case in the real world. More specifically, Section 3 presents the results of industrial partners applying the original characterisation schema. Section 4 focuses on the sources of information for characterising testing techniques. Section 5 presents the characterisation schema use and maintenance procedures. Finally, Section 6 discusses our conclusions.

2. Selecting testing techniques with full information

When we set out to address the problem of systematising testing techniques selection, the most pressing problem was to identify the parameters (items of information) that the characterisation schema should contain. However, it was not that easy to identify these parameters. The principal stumbling block that we encountered during the construction of the testing techniques characterisation schema was therefore how to get such a list of parameters. For want of a testing technique theory to establish what the relevant information for testing techniques selection is, we opted for the approach of accumulating experience. That is, we put together the characterisation schema based not only on what we thought was important information, but also on what information a sizeable number of testers and researchers would like to use to make tuned selections. Our intention is to complete the partial view developers have of the selection problem. The set of parameters was put together in a two-step process:

- An initial set of parameters was gathered by accumulating practitioners' and testing researchers' experiences in the selection of testing techniques. For this purpose, we surveyed a series of testing researchers and practitioners, who were asked what information they thought was relevant for selecting a testing technique. Each new characteristic suggested by an individual that was not yet in the schema was added.

  At the same time as information was gathered, we ran a stability analysis of the set of parameters to decide when to stop gathering information. When the schema remained stable and no new information appeared after interviewing several more individuals, we considered that the set of information gathered was representative enough of the view held generally by practitioners and researchers.
- After this, experts in the testing area inspected the generated set of parameters. Their opinions included issuing judgements about: the suitability of the organisation, the names used to label the set of parameters, the existence of possible redundancies, missing information, etc.

Table 1 shows the final schema we developed, which is detailed in Vegas et al. (2003). To ease practitioners' understanding of the characterisation schema, the parameters have been organised around nine groups, named elements. Similarly, these parameters are organised around three major groups, named levels. The groups and elements make it easier to locate any parameter within the schema, as they represent the concept to which each parameter refers. The levels, elements and parameters appear in the first three columns of Table 1, respectively. The meaning of each parameter, marked with the letter D, and its permitted values, specified by letter V, appears in the description/value column.

We can see from Table 1 that the *tactical* level, related to what is to be tested, consists of these two elements: the purpose or *Objective* of the test and the *Scope* of the test. The *operational* level, related to technique operation consists of five elements: features of the actual *Technique*; results of applying the technique, this is, *Test cases*; the software (*Object*) features on which the technique is to be applied; the *Tools* available for improving technique application; and what characteristics the subject (*Agents*) should have to be considered as qualified to use the technique. Finally,

Table 1
Characterisation schema for testing technique selection

Level	Element	Parameter	Description/values
Tactical	Objective	Purpose	D: Type of evaluation and quality attribute to be tested in the system V: Two values: find defects, assess software
		Type of defects	D: Defect types detected in the system V: (control, assignation, initialisation, etc.)
		Effectiveness	D: What capability the set of cases should have to detect defects V: Percentage
	Scope	Element	D: Elements of the system on which the test acts V: (function, procedure, system, subsystem, etc.)
		Aspect	D: Functionality of the system to be tested V: (communications, database, GUI's, etc.)
Operational	Technique	Comprehensibility	D: Extent to which the technique is easy to understand V: (high, medium, low)
		Cost of application	D: How much effort it takes to apply the technique V: (high, medium, low)
		Inputs	D: Inputs required to apply the technique V: (requirements, code, design, etc.)
		Adequacy criterion	D: Test case generation and stopping rule V: family (data flow, flow control, etc.) and technique (sentence coverage, etc.)
		Test data cost	D: Cost of identifying the test data V: (high, medium, low)
		Dependencies	D: Relationships of one technique with another V: Two values: [technique] and dependency type (should be applied before, after, should never be used with, etc.)
		Repeatability	D: Whether two people generate the same test cases V: (yes, no)
		Sources of information	D: Where to find information about the technique V: (a person, a book, an article, an experiment, etc.)
	Test cases	Completeness	D: Coverage of the adequacy criterion provided by the set of cases V: Percentage
		Precision	D: How many repeated test cases the technique generates V: Percentage
		Number of generated cases	D: Number of cases generated per software size unit V: Formula
	Object	Software type	D: Type of software that can be tested using the technique V: (real time, batch, iterative, expert system, etc.)
		Software architecture	D: Development paradigm to which it is linked V: (call and return, OO, etc.)
		Programming language	D: Programming language with which the technique can be used V: (structured, functional, logical, real time, concurrent, etc.)
		Development method	D: Development method or life cycle to which it is linked V: (prototyping, reuse, waterfall, knowledge-based system, etc.)
		Size	D: Size that the software should have to be able to use the technique V: (number in KLOC)
	Tools	Identifier	D: Name of the tool and the manufacturer V: Two values: [tool name] and [company name]
		Automation	D: Part of the technique automated by the tool V: (flow chart, mutant generation, test case generation, etc.)
		Cost	D: Cost of tool purchase and maintenance V: Two values: [purchase cost] and [maintenance]
		Environment	D: Platform (sw and hw) and programming language with which the tool operates V: Three values: [SW requirements], [HW requirements] and [programming language]
		Support	D: Support provided by the tool manufacturer V: (24-hour hotline, technical assistance, etc.)
	Agents	Experience	D: Knowledge required to be able to apply the technique V: (cyclomatic complexity, flow charts, etc.)
		Knowledge	D: Experience required to be able to apply the technique V: (tool understanding, etc.)

