Empirical Software Engineering, 9, 7–44, 2004.

Reviewing 25 Years of Testing Technique Experiments

NATALIA JURISTO natalia@fi.upm.es
Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid, Spain

ANA M. MORENO
Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid, Spain

SIRA VEGAS
Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n, 28660 Boadilla del Monte, Madrid, Spain

Editor: Ross Jeffery

Abstract. Mature knowledge allows engineering disciplines the achievement of predictable results. Unfortunately, the type of knowledge used in software engineering can be considered to be of a relatively low maturity, and developers are guided by intuition, fashion or market-speak rather than by facts or undisputed statements proper to an engineering discipline. Testing techniques determine different criteria for selecting the test cases that will be used as input to the system under examination, which means that an effective and efficient selection of test cases conditions the success of the tests. The knowledge for selecting testing techniques should come from studies that empirically justify the benefits and application conditions of the different techniques. This paper analyzes the maturity level of the knowledge about testing techniques by examining existing empirical studies about these techniques. We have analyzed their results, and obtained a testing technique knowledge classification based on their factuality and objectivity, according to four parameters.

Keywords: Testing, testing techniques, empirical body of knowledge.

1. Introduction

Engineering disciplines are characterized by the use of mature knowledge by means of which they can achieve predictable results. A series of intermediate steps can be defined on a scale of knowledge that ranges from the more mature knowledge, considered as proven facts, to the less mature knowledge, composed of beliefs or speculations. Therefore, several levels of knowledge can be considered in decreasing order of maturity: facts given as founded and accepted by all, undisputed statements, disputed statements and conjectures or speculations. The ranking of an enunciation and its respective maturity level depends on the change in its factuality status. So, the path from subjectivity to objectivity is paved by testing or empirical comparison with reality. Associating these levels with the type of knowledge used by the engineering disciplines, it can be said that the knowledge formed by facts and undisputed statements would be what is used by these disciplines to obtain products with predictable quality.

Unfortunately, software development has been characterized from its origins by a serious want of empirical facts tested against reality, which provide evidence of the advantages or disadvantages of the different methods, techniques or tools used in the development of software systems. In this respect, the type of knowledge used in this discipline can be considered to be of a relatively low maturity, and developers are guided by reasoning based on intuition, fashion or market-speak rather than by facts or undisputed statements proper to an engineering discipline. This paper intends to analyze the maturity level of the knowledge about a particular part of the software development, the testing area, and more precisely testing techniques.

This is equally applicable to software testing and is in open opposition to the importance of software quality control and assurance and, in particular, software testing. Testing is the last chance during development to detect and correct possible software defects at a reasonable price. It is a well-known fact that it is a lot more expensive to correct defects that are detected during later system operation (Davis, 1993). Therefore, it is of critical importance to rely on knowledge that is mature enough to get predictable results during the testing process.

The selection of the testing techniques to be used is one of the circumstances during testing where objective and factual knowledge is essential. Testing techniques determine different criteria for selecting the test cases that will be used as input to the system under examination, which means that an effective and efficient selection of test cases conditions the success of the tests. The knowledge for selecting testing techniques should come from studies that empirically justify the benefits and application conditions of the different techniques. However, as authors like Hamlet (1989) have noted, formal and practical studies of this kind do not abound, as: (1) it is difficult to compare testing techniques, because they do not have a solid theoretical foundation; (2) it is difficult to determine what testing techniques variables are of interest in these studies.

In view of the importance of having mature testing knowledge, this paper intends to analyze the maturity level of the knowledge in this area. For this purpose, we have surveyed the major empirical studies on testing in order to analyze their results and establish the factuality and objectivity level of the testing body of knowledge regarding the benefits of some techniques over others.

The maturity parameters that we have used are as follows:

- *Laboratory study:* An empirical study should be performed to check whether the perception of the differences between the different testing techniques is subjective or can be objectively confirmed by measurement. In case such a study has been performed, we assign a confirmed value to this criteria; in case the study has not used them, this level of knowledge is pending.

- *Formal analysis:* Statistical analysis techniques should be applied to the results output to find out whether the differences observed between the techniques are really significant and are not due to variations in the environment. In case the study has used these techniques, we assign a confirmed value to this criteria; in case the study has not used them, this level of knowledge is pending.

- *Laboratory replication:* Other investigators should replicate the same experiment to confirm that they get the same results and that they are not the fruit of any uncontroled variation. In case the study has been replicated in laboratory, we assign a confirmed value to this criteria; in case it has not been, this level of knowledge is pending.

- *Field study:* The study should be also replicated using real rather than toy (laboratory) programs or faults in order to understand how the knowledge behaves in real situations. In case the study has a field study associated, we assign a confirmed value to this criteria; in case the study does not have it, this level of knowledge is pending.

As a result, this paper addresses several objectives:

- The main objective is to compile the body of knowledge on testing techniques and its maturity level, in such a way that the information could be useful to developers for identifying the applicability conditions of the different testing techniques.

- The second objective is to provide a picture of what aspects of the testing techniques have been tested empirically, which need more testing and which have not been considered at all, in such a way that researchers can use this information to focus their research on this issue, raising the maturity of the knowledge in the area.

For this purpose, the paper has been structured as follows. Section 2 presents the chosen approach for grouping the different testing studies. Sections 3 to 9 focus on each of the study categories described in Section 1. Each of these sections will first describe the studies considered depending on the testing techniques addressed in each study and the aspects examined by each one. Each study and its results are analyzed in detail and, finally, the findings are summarized. Section 10 outlines the practical recommendations that can be derived from these studies, along with knowledge maturity level, that is, how reliable these recommendations are. Section 10 also indicates what aspects should be addressed in future studies in order to increase the body of empirical knowledge on testing techniques. Finally, Section 11 shows the conclusions drawn from this piece of research.

The organization of this paper means that it can be read differently by different audiences. Software practitioners interested in the practical results of the application of testing techniques will find Section 10, which summarizes the practical recommendations on the use of different testing techniques and their confidence level, more interesting. Researchers interested in raising the maturity of testing knowledge will find the central sections of this paper, which contain a detailed description of the different studies and their advantages and limitations, more interesting. The replication of particular aspects of these studies to overcome the

above-mentioned limitations will contribute to providing useful knowledge on testing techniques. Researchers will also find a quick reference to aspects of testing techniques in need of further investigation in Section 10.

2. Classification of Testing Techniques

The term software testing or dynamic analysis encompasses a set of corrective methods, which aim at determining the quality of a software system by analyzing the results of running it. Testing techniques provide different criteria to select the set of test cases that will be used to exercise the software. These criteria allow grouping testing techniques in families. This way, techniques belonging to the same family are similar as regards the information they need to generate the test cases (source code or specifications) or the aspect of the code exercised by the test cases (control flow, data flow, typical errors, etc.).

This is not the place to describe the features of testing techniques or their families, as this information can be gathered from the classical literature on testing techniques, like, for example, Beizer (1990) and Myers (1979). For readers not versed in the ins and outs of each testing techniques family, however, we will briefly mention each family covered in this paper.

- *Random testing techniques*. The random testing techniques family is composed of the oldest and most intuitive techniques. This family of techniques proposes randomly generating test cases without following any pre-established guidelines. Nevertheless, pure randomness seldom occurs in reality, and the other two variants of the family, are the most commonly used.

- *Functional testing techniques*. This family of techniques proposes an approach in which the program specification is used to generate test cases. The component to be tested is viewed as a black box, whose behavior is determined by studying its inputs and associated outputs.

  Of the set of possible system inputs, this family considers a subset formed by the inputs that cause anomalous system behavior. The key for generating the test cases is to find the system inputs that have a high probability of belonging to this subset. For this purpose, the technique divides the system inputs set into subsets termed equivalence classes, where each class element behaves similarly, so that all the elements of a class will be inputs that cause either anomalous or normal system behavior. The techniques of which this family is composed differ from each other in terms of the rigorousness with which they cover the equivalence classes.

- *Control flow testing techniques*. Control flow testing techniques require knowledge of source code. This family selects a series of paths[1] throughout the program, thereby examining the program control model. The techniques in this family vary as to the rigour with which they cover the code.

- *Data flow testing techniques*. Data flow testing techniques also require knowledge of source code. The objective of this family is to select program paths to explore sequences of events related to the data state. Again, the techniques in this family vary as to the rigour with which they cover the code variable states.

- *Mutation testing techniques*. Mutation testing techniques are based on modeling typical programming faults by means of what are known as mutation operators (dependent on the programming language). Each mutation operator is applied to the program, giving rise to a series of mutants (programs that are exactly the same as the original program, apart from one modified sentence, originated precisely by the mutation operator). Having generated the set of mutants, test cases are generated to examine the mutated part of the program. After generating test cases to cover all the mutants, all the possible faults should, in theory, be accounted for (in practice, however, coverage is confined to the faults modeled by the mutation operators).

  The problem with the techniques that belong to this family is scalability. A mutation operator can generate several mutants per line of code. Therefore, there will be a sizeable number of a mutants for long programs. The different techniques within this family aim to improve the scalability of standard (or strong) mutation to achieve greater efficiency.

- *Regression testing techniques*. Regression testing techniques reduce the cost of regression testing by selectively reusing tests from a program's existing test suite. Selective regression testing techniques differ from the retest-all technique, which runs all tests in the existing test suite. A selective regression testing technique is more economical than the retest-all technique only if the cost of selecting a reduced subset of tests is less than the cost of running the tests that the regression testing technique omits.

- *Improvement testing techniques*. The families of testing techniques above mentioned can be complemented with other techniques that may help to improve their behavior. These techniques can be named *improvement* testing techniques. This family of techniques encompasses those techniques that help either to reduce the number of test cases generated by any of the above mentioned techniques (minimization techniques) or to establish an order in the set of test cases generated, in such a way that defects in the program are found as early as possible during the testing process (prioritization techniques). Although this family can be applied to the set of test cases obtained with any family of techniques, they have been studied only in the regression context.

The techniques that belong to each family appear in Table 1.

This paper focuses on all currently known existing testing technique empirical studies (including replications). For the purpose of simplifying the comparative

Table 1. Classification of testing techniques.

Family	Techniques	Description
Random	Pure random	Test cases are generated at random, and generation stops when there appear to be enough.
	Guided by number of cases	Test cases are generated at random, and generation stops when a given number of cases has been reached.
	Error guessing	Test cases are generated guided by the subject's knowledge of what typical errors are usually made when programming. It stops when they all appear to have been covered.
Functional	Equivalence partitioning	A test case is generated for each equivalence class found. The test case is selected at random from within the class.
	Boundary value analysis	Several test cases are generated for each equivalence class, one that belongs to the inside of the class and as many as necessary to cover the limits (or boundaries) of the class.
Control-flow	Sentence coverage	The test cases are generated so that all the program sentences are executed at least once.
	Decision coverage (branch testing)	The test cases are generated so that all the program decisions take the value true or false.
	Condition coverage	The test cases are generated so that all the conditions (predicates) that form the logical expression of the decision take the value true or false.
	Decision/condition coverage	Decision coverage is not always achieved with condition coverage. Here, the cases generated with condition coverage are supplemented to achieve decision coverage.
	Path coverage	Test cases are generated to execute all program paths. This criterion is not workable in practice.
Data-flow	All-definitions	Test cases are generated to cover each definition of each variable for at least one use of the variable.
	All-c-uses/some-p-uses	Test cases are generated so that there is at least one path of each variable definition to each c-use of the variable. If there are variable definitions that are not covered, use p-uses.
	All-p-uses/some-c-uses	Test cases are generated so that there is at least one path of each variable definition to each p-use of the variable. If there are variable definitions that are not covered, use c-uses.
	All-c-uses	Test cases are generated so that there is at least one path of each variable definition to each c-use of the variable.
	All-p-uses	Test cases are generated so that there is at least one path of each variable definition to each p-use of the variable.
	All-uses	Test cases are generated so that there is at least one path of each variable definition to each use of the definition.
	All-du-paths	Test cases are generated for all the possible paths of each definition of each variable to each use of the definition.
	All-dus	Test cases are generated for all the possible executable paths of each definition of each variable to each use of the definition.

Table 1. Continued.

Family	Techniques	Description
Mutation	Strong (standard) mutation	Test cases are generated to cover all the mutants generated by applying all the mutation operators defined for the programming language in question.
	Selective (or constrained) mutation	Test cases are generated to cover all the mutants generated by applying some of the mutation operators defined for the programming language. This gives rise to selective mutation variants depending on the selected operators, like, for example, two, four or six selective mutation (depending on the number of mutation operators not taken into account) or abs/ror mutation, which only uses these two operators.
	Weak mutation	Test cases are generated to cover a given percentage of mutants generated by applying all the mutation operators defined for the programming language in question. This gives rise to weak mutation variants, depending on the percentage covered, for example, randomly selected 10% mutation, ex-weak, st-weak, bb-weak/1, or bb-weak/n.
Regression	Random	The test cases are randomly selected from the existing test suite.
	Retest-all (traditional)	Runs all tests in the existing test suite.
	Safe	The test selection algorithm excludes no tests from the original test suite that if executed would reveal faults in the modified program. Examples are: Dejá vu, Test tube, Textual differencing, etc.
	Based on modifications	Place an emphasis on selecting existing test cases to cover modified program components and those that may be affected by the modifications.
	Dataflow/coverage-based	Select tests that exercise data interactions that have been affected by modifications.
Improvement	Minimization	Reduces the number of test cases generated.
	Prioritization	Establishes an order in the set of test cases generated. Examples are: Branch/St/Fn-techniques.

analysis of these empirical studies, they have been grouped into several sets according to their affinity as regards the techniques studied. The sets of studies that have been established are:

- *Intra-family studies*. Compare techniques of the same family to find out the best criterion, that is, which technique of all the family members should be used. We have found:

  1. Studies on data-flow testing techniques.

  2. Studies on mutation testing techniques.

  3. Studies on regression techniques.

- *Inter-family studies*. Compare techniques belonging to different families to find out which family is better, that is, which type of techniques should be used. We have identified:

  1. Comparisons between control-flow, data-flow and random techniques.

  2. Comparisons between functional and structural control-flow techniques.

  3. Comparisons between mutation and data-flow techniques.

  4. Comparisons between regression and improvement techniques.

  In the following sections, we examine all these sets of studies, together with the empirical results obtained.

3. Studies on the Data Flow Testing Techniques Family

The objective of this series of studies is to analyze the differences between the techniques within the data flow testing techniques family. Table 2 shows which aspects were studied for which testing techniques. The table can be read as follows: Weyuker analyzed the criterion compliance and the number of test cases generated by four techniques (all-c-uses, all-p-uses, all-uses and all-du-paths), whereas Bieman and Schultz (1992) studied the number of test cases generated for the all-du-paths technique alone.

Table 2. Studies on data flow testing techniques.

	Study	Weyuker (1990)[2]	Bieman and Schultz (1992)
Aspect studied	Criterion compliance	X	
	Number of test cases generated	X	X
Testing technique	All-c-uses	O	
	All-p-uses	O	
	All-uses	O	
	All-du-paths	O	O

The following conclusions can be drawn from these studies:

- All-p-uses should be used instead of all-uses, and all-uses instead of all-du-paths, as they generate fewer test cases and generally cover the test cases generated by the other criteria.

- It is not clear that it is better to use all-c-uses instead of all-p-uses, as, even though all-c-uses generates fewer test cases, there is no guarantee that the generated test cases meet the criterion imposed by all-p-uses.