Table 1 (*continued*)

Level	Element	Parameter	Description/values
Historical	Project	Reference projects	D: Earlier projects in which the technique has been used V: [project name]
		Tools used	D: Tools used in earlier projects V: [tool name]
		Personnel	D: Personnel who worked on earlier projects V: [people's names]
	Satisfaction	Opinion	D: General opinion about the technique after having used it V: [sentence or paragraph explaining the opinion]
		Benefits	D: Benefits of using the technique V: [sentence or paragraph explaining the benefits of the technique]
		Problems	D: Problems with using the technique V: [sentence or paragraph explaining the drawbacks of the technique]

the *historical* level, related to experience in using the technique, consists of two elements: earlier *Projects* in which the technique was used and opinion (*Satisfaction*) the technique merits among people who have used it before.

After the schema had been developed, it was evaluated, as detailed in Vegas and Basili (2005). The schema was evaluated by means of an experiment conducted with students from the Technical University of Madrid to check that the schema can be used to make better selections than would be achieved if it were not used.

Appendix A shows an example of three testing techniques instantiated for the developed schema. Ten more instantiated techniques are illustrated in Vegas et al. (2003). The techniques described here are: boundary value analysis and random testing from the family of functional techniques; sentence coverage, decision coverage, path coverage and threads coverage from the family of control-flow techniques; all-c-uses, all-p-uses, all-uses, all-du-paths and all-possible-rendezvous, from the family of data-flow techniques; and standard mutation and selective mutation from the family of mutation techniques.

3. Selecting testing techniques with real information

Having built and run a preliminary evaluation of the characterisation schema, it was applied in practice. A number of software development organisations agreed to incorporate the characterisation schema into routine practice. This was a two-phase experience. During the first phase, the industrial partners instantiated the schema for the testing techniques in which they were interested. In a second phase, they used the schema and the set of instantiated techniques to support selection. At the end of each phase, a meeting was held with each organisation, during which we gathered feedback and discussed possible solutions for the shortcomings that were found after using the schema in real practice.

Because the schema was developed with input from practitioners, we expected practitioners (although they were not the same professionals as interviewed to generate the schema) to be satisfied with the schema. One of the primary difficulties that we came across was that the *information the practitioners would like to have* to support the selection does not necessarily match the *information that they have access to*. During the construction of the characterisation schema, we did not notice that when someone suggested that a piece of information should be taken into account for selection, this did not necessarily mean that this information was accessible. The real world has shown this approach to be naïve. At the end of the industrial evaluation, all practitioners complained about how difficult it was to instantiate the techniques, to find the information needed to fill in the schema and asked us where they could access the information concerning the selection parameters. When we delivered the schema to them, they had assumed that an experienced tester's knowledge of a technique would be quite enough to fill in all the information about this technique.

This new problem led us to analyse, for each schema parameter, what type of knowledge is needed to get its value, what the level of maturity of this knowledge is, as well as what sources of information can be consulted to gather this knowledge and how the knowledge can be extracted from these sources. This study is presented in Section 4.

This information accessibility analysis led us to slightly remodel the characterisation schema presented in Section 2. These changes primarily involve the deletion of those parameters whose value we are unable to generate at the present time. Fig. 1 summarises the changes made to the characterisation schema.

- The *purpose* parameter belonging to the *objective* element at the *tactical* level is removed. For the time being, the schema is being used only for testing techniques, which implies that they all have the same purpose, namely, software defect detection. This means that this is not a selective parameter. Its inclusion will be reconsidered in the future when techniques for evaluating other software quality attributes, like usability or reliability, are added.
- The *completeness* and *precision* parameters belonging to the *test cases* element of the *operational* level have been removed from the schema. They are deleted because, due to the maturity level of testing today, they cannot