- Both Weyuker (1993), using toy programs, and Bieman and Schultz (1992), using industrial software, appear to agree that, contrary to testing theory, the all-du-paths technique is usable in practice, since it does not generate too many test cases.

Notice that these results are affected by the following limitations:

- Weyuker uses relatively simple toy programs, which means that the results cannot be directly generalized to real practice.

- Bieman and Schultz (1992) do not conduct a statistical analysis of the extracted data, and their study is confined to a qualitative interpretation of the data.

- The response variable used by Weyuker (1990) and Bieman and Schultz (1992) is the number of test cases generated. This characteristic merits analysis insofar as the fewer test cases are generated, the fewer are run and the fewer need to be maintained. However, it should be supplemented by a study of case effectiveness, which is a variable that better describes what is expected of the testing techniques.

- What the number of test cases generated by all-du-paths depends on needs to be examined in more detail, as one study says it is related to the number of decisions and the other to the number of lines of code, although neither further specifies this relationship.

The results of these studies are summarized in Table 3.

4. Studies on the Mutation Testing Techniques Family

This family is examined in three papers, which look at types of mutation that are less costly than traditional mutation. Generally, these papers aim to ascertain what the costs and benefits of using different mutation testing techniques are. These studies, along with the characteristics they examine and the techniques they address, are shown in Table 4.

The conclusions reached by this group of studies are:

- Standard mutation appears to be more effective, but is also more costly than any of the other techniques studied.

- The mutation variants provide similar, although slightly lower effectiveness, and are less costly (generate fewer mutants and, therefore, fewer test cases), which

Table 3. Results of the studies on data flow testing techniques.

	Study	Weyuker (1990)	Bieman and Schultz (1992)
Aspect studied	Criteria compliance	• All-p-uses includes all-uses • All-uses includes all-du-paths	—
	Number of test cases generated	• All-c-uses generates fewer test cases than all-p-uses • All-p-uses generates fewer test cases than all-uses • All-uses generates fewer test cases than all-du-paths • The number of test cases generated by all-du-paths is linear as regards the number of decisions in the program, rather than exponential as stated in theory	• The number of test cases generated with all-du-paths is not exponential, as stated in theory, and is reasonable • The number of test cases generated by all-du-paths seems to depend on the number of lines of code
Practical results	• All-p-uses should be used instead of all-uses, and all-uses instead of all-du-paths, as they generate fewer test cases and generally cover the test cases generated by the other criteria. • It is not clear that it is better to use all-c-uses instead of all-p-uses, as, even though all-c-uses generates fewer test cases, coverage is not assured. • Contrary to testing theory, the all-du-paths technique is usable in practice, since it does not generate too many test cases.		
Limitations	• It remains to ratify the laboratory results of Weyuker's study in industry. • The results of Bieman and Schultz's study have to be corroborated using formal statistical analysis techniques. • Technique effectiveness should be studied, as the fact that the test cases generated with one criterion cover the other criteria is not necessarily related to effectiveness. • What the number of test cases generated in all-du-paths depends on should be studied in more detail, as one study says it depends on the number of decisions and the other on the number of lines of code.		

means that the different mutation variants could be used instead of strong mutation for non-critical systems.

However, the following limitations have to be taken into account:

- The programs considered in these studies are not real, which means that the results cannot be generalized to industrial cases, and a replication in this context is required to get greater results reliability.

- Offut et al. (1994) and Wong and Mathur (1995) do not use formal techniques of statistical analysis, which means that their results are questionable.

- Additionally, it would be interesting to compare the standard mutation variants with each other and not only with standard mutation to find out which are more effective from the cost and performance viewpoint.

Table 4. Studies on mutation testing techniques.

	Study	Offut and Lee (1994)[3]	Offut et al. (1996)[4]	Wong and Mathur (1995)
Aspect studied	Percent mutants killed[5] by each technique	X	X	
	No. of generated cases	X	X	
	No. of generated mutants		X	X
	Percent generated sets that detect at least one fault			X
Testing technique	Mutation (strong/standard)	O	O	O
	MD EX-WEAK	O		
	MD ST-WEAK	O		
	MD BB-WEAK/1	O		
	MD BB-WEAK/n	O		
	two-selective mutation		O	
	four-selective mutation		O	
	six-selective mutation		O	
	Random selected 10% mutation			O
	Constrained (abs/ror) mutation			O

- Furthermore, the number of mutants that a technique kills is not necessarily a good measure of effectiveness, because it is not explicitly related to the number of faults the technique detects.

Table 5 shows the results of this set of studies.

5. Studies on Regression Testing Techniques Family

This group includes studies that analyze different regression techniques, that is, techniques applied during software system maintenance. These studies are shown in Table 6.

The conclusions that can be drawn from these studies are:

- For small programs:

  1. If the sets of test cases are small, then it is advisable to apply retest-all.

  2. If the sets of test cases are big, then it is advisable to apply safe techniques, like deja-vu or test-tube.

  3. For medium-sized sets of test cases, no results have been found as regards time. So, no conclusions can be drawn.

Table 5. Results of the studies on mutation testing techniques.

	Study	Offut and Lee (1994)	Offut et al. (1996)	Wong and Mathur (1995)
Aspect studied	Percent mutants killed by each technique	The percentage of mutants killed by weak mutation is high	Selective mutation kills more than 99% of the mutants generated by strong mutation	—
	No. cases generated	Weak mutation generates fewer test cases than strong mutation	• st-weak/1 and bb-weak/1 generate more test cases than exweak/1 and bb-weak/n • Although not explicitly stated, strong mutation generates more test cases than selective mutation	—
	No. mutants generated	—	Selective mutation generates fewer mutants than strong mutation	• 10% mutation generates fewer mutants than abs/ror mutation (approx. half) • Abs/ror generates from 50 to 100 times fewer mutants than standard mutation
	Percent sets generated that detect at least one fault	—	—	• Standard mutation is more effective than 10% mutation in 90% of cases and equal in 10% • Standard mutation is more effective than abs/ror in 40% of the cases and equal in 60% • Abs/ror is equally or more effective than 10% mutation in 90% of the cases
Practical results	• Where time is a critical factor, it is better to use weak as opposed to standard mutation, as it generates fewer test cases and effectiveness is approximately the same. • Where time is a critical factor, it is better to use selective (exweak/1 and bb-weak/n) as opposed to standard mutation, as it generates fewer mutants (and therefore fewer test cases) and its effectiveness is practically the same. • Where time is a critical factor, it is better to use 10% selective as opposed to standard mutation, although there is some loss in effectiveness, because it generates much fewer test cases. In intermediate cases, it is preferable to use abs/ror mutation, because, although it generates more cases (from 50 to 100 times more), it raises effectiveness by seven points. If time is not a critical factor, it is preferable to use standard mutation.			
Limitations	• It remains to ratify the laboratory results of these studies in industry. • The results of the studies by Offut et al. (1996) and Wong and Mathur (1995) should be corroborated using formal techniques of statistical analysis. • It remains to compare the variants of strong mutation with each other. • The studies should be repeated with another measure of effectiveness, as the number of mutants killed by a technique is not necessarily a good measure of effectiveness.			

Table 6. Studies on regression testing techniques.

	Study	Rothermel and Harrold (1998)	Bible et al. (1999)	Vokolos and Frankl (1998)	Kim et al. (2000)	Graves et al. (1998)
Aspect studied	Time to select test cases	X	X	X		
	Time to run test cases	X	X			
	Percent selected test cases over the initial set	X	X	X	X	X
	Percent sets that detect a fault in the program					X
	N. faults T′/n. faults T (relative effectiv.)				X	
	N. faults T′/n. faults in the program (Abs. effect.)				X	
	Speed of fault detection (per n. cases)					
Testing technique	Deja-Vu	O	O		O	
	Retest-all (Traditional)	O	O	O	O	O
	Test Tube		O		O	
	Textual differencing			O		
	Random (ad hoc)				O	O
	Dataflow/coverage based					O

- For big programs (more than 10,000 LOC), it is better to use safe techniques as regards time.

- Finally, both Kim et al. (2000) and Bible et al. (1999) predict that the technique efficiency depends on the fault inserted (modification made to the program) and on the program. It would be interesting to study this dependency formally and determine the application conditions of these techniques more precisely.

- The percentage of selected test cases is lower for randomization than for safe and data-flow techniques.

- Effectiveness rises for the random technique as the number of test cases selected increases, is similar for safe and data-flow and depends on the program for random, data-flow and safe.

However, the previous results are subject to the following limitations:

- As was the case with earlier studies, the first question to be taken into account in this group is that the conclusions obtained by the authors are based not on statistical analysis but on the qualitative analysis of graphical representations of the results. This means that their conclusions cannot be considered as much more than orientations and in no case as reliable results.

Table 7. Results of the studies on regression testing techniques.

	Study	Rothermel and Harrold (1998)	Bible et al. (1999)	Kim et al. (2000)	Vokolos and Frankl (1998)	Graves et al. (1998)
Aspect studied	Time to select cases	Deja-vu takes less time to select cases than retest-all	• In small programs: • With small sets of test cases, deja-vu takes longer than test-tube and both take longer than retest-all • With big sets of test cases, deja-vu and test-tube take less time than retest-all • With medium-sized test cases, there are no results • In big programs: • Deja-vu takes less time than retest-all • There are no definite results between deja-vu and test-tube	—	Textual differencing takes less time than deja-vu, but they cannot be compared, because they use different computers	—
	Time to run cases	Deja-vu takes less time to run cases than retest-all	• In small programs: • With small sets of test cases, deja-vu takes longer than test-tube and both take longer than retest-all • With big sets of test cases, deja-vu and test-tube take less time than retest-all • With medium-sized test cases, there are no results • In big programs: • Deja-vu takes less time than retest-all • There are no definite results between deja-vu and test-tube	—	There are no definite results regarding the time taken by textual differencing, deja-vu and retest-all	—

Table 7. Continued.

	Study	Rothermel and Harrold (1998)	Bible et al. (1999)	Kim et al. (2000)	Vokolos and Frankl (1998)	Graves et al. (1998)
	Percent selected cases over initial set	• Deja-vu selects a lower percentage of test cases than retest-all • The percentage of selected cases depends on the structure of the program, its modifications and the coverage	• Deja-vu selects fewer test cases than test-tube • Deja-vu and test-tube select fewer test cases than retest-all	Depends on the program, fault type and location of the fault	Much lower percentage for textual differencing compared to retest-all	• Depends on the program • Higher reduction for randomization and than for safe and data-flow
	Percent sets that detect a fault	—	—	—	—	• Rises for the random technique as the number of test cases selected increases • Similar for safe and data-flow • Depends on the program for random, data-flow and safe
	N. faults T′/n. faults T (Relative effectiveness)	—	—	• Relative effectiveness is maximum for safe techniques and retest-all • More than 90% for random	—	—
Aspect studied	N. faults T′/ total n. faults in program (Absolute effectiveness)	—	—	The more faults there are in the program, the less effective all techniques are	—	—

Table 7. Continued.

Practical results	• For small programs with small sets of test cases, it is preferable to use retest-all as opposed to deja-vu and test-tube, as it takes less time to select and run test cases. For big programs, and small programs with big sets of test cases, it is preferable to use deja-vu or test-tube as opposed to retest-all, as it takes less time to select and run test cases. For small programs with medium-sized test cases, there are n results. • When it is critical the number of test cases selected, it is preferable to use deja-vu as opposed to test-tube and test-tube as opposed to retest-all, since deja-vu selects fewer test cases than test-tube and test-tube selects fewer test cases than retest-all. • The percentage of selected cases over initial set depends on the structure of the program, and its modifications. • It is not clear whether to use textual differencing as opposed to retest-all. Although textual differencing selects a lower percentage of test cases over the initial set than retest-all, there are no definite results regarding the time taken to run test cases. There are also no definite results for textual differencing and deja-vu.
Limitations	• It remains to ratify the laboratory results of these studies in industry. • The results of these studies should be corroborated using formal techniques of statistical analysis. • The studies should be repeated for all techniques and all response variables. • The difference between function coverage techniques should be explored with more detail.

- Additionally, not all the programs used in these studies are real, which means that their representativeness would be questionable, especially in the studies that have considered versions of the programs with only one fault.

- When trying to extract homogeneous conclusions from all the studies, we find that, although some techniques and response variables considered in the studies coincide, not all of them do, which makes it difficult to obtain comparable conclusions from these studies.

Table 7 show the results of this set of studies.

6. Comparative Studies between the Data Flow, Control Flow and Random Testing Techniques Families

The objective of this series of studies is to analyze the differences between three families, selecting, for this purpose, given techniques from each family. The selected techniques are the branch testing (decision coverage) control flow technique, all-uses and all-dus within the data flow family and random in the random testing technique family. Table 8 shows the studies considered, the aspects studied by each one and for which testing techniques.

Table 8. Comparative studies of the data flow, control flow and random testing technique families.

	Study	Frankl and Weiss (1993)[6]	Hutchins et al. (1994)	Frankl and Iakounenko (1998)
Aspect studied	Number of test cases generated	X	X	
	No. of sets with at least one fault/no. of sets generated	X	X	X
Testing technique	All-uses	O		O
	Branch testing (all-edges)	O	O	O
	All-dus (modified all-uses)		O	
	Random (null)	O	O	O

The knowledge that can be drawn from these studies is:

- There does not appear to be a difference between all-uses, all-edges and random testing as regards effectiveness from the statistical viewpoint, as the number of programs in which one comes out on top of the other is not statistically significant. However, from the practical viewpoint, random testing is easier to satisfy than all-edges and, in turn, all-edges is easier to satisfy than all-uses. On the other hand, all-uses is better than all-edges and than random testing as a technique, whereas all-edges is better than random because it generates more test cases. It follows from the results of the above studies that, in the event of time constraints, the use

of the random testing technique can be relied upon to yield an effectiveness similar to all-uses in 50% of the cases. Where testing needs to be exhaustive, the application of all-uses provides assurance, as, in the other half of the cases, this criterion yielded more efficient results thanks to the actual technique and not because it generated more test cases.

- A logical relationship between coverage and effectiveness was also detected (the greater the coverage, the greater the effectiveness). However, effectiveness is not necessarily optimum in all cases even if maximum coverage is achieved. Therefore, it would be interesting to analyze in detail the faults entered in the programs in which the effectiveness of the techniques is below optimum, as a dependency could possibly be identified between the fault types and the techniques that detect these faults.

- Hutchins et al. (1994) discover a direct relationship between coverage and effectiveness for all-uses and all-edges, whereas no such relationship exists for random testing. For high coverage levels, the effectiveness of all-uses and all-edges is similar.

- Frankl and Iakounenko (1998) also discover a direct relationship between coverage and effectiveness for all-uses and all-edges. Again, the effectiveness of both techniques is similar, although all-edges and all-dus are complementary because they detect different faults.

- Even when there is maximum coverage, however, there is no guarantee that a fault will be detected. This suggests that the techniques may be sensitive to certain fault types.

The limitations discovered in these studies are:

- Frankl and Weiss (1993) and Hutchins et al. (1994) use relatively simple, non-industrial programs, which means that the results cannot be directly generalized to real practice.

- Of the three studies, Frankl and Iakounenko (1998) do not run a statistical analysis of the extracted data, which means that the significance of the results is questionable.

- The evaluation of the effectiveness of the techniques studied, measured as the probability of detecting at least one fault in the programs, is not useful in real practice. Measures of effectiveness like, for example, number of faults detected over number of total faults are more attractive in practice.