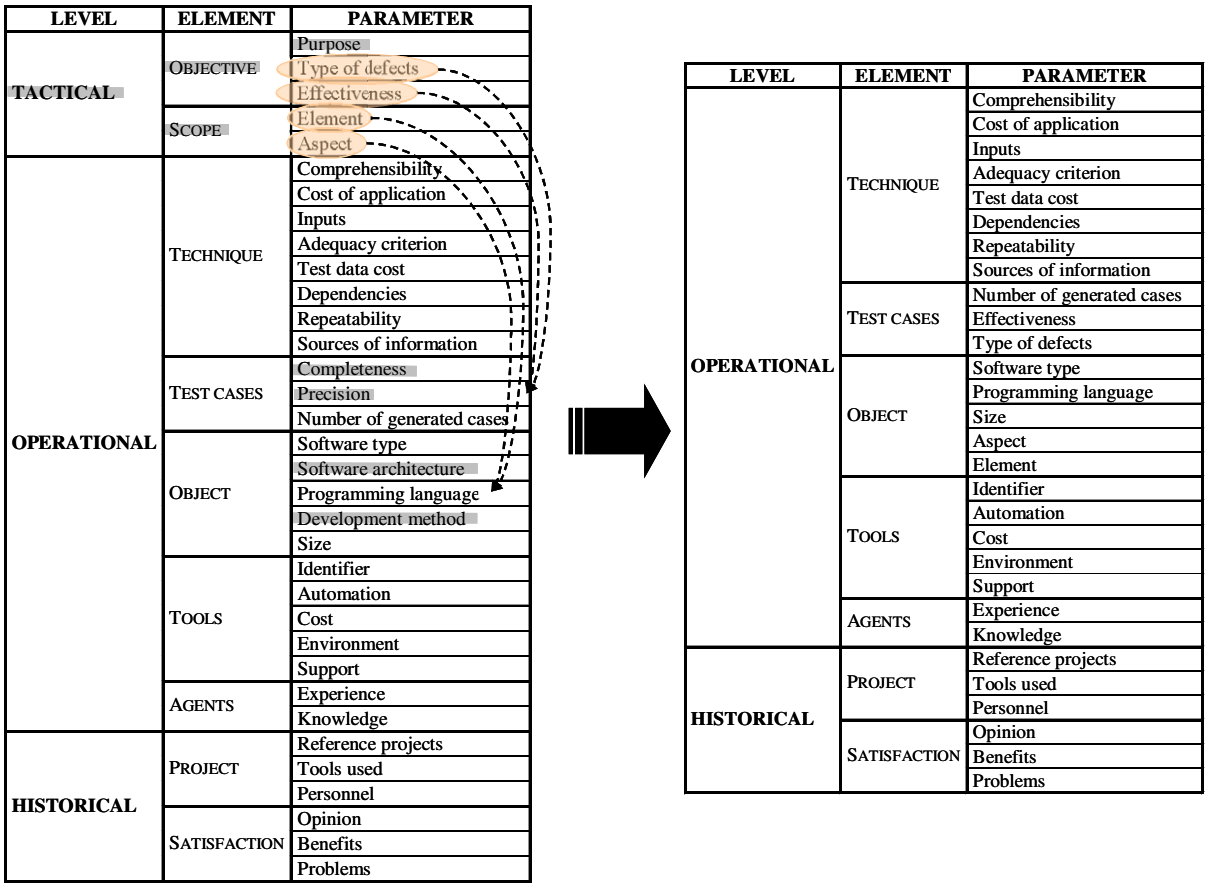

Fig. 1. Evolution of the characterisation schema.

be assigned a value. This information is not known at present; neither textbooks nor research papers provide information on the precision and completeness of the testing techniques. Additionally, there is no research under way to find values for these factors. To be able to gain access to this, new knowledge needs to be discovered on testing techniques. Furthermore, there is not even any information on how to calculate these parameters. Therefore, they are useless (at least for the time being) for selection in real situations. Although *completeness* can be ascertained for a few techniques, such as, for example, white-box techniques, there are other techniques, like mutation, whose *completeness* cannot be ascertained. We may want to generate test cases to kill 100% of the generated mutants. However, sometimes this is not possible. This means that the completeness of the generated test cases is not 100%, but it is not possible to anticipate when this will happen or what it depends on.

- The *software architecture* parameter belonging to the *object* element of *operational* level has been removed, as, from the industrial evaluation, we realised that it has no impact on selection, that is, it is not selective.
- The *development method* parameter belonging to the *object* element of the *operational* level has been deleted, as we realised from the industrial evaluation that the information it provides is redundant with the *software type* element. Therefore, this is a redundancy that the experts in evaluation that built the preliminary schema did not detect.

As regards the structural changes of the schema:

- The *type of defects* and *effectiveness* elements belonging to the *objective* element of the *tactical* level are transferred to the *test cases* element of the *operational* level, as the practitioners who applied the schema appear to prefer them to be there. Analysing this preference, we realised that both elements really do refer to the set of test cases generated by the technique and, therefore, should be placed as parameters of this element.
- The *element* and *aspect* parameters of the *scope* element of *tactical* level have been transferred to the *object* element of the *operational* level, as the practitioners who applied the schema appeared to prefer them to be there. Analysing this preference we realised that both elements refer to the software object that they are designed to test and, therefore, should be placed as parameters of this element.
- As a consequence of the above changes, the *objective* and *scope* elements of the *tactical* level have been removed from the schema, as has the *tactical* level, because they no longer have any parameters.

4. Knowledge types and sources of information for characterising testing techniques

Identifying the information for instantiating a technique is also an important part of the selection problem. The difficulties related to the selection problem lie not only in knowing what is the right information on which to base selection and the most comprehensive universe of techniques but also in finding and generating this information. Table 2 lists, for each characterisation schema parameter, the type of knowledge to which it refers, its maturity level, the source of information from which it can be gathered, and the procedure to be followed to generate the value.

The *knowledge type* column reflects what kind of knowledge the parameter involves. We have identified two types of knowledge: intrinsic to the technique and gained by experience. *Intrinsic* knowledge represents a feature that is inherent to the technique. For example, the *inputs* parameter is a characteristic proper to the technique and which distinguishes it from other techniques. On the other hand, as its very name indicates, the *experience*-type knowledge represents knowledge gained by experience after having used the technique in real projects, as in the case of *tools* used to apply a given technique to a project.

Intrinsic knowledge about a technique can be found at two *maturity* levels: known and reliable, and unknown but under study.[1] The *known and reliable* maturity level means that the parameter value is known (can be found in testing books) and has a reliable value. This is the case of technique *comprehensibility* or the *test data cost*. The *unknown but under study* maturity level implies that, although textbooks provide a value, this value is not reliable and researchers are still working on finding out its real value. This is the case, for example, of the *number of generated test cases.* The unknown but under study values are not generally accepted, but they are being researched and values proposed.[2]

Knowledge with the *known and reliable* maturity level can be gathered from several *sources of information*: testing books, technique application exercises or advertising on testing tools. Information can be gathered from testing books in two different ways: either *directly*, as in the case of information that explicitly appears in testing books, such as technique inputs, or by *deducing* it from the information that appears in the testing books, in which case the information can be deduced from the book contents, although it does not appear explicitly in the text. For example, technique comprehensibility is deduced from the explanations of how to apply the technique. When *exercises* on technique application are the source, the information is gathered from technique application, not necessarily in real projects, but simply by means of an exercise, like the ones that are set in testing books. This applies, for example, to *cost of application*. Finally, this type of knowledge can also be gathered from *advertising* on testing tools. In this case, the information will be gathered directly from this advertising. This would be the case of *tool identifier* or *cost*.