- Besides technique effectiveness, Frankl and Weiss (1993) and Frankl and Iakounenko (1998) should also study technique complementarity (as in Hutchins

et al., 1994), in order to be able to determine whether or not technique application could be considered exclusive, apart from extracting results regarding similar technique effectiveness levels.

The conclusions of these studies have been summarized in Table 9.

Table 9. Results of the studies comparing data flow, control flow and random testing techniques.

	Study	Frankl and Weiss (1993)	Hutchins et al. (1994)	Frankl and Iakounenko (1998)
Aspect studied	Number of test cases generated	• All uses is a better technique than all-edges and random by the technique itself • All-edges is better than random because it generates more test cases	• All-edges and all-dus generate approx. the same number of test cases • To achieve the same effectiveness as all-edges and all-dus, random has to generate from 50% to 160% more test cases	—
	No. of sets detecting at least one fault/no. of sets generated	• There is no convincing result regarding all-uses being more effective than all-edges and random: • In approximately 50% of the cases, all-uses is more effective than all-edges and random, and all-edges is more effective than random • In approximately 50% of the cases, all-uses, all-edges and random behave equally	• The effectiveness of all-edges and all-dus is similar, but they find different faults • Maximum coverage does not guarantee that a fault will be detected	• There is an effectiveness/coverage relationship in all-edges and all-uses (not so in random) • There is no difference as regards effectiveness between all-uses and all-edges for high coverages
Practical results	• In the event of time constraints, the use of the random testing technique can be relied upon to yield an effectiveness similar to all-uses and all-edges (the differences being smaller the higher coverage is) in 50% of the cases. Where testing needs to be exhaustive, the application of all-uses provides assurance, as, in the other half of the cases, this criterion yielded more efficient results thanks to the actual technique, unlike all-edges, which was more efficient because it generated more test cases. • All-edges should be applied together with all-dus, as they are equally effective and detect different faults. Additionally, they generate about the same number of test cases, and the random testing technique has to generate between 50% and 160% more test cases to achieve the same effectiveness as all-edges and all-dus. • High coverage levels are recommended for all-edges, all-uses and all-dus, as this increases their effectiveness. This is not the case for the random testing technique. Even when there is maximum coverage, however, there is no guarantee that a fault will be detected.			
Limitations	• It remains to ratify the laboratory results of the studies by Hutchins et al. (1994) and Frankl and Ianounenko (1998) in industry. • The results of the studies by Frankl and Weiss (1993) should be corroborated using formal techniques of statistical analysis. • The type of faults should be studied in the programs where maximum effectiveness is not achieved despite there being maximum coverage, as this would help to determine technique complementarity. • The studies should be repeated for a more practical measure of effectiveness, as the percentage of test case sets that find at least one fault is not real.			

7. Comparisons between the Mutation and the Data Flow Testing Techniques Families

We have found two studies that compare mutation with data flow techniques. These studies, along with the characteristics studied and the techniques addressed, are shown in Table 10.

We cannot conclude from these results that there is a clear difference in terms of effectiveness between mutation testing and all-uses. Additionally, the authors highlight that it is harder to get high coverage with mutation as compared with all-uses.

As a general rule, mutation testing appears to be as or more effective than all-uses, although it is more costly.

The limitations of these results are:

- The results of Frankl et al. (1997) can be considered as a first attempt at comparing mutation testing techniques with all-uses, as this study has some drawbacks. First, the faults introduced into the programs had faults that, according to the authors, "occurred naturally". However, the programs are relatively small (no more than 78 LOC), and it is not said whether or not they are real. Additionally, the fact that the programs had no more than two faults is not significant from a practical viewpoint.

- Wong and Mathur (1995) do not use real programs or formal techniques of statistical analysis, which means that their results cannot be considered conclusive until a formal analysis of the results has been conducted on real programs.

- The use of the percentage of sets that discover at least one fault as the response variable is not significant from a practical viewpoint.

- Note that a potentially interesting question for this study would have been to examine the differences in the programs for which mutation and data flow testing techniques yield different results. This could have identified a possible relationship between program or fault types and the techniques studied, which would help to define application conditions for these techniques. There should be a more detailed

Table 10. Comparative studies between the mutation and data flow testing techniques families.

	Study	Frankl et al. (1997)[7]	Wong and Mathur (1995)
Aspect studied	Percent mutants killed by each technique	X	
	Ratio of generated sets that detect at least one fault	X	X
Testing technique	Mutation (strong or standard)	O	O
	All-uses	O	O
	Random selected 10% mutation		O
	Constrained (abs/ror) mutation		O

Table 11. Comparisons between mutation and all-uses.

	Study	Frankl et al. (1997)	Wong and Mathur (1995)
Aspect studied	Percent mutants killed by each technique	• It is more costly to reach high coverage levels with mutation than with all-uses	—
	Ratio of sets generated that detect at least one fault	• There is not a clear difference between mutation and all-uses	• Standard mutation is more effective than all-uses in 63% of the cases and equally effective in 37% • Abs/ror is more effective than all-uses in 50% of the cases, equally effective in 30% and less effective in 20% • All-uses is more effective than 10% mutation in 40% of the cases, equally effective in 20% and less effective in 40%
Practical results	• If high coverage is important and time is limited, it is preferable to use all-uses as opposed to mutation, as it will be just as effective as mutation in about half of the cases. • All-uses behaves similarly as regards effectiveness to abs/ror and 10% mutation.		
Limitations	• It remains to ratify the laboratory results of the studies in industry. • The studies should be repeated for a more practical measure of effectiveness, as the percentage of sets of cases that find at least one fault is not real. • It would be of interest to further examine the differences in the programs in which mutation and the data flow testing technique yield different results. • The cost of technique application should be studied.		

study of the dependency between the technique and the program type to be able to more objectively determine the benefits of each of these techniques.

- In any replications of this study, it would be important to analyze the cost of technique application (in the sense of application time and number of test cases to be applied) to conduct a more detailed cost/benefit analysis.

The main results of this group are summarized in Table 11.

8. Comparisons between the Functional and Control Flow Testing Techniques Families

The four studies of which this group is composed are reflected in Table 12. These are empirical studies in which the authors investigate the differences between control flow testing techniques and the functional testing techniques family. These studies actually also compare these two testing technique families with some static code

Table 12. Comparative studies of functional and control testing techniques.

	Study	Myers (1978)	Basili and Selby (1987)[8]	Kamsties and Lott (1995)	Wood et al. (1997)
Aspect studied	No. faults detected	X	X		X
	Time to detect faults	X	X	X	
	Time to detect faults/fault type	X			
	No. faults detected combining techniques	X			X
	Time combining techniques/fault type	X			
	No. faults found/time	X	X	X	X
	No. faults isolated/hour			X	
	% faults detected/type		X	X	
	% faults isolated/type			X	
	Time to isolate faults			X	
	Total time to detect and isolate			X	
	% faults detected		X	X	X
	% faults isolated			X	
Testing technique	White box	O			
	Black box	O			
	Boundary value analysis		O	O	O
	Sentence coverage		O		
	Condition coverage			O	
	Decision coverage (branch testing)				O

analysis techniques, which are not taken into account for the purposes of this paper, as they are not testing techniques.

The conclusions that can be drawn are:

- The boundary analysis technique appears to behave differently compared with different structural testing techniques (particularly, sentence coverage and condition coverage). Note that from the practical viewpoint, condition coverage is more applicable, which means that future replications should focus on condition coverage rather than sentence coverage. Nothing can be said about branch testing.

- Basili and Selby (1987), Kamsties and Lott (1995) and Wood et al. (1997) find effectiveness-related differences between functional and structural techniques depending on the program to which they are applied. Wood et al. (1997) further examine this relationship, indicating that it is the fault type that really influences the detected faults (and, more specifically, the influential factor is the failures that these faults cause in programs), whereas Kamsties and Lott (1995) and Myers (1978) find no such difference.

- Also there appears to be a relationship between the programs, or the type of faults entered in the programs, and technique effectiveness, as indicated by all three studies. However, this relationship has not been defined in detail. Basili and Selby (1987) point out that the functional technique detects more control faults. Myers (1978) also discerns a difference as regards different faults, but fails to conduct a statistical analysis. Finally, Kamsties and Lott (1995) find no such difference, which means that a more exhaustive study would be desirable.

- More faults are detected using the same technique if different people are combined than individually.

- Any possible extensions of these studies should deal, whenever possible, with real problems and faults in order to be able to generalize the results obtained.

- Finally, it would be recommendable to unify the techniques under study in future replications to be able to generalize conclusions.

The studies considered in this group generally include an experimental design and analysis, which means that their results are reliable. However, caution needs to be exercised when generalizing and directly comparing these results for several reasons:

- They use relatively small programs, between 150 and 350 LOC, which are generally toy programs and might not be representative of industrial software.

- Most, although not all, of the faults considered in these programs are inserted by the authors ad hoc for the experiments run, which means that there is no guarantee that these are faults that would occur in real programs.

- The studies by Basili and Selby (1987), Kamsties and Lott (1995), and Wood et al. (1997) compare the boundary value analysis technique with three different structural techniques. Hence, although some results of different studies may appear to be contradictory at first glance, a more detailed analysis would be needed to compare the structural techniques with each other.

- The results of Myers' study are not useful, since it is not clear what are exactly the techniques the subjects used.

- Although the response variables used in all the studies are quite similar, care should be exercised when directly comparing the results, because, as mentioned above, the techniques studied are not absolutely identical.

We have summarized the results of this group in Table 13.

Table 13. Results of the comparison of the functional and control flow testing technique families.

	Study	Basili and Selby (1987)	Kamsties and Lott (1995)	Wood et al. (1997)

Aspect studied	No. faults detected	• Experienced subjects: the functional technique detects more faults than the structural technique
		• Inexperienced subjects:
		  • In one case, there is no difference between structural and functional techniques
		  • In the other, the functional technique detects more faults than the structural technique
			—	The number of detected faults depends on the program/technique combination

	Percent faults detected	• Experienced subjects: The functional technique detects more faults than the structural technique
		• Inexperienced subjects:
		  • In one case, there is no difference between the structural and functional techniques
		  • In the other case, the functional technique detects more faults than the structural technique
			Depends on the program, not the technique	The percentage of detected faults depends on the program/technique combination

	Percent faults detected/type	• Boundary value analysis detects more control faults than sentence coverage • There is no difference between these techniques for other fault types	There is no difference between techniques	—
	No. faults detected combining techniques	—	—	Higher number of faults combining techniques
	Time to detect faults	• Experienced subjects: Boundary value analysis takes longer than sentence coverage • Inexperienced subjects: Boundary value analysis takes as long or longer than sentence coverage	• Inexperienced subjects: • Boundary value analysis takes less time than condition coverage • The time taken to faults also depends on the subject	—
	No. faults detected/hour	• The fault rate with boundary value analysis and sentence coverage does not depend on experience • The fault rate depends on the program	Boundary value analysis has a higher fault detection rate than condition coverage	Depends on the type of faults in the programs
	Percent faults isolated	—	Depends on the program and subject, not on the technique	—
	No. faults isolated/hour	—	Is influenced by the subject not by the technique	—
	Percent faults isolated/type	—	There is no difference between techniques	—
	Time to isolate faults	—	With inexperienced subjects, boundary value analysis takes longer than condition coverage	—
	Total time to detect and isolate	—	• With inexperienced subjects, boundary value analysis takes less time than condition coverage • Time also depends on the subject	—
Practical results	• For experienced subjects and when there is plenty of time, it is better to use the boundary value analysis technique as opposed to sentence coverage, as subjects will detect more faults, although it will take longer. On the other hand, for inexperienced subjects and when time is short, it is better to use sentence coverage as opposed to boundary value analysis, although there could be a loss of effectiveness. The time will also depend on the program. • It is preferable to use boundary value analysis as opposed to condition coverage, as there is no difference as regards effectiveness and it takes less time to detect and isolate faults. • There appears to be a dependency on the subject as regards technique application time, fault detection and fault isolation. • There appears to be a dependency on the program as regards the number and type of faults detected. • More faults are detected by combining subjects than techniques of the two families.			

Table 13. Continued.

	Study	Basili and Selby (1987)	Kamsties and Lott (1995)	Wood et al. (1997)
	• If control faults are to be detected, it is better to use boundary value analysis or condition coverage than sentence coverage. Otherwise, it does not matter which of the three are used. • The effect of boundary value analysis and branch testing techniques on effectiveness cannot be separated from the program effect.			
Limitations	• It remains to ratify the laboratory results of the studies in industry. • The studies compare boundary values analysis with three different structural testing techniques, hence a more detailed analysis is needed to compare the structural testing techniques with each other.			

9. Comparisons between the Regression and Improvement Testing Techniques Families

This group includes studies that compare the use of regression techniques with incorporating to these techniques test case improvement techniques. Table 14 shows these studies.

The conclusions that can be drawn from these studies are:

- Regression based on modifications selects a lower percentage of test cases when used with minimization, precision does not fall too much, and recall is increased.
- The more faults that there are in the program, the more effective minimization is.
- The percentage of selected test cases is lower for minimization, as well as effectiveness.
- Prioritization yields better results than no prioritization for recall and fault detection speed.

However, these conclusions are limited by the following constraints:

- Note that, as in the study by Kim et al. (2000) the results obtained within this group have not been generated statistically, which means that they are not completely reliable. Furthermore, as mentioned earlier, the representativeness of the programs with only one fault is limited in a real environment. Remember that their interpretation is subject to the above-mentioned factors.
- Also, the study by Graves et al. (1998) could be improved with a study of the time, since it indicates that the studied techniques have a similar percentage of test case set reduction and effectiveness. Therefore, it would be interesting to quantify this reduction in terms of time savings in order to establish whether or not they should be used in preference to other techniques.
- The main obstacle encountered when drawing joint conclusions from the studies in this group is that they use programs with different number of faults. More

Table 14. Comparative studies of regression and improvement testing technique families.

	Study	Graves et al. (1998)	Wong et al. (1998)	Rothermel et al. (1999)	Elbaum et al. (2000)	Kim et al. (2000)
Aspect studied	Percent selected test cases over the initial set	X	X			X
	Percent sets that detect a fault in the program	X				
	N. faults T′/n. faults T (relative effectiv.)					X
	Precision (% cases that detect a fault)		X			
	N. faults T′/n. faults in the program (Abs. effect.)					X
	Recall (% selected cases over the cases that should have been selected)		X			
	Speed of fault detection (per n. cases)			X	X	
Testing techniques	Deja-Vu					O
	Retest-all (Traditional)	O		O		O
	Test Tube					O
	Regression based on modifications		O			
	Regression based on modifications + prioritization		O			
	Regression based on modifications + minimization		O			
	No ordering			O		
	Random (ad hoc)	O		O	O	O
	Optimum		O	O	O	
	St-total			O	O	
	St-addtl			O	O	
	St-fep-total			O	O	
	St-fep-addtl			O	O	
	Branch-total			O		
	Branch-addtl			O		
	Fn-total				O	
	Fn-addtl				O	
	Fn-fep-total				O	
	Fn-fep-addtl				O	
	Fn-fi-total				O	
	Fn-fi-addtl				O	
	Fn-fi-fep-total				O	
	Fn-fi-fep-addtl				O	
	Minimization	O				O
	Safe	O				
	Dataflow/coverage-based	O				

precisely, as already mentioned, Kim et al. (2000) use programs with more than one fault, and Graves et al. (1998) use the same programs with just one fault. Although they use a different number of faults, there seems to some agreement on the fact that the number of selected tests cases depends on the programs. This idea was also observed in group 4.1, and it would, therefore, be interesting to go into this issue in more depth.