As regards unknown but under study knowledge, that is, knowledge that can now be found in testing books, but whose value is not reliable, values can be assigned in different ways depending on the required reliability level. The simplest way is to use the value provided in textbooks, even though this information is not accurate for these parameters. The next step up is to use values provided by researchers, for which purpose the literature on the subject matter needs to be surveyed. Finally, the third possibility is to use the organisation's in-house data, but organisations need to be highly mature, because they need to systematically collect and later analyse data about each finished project to find these values. It should be noted that this means filing information on the application of testing techniques within the organisation. The ideal thing would be for these data to be recorded globally rather than at the organisational level, but, bearing in mind how far away we are from the global solution, even starting with a part solution is an improvement on the current state of affairs.

Our recommendation for organisations that do not have the resources to invest in data collection and analysis or in searching research papers for information is that they should do without these parameters, as the information available in the literature is not accurate and could impair the selection. On the other hand, we advise organisations that can afford to spend time keeping up with the latest research findings to use the values given in research articles. Finally, organisations that actively collect project data and research these data will have access to the results of earlier projects and be able to calculate the value of these parameters for themselves. Specifically, the values for each parameter of this type listed in Fig. 1 can be calculated as follows:

- *Number of generated cases:* The value of this parameter is calculated by means of a formula, which, depending on the technique, will link the number of generated test cases to certain code characteristics, like the number of sentences (sentence coverage technique), number of decisions (decision coverage technique), input domain complexity (equivalence class partitioning technique), etc. To calculate this value, practitioners should count the number of test cases generated by the technique and relate this to any of the above-mentioned code characteristics.
- *Effectiveness:* To calculate this value, practitioners should record the number of software defects detected by each technique used. The effectiveness of each technique will be output by calculating the percentage of defects found using each technique over the total

[1] Notice that it could also have the unknown and not under study value, but all the parameters with this maturity level have been removed from the schema, as discussed in Section 3.

[2] Maturity is even lower in the case of the disregarded parameters, where there is practically no research into the parameter values, and values cannot even be suggested.

Table 2
Where to find the information for testing techniques selection

Level	Element	Parameter	Knowledge		Source	How to get it
			Type	Maturity level		
Operational	Technique	Comprehensibility	Intrinsic	Known and reliable	Books	Deducible
		Cost of application	Intrinsic	Known and reliable	Exercise	Deducible
		Inputs	Intrinsic	Known and reliable	Books	Direct
		Adequacy criterion	Intrinsic	Known and reliable	Books	Direct
		Test data cost	Intrinsic	Known and reliable	Exercise	Deducible
		Dependencies	Intrinsic	Unknown but under study	Books or projects	Deducible Calculated
		Repeatability	Intrinsic	Known and reliable	Exercise	Deducible
		Sources of information	Intrinsic	Known and reliable	Books	Direct
	Test cases	Number of generated cases	Intrinsic	Unknown but under study	Books or projects or papers	Direct, calculated or researched
		Effectiveness	Intrinsic	Unknown but under study	Projects or papers	Calculated or researched
		Type of defects	Intrinsic	Unknown but under study	Projects or papers	Calculated or researched
	Object	Software type	Intrinsic	Known and reliable	Books	Deducible
		Programming language	Intrinsic	Known and reliable	Books	Deducible
		Size	Intrinsic	Known and reliable	Books	Deducible
		Aspect	Intrinsic	Known and reliable	Books	Deducible
		Element	Intrinsic	Known and reliable	Books	Deducible
	Tools	Identifier	Intrinsic	Known and reliable	Advertising	Direct
		Automation	Intrinsic	Known and reliable	Advertising	Direct
		Cost	Intrinsic	Known and reliable	Advertising	Direct
		Environment	Intrinsic	Known and reliable	Advertising	Direct
		Support	Intrinsic	Known and reliable	Advertising	Direct
	Agents	Experience	Intrinsic	Known and reliable	Books	Deducible
		Knowledge	Intrinsic	Known and reliable	Books	Deducible
Historical	Project	Reference projects	Experience	Amount of historic data	Projects	Direct
		Tools used	Experience	Amount of historic data	Projects	Direct
		Personnel	Experience	Amount of historic data	Projects	Direct
	Satisfaction	Opinion	Experience	Amount of historic data	Projects	Deducible
		Benefits	Experience	Amount of historic data	Projects	Deducible
		Problems	Experience	Amount of historic data	Projects	Deducible

number of defects detected in the software. The total number of defects is calculated by adding the number of defects reported by users during maintenance to the number of defects found during the testing phase.

- *Defect type:* To find out this value, practitioners have to have defined a defect classification. The type of defects that the technique can find is calculated by recording, for each defect detected by the technique, its type.
- *Dependencies:* Practitioners can find out this value by examining the type of defects that each technique detects. They will then be compared to decide which are complementary and which overlap.

On the other hand, *experience*-type knowledge will not have a fixed maturity level. It will vary depending on *how many historical data* the organisation has gathered. Obviously, the more historical data there are, the more reliable the value of this type of parameters will be. From its definition, it follows that the source for this type of parameters is historical information on projects gathered within the organisation. This information can be gathered in two different ways: either *directly*, from the historical data on projects that the organisation has, such as the people who have used the technique in the company before, or can be *deduced* from the use of the technique in real projects, through the feedback provided by practitioners after having used the technique on real projects, such as what they think about the technique.

Like unknown but under study knowledge, experience-type knowledge calls for some degree of organisational maturity. In this case, it implies systematically recording the experience gained in each finished project. Again like unknown but under study information, the problem with this type of knowledge is that today's organisations do not have access to a global record of experiences, which means that, for all intents and purposes, this knowledge cannot be used in real selections unless the organisation systematically collects its own experiences. This means that not all organisations will have this knowledge for use during selection.

Our recommendation is for organisations to try to make the effort to gather these values, as they are neces-

sary for making tuned selections. On the other hand, as we have already seen, this type of knowledge refers to information that will not be available for the first selection, because it is in one way or another based on earlier applications of the technique in the organisation's projects. This means that the solution proposed here will not achieve an optimal selection the first time round, but selections will gradually improve with time, as this type of information is learned.

5. Technique selection assisted by a characterisation schema

5.1. Mechanics of selection

Testers use the characterisation schema proposed here to select testing techniques in a structured and objective rather than a disorganised and subjective manner: matching the attributes of the testing techniques to certain relevant characteristics of the situation or project. More specifically, the selection process would be:

1. *Identification of project characteristics.* First, the situation in which the techniques to be selected are to be applied has to be described. A project descriptor is used for this purpose. The project descriptor is one part of the proposed characterisation that instantiates the parameters that describe the project characteristics by entering the appropriate values. That is, the developer will describe in this step the relevant characteristics of the project in which the techniques are to be applied. This will be done by assigning values to the criteria belonging to the elements: test cases, object, tools and agents. For example, for agent, whose parameters are experience and knowledge, the tester will describe what experience and what knowledge the testers who are to apply the technique have. Ideally, a value should be entered for all these parameters in the project descriptor. The more characterised the project is the better, because the match between the technique and situation will be finer tuned. However, if the information is missing, project description parameters can be left blank in the descriptor. The *project description* rows of Table 3 show the list of parameters making up the project descriptor.
2. *Identification of constraints.* If constraints are to be placed on the use of techniques, for example, if we want to apply techniques that are easy to use or that have already been used at the company, the project descriptor should be added to, specifying values for the schema parameters that match the constraints to be imposed. The *constraints* rows of Table 3 show the list of parameters for project constraints. Note that the sum of the description plus the constraint parameters outputs the complete schema.
3. *Identification of the best-suited techniques.* Values are compared for the criteria that appear in the project descriptor (both description and constraint parameters) with the values that are available for the techniques. This will yield the techniques that match the project conditions.
4. *Analysis of the set of preselected techniques.* When analysing the set of preselected techniques, it may appear that:
   (a) *It is empty.* This means that there is at present no technique in the selection universe used that meets the stipulations in the project descriptor. In this case, the value of one or more parameters that appear in the project descriptor should be relaxed and you should return to step 3.
   (b) *It contains just one technique.* In this case, this will be the technique applied.
   (c) *It contains several techniques.* This means that there are several techniques in the selection universe that meet the project constraints. It will suffice to select one of the techniques that appear, unless the dependencies parameter value says otherwise, in which case the techniques should be applied following the specified dependencies.[3] The choice, if there are no dependencies, will be based on the personal preferences of the selector.

Table 3
Project description and constraints

Type	Element	Parameter	Value
Project description (Enter a value for all parameters for which information is available and preferably all)	**Test cases**	**Number of generated cases**	
		Effectiveness	
		Type of defects	
	Object	**Software type**	
		Programming language	
		Size	
		Aspect	
		Element	
	Tools	**Identifier**	
		Automation	
		Cost	
		Environment	
		Support	
	Agents	**Experience**	
		Knowledge	
Project constraints (Enter a value for parameters on which you wish to place constraints only)	Technique	Comprehensibility	
		Cost of application	
		Inputs	
		Adequacy criterion	
		Test data cost	
		Dependencies	
		Repeatability	
		Sources of information	
	Project	Reference projects	
		Tools used	
		Personnel	
	Satisfaction	Opinion	
		Benefits	
		Problems	

[3] However, care should be taken to examine the dependencies between techniques. As discussed in Section 4, this parameter has an unknown but under study maturity level, and therefore its value is not reliable.

5.2. Example of selection

Now that we have presented the selection process, let us give an example using the characterisation schema. For this purpose, the set of techniques mentioned in Section 2 will be used as the selection universe, alongside the respective information on the specified techniques.

The problem is stated as follows:

> A car park management system (concurrent system) is to be built. At this stage of the project, the QA team has identified the key quality attributes of this software system. These were obtained by examining the characteristics of the software under development, as well as its application domain. In this particular case, the essential attributes are: correctness, security and timing.
> The project situation is as follows: the system is to be coded in Ada, the development team is fairly experienced in developing similar systems, and almost all the errors they make are found to be typical of concurrent programming. The testing team is also experienced in testing this type of systems.

It is solved as follows.

1. *Identification of project characteristics.* Only some of the project descriptor parameters are known, as shown in Table 4.
2. *Identification of project restrictions.* No additional constraints are identified for this project, as shown in Table 4.
3. *Identification of the best-suited techniques.* If the values identified in the previous steps are compared with the three techniques described in Appendix A, we get Table 5. The parameters whose value matches the project are shown in bold. From Table 5, we find that two of the three techniques taken into account match the stipulations of the project descriptor (all the cells entries in the column are in bold), whereas one of them does not (there is one cell entries in the column that is not in bold). This will lead us to disregard this technique because it does not satisfy all the characteristics imposed by the project descriptor.