- The results of Rothermel et al. (1999) and Elbaum et al. (2000) have been obtained statistically, which means that they do not have the limitations as the previous groups had.

Table 15. Results of the comparison of the regression and improvement testing technique families.

	Study	Wong et al. (1998)	Graves et al. (1998)	Kim et al. (2000)	Rothermel et al. (1999)	Elbaum et al. (2000)
Aspect	Percent selected cases over initial set	Lower percentage of selected cases with regression based on modifications and minimization	• Depends on the program • Higher reduction for minimization	Depends on the program, fault type and location of the fault	—	—
	Percent sets that detect a fault	—	• Lower for minimization	—	—	—
	N. faults T′/n. faults T (Relative effectiveness)	—	—	• From 16% to 60% for minimization	—	—
	Precision (% cases that detect a fault)	Precision does not fall too much in the case of regression based on modifications with prioritization and minimization	—	—	—	—
	N. faults T′/ total n. faults in program (Absolute effectiveness)	—	—	The more faults there are in the program, the more effective minimization is	—	—
	Recall	Better recall with regression based on modifications with prioritization and minimization	—	—	—	—
	Fault detection speed	—	—	—	• Although the program does have an influence on efficiency, fep techniques are faster than sentence coverage techniques	• Although the program does have an influence on efficiency, fep techniques are faster than sentence coverage techniques • There is a difference between function coverage techniques

Table 15. Continued.

	Study	Wong et al (1998)	Graves et al (1998)	Kim et al (2000)	Rothermel et al (1999)	Elbaum et al, (2000)
Aspect	Fault detection speed				• Prioritization yields better results than no prioritization	• Sentence coverage techniques are more effective and more costly than function coverage techniques • Prioritization yields better results than no prioritization
Practical results	• The more faults there are in the program, the more effective minimization is. • In the event of time constraints, the use of regression based on modifications and minimization and prioritization can be relied upon to yield an effectiveness similar to regression based on modifications only. Also, a better recall is obtained. • In the event of time constraints, the use of minimization can be relied upon to yield a lower effectiveness than safe and data-flow (these two equal) and random. Effectiveness for these techniques depends on the program. • In the event of time constraints, the use of prioritization can be relied upon to yield a higher fault detection speed than no prioritization. • It is preferable to use fep techniques as opposed to sentence coverage technique, as they have a higher fault detection speed. • Function coverage techniques do not behave equally. • When time is not an issue, it is preferable to use sentence coverage techniques as opposed to function coverage techniques, as they are more effective, but also more costly.					
Limitations	• It remains to ratify the laboratory results of these studies. • The results of the study by Kim et al. (2000) should be corroborated using formal techniques of statistical analysis. • It remains to study the time in Graves et al. (1998) study with a study of the time, since it indicates that the studied techniques have a similar percentage of test case set reduction and effectiveness. Therefore, it would be interesting to quantify this reduction in terms of time savings in order to establish whether or not they should be used in preference to other techniques. • The studies should use programs with the same number of faults. • It would be interesting to go into the issue that the number of selected test cases depends on the program in more depth. • Wong et al. (1998) study should provide quantitative values that can be used to extract numerical results as regards efficiency and cost of the techniques it studies. • It remains to replicate these studies in order to generalize results. • The application cost of the groups of techniques studied, sentence and function coverage, should be quantified with the aim of conducting a detailed cost-benefit analysis about the applicability of both groups of techniques.					

- Wong et al. (1998) study has a limited practical usefulness, since it does not provide quantitative values that can be used to extract numerical results as regards efficiency and cost of the techniques it studies.

- It is interesting to note that both studies agree on the benefits of st-fep techniques. However, replications should be carried out with different programs in order to generalize these results.

- The application cost of the groups of techniques studied, sentence and function coverage, should be quantified with the aim of conducting a detailed cost-benefit analysis about the applicability of both groups of techniques.

Table 15 shows the results of these studies.

10. Discusion on Testing Technique Knowledge Maturity Level

Despite the difficulties encountered, *practitioners* can find interesting recommendations in Tables 16–22. These tables also show the maturity level and tests *pending* performance for every statement, which can be interesting for *researchers*. Note that there is no statement on testing techniques that can be accepted as fact, as they

Table 16. Conclusions for the data flow family studies.

Technique	Practical recommendation	Maturity status	Pending knowledge
Data flow	If *time is an* issue, all-p-uses should be used instead of all-uses, and all-uses instead of all-du-paths, as they generate fewer test cases and generally cover the test cases generated by the other criteria.	• *Confirmed* with laboratory programs and faults. • *Confirmed* formally.	• Find out the difference in terms of effectiveness between all-c-uses, all-p-uses, all-uses and all-du-paths.
	It is not clear that it is better to use all-c-uses instead of all-p-uses, as, even though all-c-uses generates fewer test cases, coverage is not assured.	• *Pending* laboratory replication. • *Pending* field study.	• Compare with the other techniques in the family. • Find out whether the fact that maximum coverage does not detect a fault depends on the fault itself.
	All-du-paths is not as time consuming as stated by the theory, as it generates a reasonable and not an exponential number of test cases.	• *Confirmed* with laboratory programs and faults. • *Confirmed* with field study. • *Pending* formal analysis. • *Pending* laboratory replication.	• Confirm whether the number of test cases generated by all-du-paths depends on the number of sentences or the number of decisions, as the two authors disagree.

Table 17. Conclusions for the mutation family studies.

Technique	Practical recommendation	Maturity status	Pending knowledge

Mutation	Where time is a critical factor, it is better to use selective (exweak/1 and bb-weak/n) as opposed to standard mutation, as it generates fewer mutants (and, therefore, fewer test cases) and its effectiveness is practically the same.
	
	Where time is a critical factor, it is better to use weak as opposed to standard mutation, as it generates fewer test cases and effectiveness is approximately the same.
	Where time is a critical factor, it is better to use 10% selective as opposed to standard mutation, although there is some loss in effectiveness, because it generates much fewer test cases. In intermediate cases, it is preferable to use abs/ror mutation, because, although it generates more cases (from 50 to 100 times more), it raises effectiveness by seven points. If time is not a critical factor, it is preferable to use standard mutation.
		• *Confirmed* with laboratory programs and faults.
		• *Confirmed* formally.
		• *Pending* laboratory replication.
		• *Pending* field study.
		  • *Confirmed* with laboratory programs and faults.
		  • *Pending* formal analysis.
		  • *Pending* laboratory replication.
		  • *Pending* field study.
			• Compare the different mutation variants with each other. Use another metric type for effectiveness, as the number of mutants killed by a technique is only useful for relative comparisons between mutation techniques.

are all pending some sort of corroboration, be it laboratory or field replication or knowledge pending formal analysis.

Furthermore, some points yet to be covered by empirical studies and which might serve as inspiration for *researchers* should be highlighted:

- The comparative study of the effectiveness of different techniques should be supplemented by studies of the fault types that each technique detects and not only the probability of detecting faults. That is, even if T1 and T2 are equally effective, this does not mean that they detect the same faults. T1 and T2 may find the same number of faults, but T1 may find faults of type A (for example, control faults) whereas T2 finds faults of type B (for example, assignation faults). This would provide a better understanding of technique complementarity, even when they are equally effective.

- An interesting question for further examination is the differences between programs for which different techniques yield different results. That is, given two programs P1 and P2, and two techniques T1 and T2 that behave differently with respect to P1, but equally with respect to P2 (either as regards the number of detected faults, the technique application time, etc.), identify what differences there are between these two programs. This could identify a possible relationship

Table 18. Conclusions for the regression family studies.