   If we extend the match to the full selection universe of the 13 techniques mentioned in Section 2, the techniques selected after situation/technique matching are: *boundary value analysis*, *random*, *path coverage*, *all-possible-rendezvous*, *all-c-uses*, *all-p-uses*, *all-uses*, *all-du-paths*, *standard mutation* and *selective mutation*. The *sentence coverage* and *decision coverage* techniques will be rejected because their effectiveness is low, and the technique *threads coverage* will be discarded because it is for object-oriented software.
4. *Analysis of the set of preselected techniques.* Of the preselected techniques, there is one that is specific for Ada-style programming languages. Although there are general-purpose techniques that are more effective, the technique that is specific for concurrent software appears to detect the faults proper to concurrency better than the other techniques. Furthermore, the *path coverage* technique states that when used with concurrent and real-time systems, a dynamic analyser cannot be used as a tool. Additionally, the techniques *all-c-uses*, *all-p-uses*, *all-uses*, *all-du-paths*, *standard mutation* and *selective mutation* cannot be used without a tool (which is not available in the situation under consideration). Therefore, the *all-*

Table 4
Sample project descriptor

Type	Element	Parameter	Value
Project description	**Test cases**	**Number of generated cases**	-
		Effectiveness	>50%
		Type of defects	-
	Object	**Software type**	Real time
		Programming language	Ada (concurrent)
		Size	Medium
		Aspect	ANY
		Element	-
	Tools	**Identifier**	-
		Automation	-
		Cost	-
		Environment	-
		Support	-
	Agents	**Experience**	-
		Knowledge	-
Project constraints	Technique	Comprehensibility	-
		Cost of application	-
		Inputs	-
		Adequacy criterion	-
		Test data cost	-
		Dependencies	-
		Repeatability	-
		Sources of information	-
	Project	Reference projects	-
		Tools used	-
		Personnel	-
	Satisfaction	Opinion	-
		Benefits	-
		Problems	-

Table 5
Matching technique/project characteristics

Parameter	Project	Mutation	Decision coverage	Boundary value analysis
Effectiveness	**>50%**	**Detects approx. 72% of the faults**	Prob. detecting a fault: 48%	**Finds 55% of defects**
Software type	**Real time**	**Any**	**Any**	**Any**
Programming language	**Ada (concurrent)**	**Structured, OO, real time and concurrent**	**Any**	**Any**
Size	**Medium**	**Medium**	**Medium**	**Any**
Aspect	**Any**	**Any**	**Any**	**Any**

possible-rendezvous techniques will be selected. However, the dependency attribute states that the technique should be supplemented with a black-box technique. Observing the black-box techniques in the preselected set (*boundary value analysis* and *random*), it is found that the *random testing* technique is useful for people with experience in the type of tests to be run and will, therefore, also be selected.

5.3. Organisation maturity and schema uses

During the industrial evaluation described in Section 3, we found that not all the practitioners reported the same number of problems when using the schema. In some cases, the schema had worked better than in others. It was found that the number and type of the problems reported was directly related to the company's maturity level, and that companies with a higher maturity level had fewer problems than those with a lower maturity level. Our goal is for all types of organisations and not just those with a high maturity level to be able to take advantage of the schema, although they would gain fewer benefits. Therefore, we have defined two different schema uses, based on the characteristics of the organisation all set to use the schema.

Working with *unknown but under study* and *experience* parameters calls for some organisational maturity in terms of both data collection and testing. Mature organisations collect data from their projects and have a collection of testing techniques described according to the characterisation schema, which is used to compare techniques. On the other hand, immature organisations have no such collection, nor do they collect data of any kind. These two use contexts are described in more detail below.

In the selection of testing techniques by immature organisations, the practitioner uses the parameters contained in the characterisation schema merely as a guide to selection. The proposed schema helps testers by providing support for making a more systematic selection, insofar as testers will always use the same parameters in selection. The schema is used like a guideline or checklist that points developers to what information they should consider to compare the testing techniques. The parameters that will be used here will be the known and reliable maturity level parameters from Table 2 that are listed in Table 6. We understand that, unless the organisation has a mature experience recording process, it is not storing any other type of information, and therefore these organisations are not acquainted with the *unknown but under study* and *experience* parameters.

However, when the characterisation schema is used by immature organisations, the only selection problem solved is related to the selection parameter subjectivity, as this will be governed not by testers' tastes but by a series of objective parameters that have been proven to be suitable. But it does not solve the other selection problems, such as missing information needed for selection purposes and possibly unknown testing techniques that could be relevant for selection.

Table 6
Characterisation schema for immature organisations

Level	Element	Parameter
Operational	Technique	Comprehensibility
		Cost of application
		Inputs
		Adequacy criterion
		Test data cost
		Repeatability
		Sources of information
	Object	Software type
		Programming language
		Size
		Aspect
		Element
	Tools	Identifier
		Automation
		Cost
		Environment
		Support
	Agents	Experience
		Knowledge

The use of the characterisation schema by mature organisations is a more advanced and better use of a characterisation schema of the type proposed here. To use the schema this way, the organisation needs to record its experience and have a description of the techniques in line with the characterisation schema. This way of using the schema involves the organisation systematically recording data and experiences from their projects so that they can assess the parameters of the schema based on their experience.

The testing techniques will be characterised (or described) incrementally. This means that when a technique is described for the first time, it does not necessarily have to be fully instantiated. It will not be until after the technique has been used several times and information has been recorded that the technique will be able to be fully instantiated. This applies to the *experience* parameters (and, depending on the company's maturity level, possibly to the *unknown but under study* parameters), as the technique needs to have been used before in some of the organisation's projects to find out what projects the technique has been used in, the opinion it merits or how effective it is. The knowledge of this information will make the selections more precise and better tailored to the organisation using the techniques. Additionally, the testers will provide feedback on the characterisation values of any techniques that they use. This will add to the available information about the techniques, at the same time as making it more reliable. This selection mode solves the selection problems related to the subjectivity of the selection, unknown techniques that could be relevant for selection purposes or missing information needed for selection purposes.

Another step further towards easing selection for testers is automation as a characterisation software tool that

Table 7
Schema maintenance policy

Maintenance type	Activating event	Activating actor
Add a new technique	Ex officio	Tester
Add information to a partially instantiated technique	After use	Tester
Update information on a (partially or fully) instantiated technique	After use	Tester

contains the organisation of the testing techniques. The benefit of such automation would be that selection would no longer be made manually, because the tool would implement the evaluation function, and the selection would be done automatically after entering the project description and constraints.[4]

5.4. Schema maintenance

A crucial aspect related to the use of the schema by mature organisations is the maintenance of the testing techniques information stored, as the information referred to experience-type parameters or intrinsic parameters whose maturity level is classed as unknown but under study could change over time. Furthermore, as the knowledge on testing techniques advances, previously unknown information may become available. All this means that it will be necessary to establish a maintenance policy to safeguard the coherence of the stored information. This policy will involve defining possible maintenance types. Additionally, for each of these types, the event and the actors who activate the policy will be specified. The policy is outlined in Table 7.

The identified maintenance types refer to:

- Instantiating the schema for new techniques.
- Adding information not currently available about a partially instantiated technique.
- Updating information already available about some technique.

The actor triggering any maintenance action will be a tester in all cases, whereas the events activating each of these three types of maintenance are:

- Ex officio by any tester. The tester gets to know a new technique and thinks it would be of interest to add it to the set of available techniques.
- After using a technique, post-mortem information is gathered about the technique.

Because the sources of any change are wide ranging (potentially all testers), a figure needs to be created to take responsibility for preserving the consistency of the stored information. This figure would be the librarian. The librarian is the only person who is entitled to update or add to the available information, always upon demand from testers.

At this point, the cost of creating and maintaining the information infrastructure associated with schema use by mature organisations also needs to be weighed up. This may, in principle, appear to be a high-cost activity, as organisations would have to meet the costs not only of updating the schema, but also of searching, collecting and then analysing the data needed to fill in the schema. However, the cost can be considered not to be high, because the data collection and analysis costs should be charged not to the schema, but to the data collection and analysis process that any mature organisation should have in place. This applies because schema use for the systematic selection of testing techniques falls within the *Decision Analysis and Resolution* area of CMMI Level 3 (CMMI, 2002), whereas data collection and analysis falls within the *Measurement and Analysis* area of CMMI Level 2. This means that the use of the schema by mature organisations will always mean that this organisation has already deployed a data collection and analysis process. The cost of data collection and analysis will, therefore, be defrayed by the *Measurement and Analysis* area.

6. Conclusions

The main problem that software developers face when choosing the best suited testing techniques for a software project is the availability of information. This article presents the results of using a systematic testing techniques selection process (called characterisation process) presented earlier.

The use of this approach has led to a study of the type of knowledge needed for testing technique selection, as well as its current maturity level, the sources of information that can be used to find the information required for testing technique selection and how these sources are to be used.

At this point, we found that there is information of interest for selection that is not currently available. Information about whose values we are clueless has been left out of the schema. However, immature information has been included as *unknown but under study* information. Despite its immaturity, we have tried to give recommendations about what these parameters should be used for.

The mechanics of selection using the characterisation schema varies with respect to how testing techniques are selected. Depending on the organisation's level of experience accumulation, there are two possible contexts of schema use, each one having its pros and cons.

Finally, we have analysed what maintenance is necessary to support a structure of the type proposed here, including its costs.

[4] A prototype of this type of tool is available at www.ls.fi.upm.es/udis/miembros/sira/tt_tool.

Appendix A. Instantiation of three testing techniques

Level	Element	Parameter	Mutation	Decision coverage	Boundary value analysis
Tactical	Objective	Purpose	Defect detection	Defect detection	Defect detection
		Type of defects	Any	Control	Control
		Effectiveness	Detects approx. 72% of the faults	Prob. detecting a fault: 48%	Finds 55% of defects
	Scope	Element	Unit	Unit	Any
		Aspect	Any	Any	Any
Operational	Technique	Comprehensibility	High	High	High
		Cost of application	Low	Low	Low
		Inputs	Source code	Source code	Code specification
		Adequacy criterion	Mutation	Control flow: decision coverage	Functional: boundary value analysis
		Test data cost	Medium/high	High (less with tools)	Low
		Dependencies	–	Should be completed with techniques that find processing errors	When applied with black-box the effectiveness may rise to 75%
		Repeatability	Yes	No	No
		Sources of information	Frankl et al. (1997), Offut and Lee (1994), Offut et al. (1996), Wong and Mathur (1995)	Beizer (1990), Myers (1970), Pfleeger (1999), Sommerville (1998), Frankl and Iakounenko (1998), Frankl and Weiss (1993), Hutchins et al., 1994, Wood et al. (1997)	Beizer, 1990, Myers (1970), Pfleeger (1999), Sommerville (1998), Basili and Selby (1987), Kamsties and Lott (1995), Wood et al. (1997)
	Test Cases	Completeness	??	Decision	??
		Precision	??	??	??
		# of generated cases	$A + b * n + c * n2$, with $n =$ no of lines of code	Rises exponentially with the number of decisions in the code	Depends on the complexity of the input domain
	Object	Software type	Any	Any	Any
		Software architecture	Any	Any	Any
		Programming language	Structured, OO, real time and concurrent	Any	Any
		Development method	Any	Any	Any
		Size	Medium	Medium	Any
	Tools	Identifier	Mothra	LOGISCOPE	–
		Automation	Generates mutants automatically	Obtain paths	–
		Cost	Free. Academic tool	Between € 3.000 and € 6.000	–
		Environment	Windows/UNIX; Any; Pascal, C	Windows; Any; Ada, C/C++	–
		Support	Information available in tool's web page	24 Hot-line	–
	Agents	Experience	None	None	None
		Knowledge	None	Flow graphs (when tool is not used)	None

(*continued on next page*)

Appendix A (*continued*)

Level	Element	Parameter	Mutation	Decision coverage	Boundary value analysis
Historical	Project	Reference projects	??	??	??
		Tools used	??	??	??
		Personnel	??	??	??
	Satisfaction	Opinion	It is easier to use than it seems	It is okay, but should be completed with others	??
		Benefits	Finds a lot of defects. Easy to use	It is easy to apply	??
		Problems	Should not be used without a tool	If used with real time and concurrent sw, the use of the dynamic analyser should be avoided, as it instruments the code and might change timing constraints	??

References

Basili, V.R., Selby, R.W., 1987. Comparing the effectiveness of software testing strategies. IEEE Transactions on Software Engineering SE-13 (12), 1278–1296.

Beizer, B., 1990. Software Testing Techniques, second ed. International Thomson Computer Press.

CMMI Product Team, 2002. CMMISM for Software Engineering (CMMI-SW, V1.1) Continuous Representation. Capability Maturity Model® Integration (CMMISM), Version 1.1. CMU/SEI-2002-TR-028. ESC-TR-2002-028, August.

Frankl, P., Iakounenko, O., 1998. Further empirical studies of test effectiveness. In: Proceedings of the ACM SIGSOFT International Symposium on Foundations on Software Engineering, Lake Buena Vista, Florida, USA, pp. 153–162.

Frankl, P.G., Weiss, S.N., 1993. An experimental comparison of the effectiveness of branch testing and data flow testing. IEEE Transactions on Software Engineering 19 (8), 774–787.

Frankl, P.G., Weiss, S.N., Hu, C., 1997. All-uses vs mutation testing: an experimental comparison of effectiveness. Journal of Systems and Software 38 (September), 235–253.

Harrold, M.J., 2000. Testing: A roadmap. In: Proceedings of the 22nd International Conference on the Future of Software Engineering, Limerick, Ireland, pp. 63–72.

Hutchins, M., Foster, H., Goradia, T., Ostrand, T., 1994. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In: Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy. IEEE, pp. 191–200.

Kamsties, E., Lott, C.M., 1995. An empirical evaluation of three defect-detection techniques. In: Proceedings of the Fifth European Software Engineering Conference, Sitges, Spain.

Myers, G.J., 1970. The Art of Software Testing. Wiley-Interscience.

Offut, A.J., Lee, S.D., 1994. An empirical evaluation of weak mutation. IEEE Transactions on Software Engineering 20 (5), 337–344.

Offut, A.J., Lee, A., Rothermel, G., Untch, RH., Zapf, C., 1996. An experimental determination of sufficient mutant operators. ACM Transactions on Software Engineering and Methodology 5 (2), 99–118.

Pfleeger, S.L., 1999. Software Engineering: Theory and Practice. Mc-Graw Hill.

RTI, 2000. The Economic Impact of Inadequate Infrastructure for Software Testing. Planning Report 02–3, National Institute of Standards and Technology.

Sommerville, I., 1998. Software Engineering, fifth ed. Pearson Education.

Vegas, S., Basili, V.R., 2005. A characterisation schema for software testing techniques. Empirical Software Engineering 10 (4), 437–466.

Vegas, S., Juristo, N., Basili, V.R., 2003. Identifying Relevant Information for Testing Technique Selection. An Instantiated Characterisation Schema. Kluwer Academia Publishers, Boston, ISBN 1-4020-7435-2.

Wong, E., Mathur, A.P., 1995. Fault detection effectiveness of mutation and data-flow testing. Software Quality Journal 4, 69–83.

Wood, M., Roper, M., Brooks, A., Miller J., 1997. Comparing and combining software defect detection techniques: a replicated empirical study. In: Proceedings of the Sixth European Software Engineering Conference, Zurich, Switzerland.

Sira Vegas is assistant professor of Computer Science at the Universidad Politécnica de Madrid in Spain. She had a summer student grant at the European Centre for Nuclear Research (Geneva) in 1995. In 1997, she worked at GMV (Madrid) on the ENVISAT project for the European Space Agency. She was a regular visiting scholar at the University of Maryland from 1998 to 2000. Sira has a BS and PhD in computer science from the Universidad Politécnica de Madrid. She is a member of IEEE Computer Society and ACM.

Natalia Juristo is full professor of Computer Science at the Universidad Politécnica de Madrid in Spain. She is the Head of the Universidad Politécnica de Madrid's Master of Software Engineering degree programme. Natalia has worked at the European Centre for Nuclear Research (Geneva) and at the European Space Agency (Rome). In 1992 she was Resident Affiliate at the Software Engineering Institute (Pittsburgh) on a NATO Fellowship. Natalia has a BS and PhD in computer science from the Universidad Politécnica de Madrid. She served as Member of the Editorial Board of the IEEE Software Magazine from 1997 to 2001. She is a senior member of IEEE Computer Society and member of ACM, AAAS and NYAS.

Victor R. Basili is a Professor of Computer Science at the University of Maryland. He was founding director of the Fraunhofer Center for Experimental Software Engineering, Maryland, and one of the founders of the Software Engineering Laboratory (SEL) at NASA/GSFC. He received a B.S. from Fordham College, an M.S. from Syracuse University, and a PhD in Computer Science from the University of Texas at Austin. He has been working on measuring, evaluating, and improving the software development process and product for over 30 years. Methods for improving software quality include the Goal Question Metric Approach, the Quality Improvement Paradigm, and the Experience Factory organization.