Technique	Practical recommendation	Maturity status	Pending knowledge
Regression	For small programs, it is preferable to use retest-all as opposed to deja-vu and test-tube, as it takes less time to select and run test cases. For big programs, it is preferable to use deja-vu or test-tube as opposed to retest-all, as it takes less time to select and run test cases. This is due to the fact that deja-vu and test-tube select fewer test cases than retest-all. It is preferable to use a safe technique (deja-vu, test-tube or retest-all) as opposed to random, as they have more relative effectiveness. The percentage of selected cases over initial set depends on the structure of the program, and its modifications for deja-vu and test-tube. It is not clear whether to use textual differencing as opposed to retest-all. Although textual differencing selects a lower percentage of test cases over the initial set than retest-all, there are no definite results regarding the time taken to run test cases. There are also no definite results for textual differencing and deja-vu.	• *Confirmed* with laboratory programs and faults. • *Pending* formal analysis. • *Pending* laboratory replication. • *Pending* field study.	Study absolute effectiveness of the techniques. • Study effectiveness of the techniques. • Study time to select and run test cases for each technique.

between program types or fault types and the techniques studied, which would help to define application conditions for these techniques. It would be a good idea to conduct a more detailed study of technique dependency on program type to be able to more objectively determine the benefits of each technique.

- As readers will be able to appreciate, the original intention of extracting empirically validated knowledge on testing techniques from this survey has been held back for several reasons. These reasons have been mentioned throughout the article and can be summarized globally as: dispersion of the techniques studied by the different papers within one and the same family.

- Dispersion of the response variables examined even for the same techniques.

Additionally, we have also found some limitations that prevent their results from being generalized. Most of the statements about the techniques are beleaguered by one or more of the following limitations which makes testing knowledge quite immature:

Table 19. Conclusions for the data-flow, control-flow and random families studies.

Technique	Practical recommendation	Maturity status	Pending knowledge
Data flow (all-uses. all-dus) vs. Control flow (all-edges) vs. Random	If *time is an issue*, the use of the random testing technique can be relied upon to yield an effectiveness similar to all-uses and all-edges (the differences being smaller as coverage increases) in 50% of the cases. Where testing needs to be exhaustive, the application of all-uses provides assurance, as, in the other half of the cases, this criterion yielded more efficient results thanks to the actual technique, unlike all-edges, which was more efficient because it generated more test cases.	• *Confirmed* with laboratory programs and faults. • *Confirmed* formally. • *Pending* laboratory replication. • *Pending* field study.	• Compare with other techniques of the family. • Use a better metric for effectiveness.
	High coverage levels are recommended for all-edges, all-uses and all-dus, but not for the random testing technique. Even when there is maximum coverage, however, there is no guarantee that a fault will be detected.	• *Confirmed* with laboratory programs and faults. • *Confirmed* by field study. • *Pending* formal analysis. • *Pending* laboratory replication.	• Find out whether the fact the maximum coverage does not detect a fault depends on the fault itself.
	All-edges should be applied together with all-dus, as they are equally effective and detect different faults. Additionally, they generate about the same number of test cases, and the random testing technique has to generate between 50% and 160% more test cases to achieve the same effectiveness as all-edges and all-dus.	• *Confirmed* with laboratory programs and faults. • *Confirmed* formally. • *Pending* laboratory replication. • *Pending* field study.	• Compare with other techniques of the family. • Use a better metric for effectiveness.

- Informality of the results analyses (many studies are based solely on qualitative graph analysis).
- Limited usefulness of the response variables examined in practice, as is the case of the probability of detecting at least one fault.
- Non-representativeness of the programs chosen, either because of size or the number of faults (one or two) introduced.
- Non-representativeness of the faults introduced in the programs (unreal faults).

Table 20. Conclusions for the mutation and data flow families studies.

Technique	Practical recommendation	Maturity status	Pending knowledge
Mutation (standard) vs. Data flow (all-uses)	*If high coverage is important and time is limited*, it is preferable to use all-uses as opposed to mutation, as it will be just as effective as mutation in about half of the cases.	• *Confirmed* with laboratory programs and faults. • *Pending* formal analysis. • *Pending* laboratory replication. • *Pending* field study.	• Find out whether the cases in which mutation is more effective than all-uses is due to the fault type. • Study the costs of both techniques in terms of application time. Use another more significant metric type to measure effectiveness.
	All-uses behaves similarly as regards effectiveness to abs/ror mutation and 10% mutation.		• Study the number of cases generated by the three alternatives.

Table 21. Conclusions for the functional and control flow families studies.

Technique	Practical recommendation	Maturity status	Pending knowledge
Functional (boundary value analysis) vs. Control flow (sentence coverage, decision coverage, branch testing)	For *experienced subjects* and when *there is time*, it is better to use the boundary value analysis technique as opposed to sentence coverage, as subjects will detect more faults, although it will take longer. On the other hand, for *inexperienced subjects* and when *time is an issue*, it is better to use sentence coverage as opposed to boundary value analysis, although there could be a loss of effectiveness. The time will also depend on the program.	• *Confirmed* with laboratory programs and faults. • *Confirmed* formally. • *Pending* laboratory replication. • *Pending* field study.	• Compare control flow testing techniques with each other.
	It is preferable to use boundary value analysis as opposed to condition coverage, as there is no difference as regards effectiveness and it takes less time to detect and isolate faults.		
	There appears to be a dependency on the subject as regards technique application time, fault detection and fault isolation.		• Check whether it is true for all techniques. • Further examine the combination of fault and failure.
	There appears to be a dependency on the program as regards the number and type of faults detected.		• Check whether it is true for all techniques. • Further examine the type of faults detected by each technique.
	More faults are detected by combining subjects than techniques of the two families.		• Check whether it is true for all techniques.
	If *control faults are to be detected*, it is better to use boundary value analysis or condition coverage than sentence coverage. Otherwise, it does not matter which of the three are used.		• Classify the faults to which the techniques are sensitive.
	It is impossible to ascertain whether boundary value analysis is more or less effective than branch testing, because effectiveness also depends on the program (fault).		

Table 22. Conclusions for the regression and improvement families studies.

Technique	Practical recommendation	Maturity status	Pending knowledge
Regression vs. regression + improvement	In the event of time constraints for running test cases, the use of prioritization can be relied upon to yield a higher fault detection speed than no prioritization.	• *Confirmed* with laboratory programs and faults. • *Confirmed* formal analysis. • *Pending* laboratory replication. • *Pending* field study.	It remains to study how long does it take to apply prioritization.
	The more faults there are in the program, the more effective minimization is.	• *Confirmed* with laboratory programs and faults. • *Pending* formal analysis. • *Pending* laboratory replication. • *Pending* field study.	
	In the event of time constraints, the use of regression based on modifications and minimization and prioritization can be relied upon to yield an effectiveness similar to regression based on modifications only. Also, a better recall is obtained.		It would be interesting to go into the issue that the number of selected test cases depends on the program in more depth.
	In the event of time constraints for running test cases, the use of minimization can be relied upon to yield a lower effectiveness than safe and data-flow (these two equal) and random. Effectiveness for these techniques depends on the program.		• It remains to study a quantification of the selection time. • Should provide quantitative values that can be used to extract numerical results as regards efficiency and cost of the techniques it studies.
	It is preferable to use fep techniques as opposed to sentence coverage technique, as they have a higher fault detection speed. However, when time is not an issue, it is preferable to use sentence coverage techniques as opposed to function coverage techniques, as they are more effective, but also more costly.		Study more in depth this difference.
	Function coverage techniques do not behave equally.		Study in depth of the differences among function coverage techniques.

After analyzing the empirical studies of testing techniques, the main conclusion is that more experimentation is needed and much more replication has to be conducted before general results can be stated. While it is true that this conclusion was to be expected, as experimental software engineering is not a usual practice in our field, more experimenters are needed, so that the ideas thrown into the arena can be corroborated and tested and then used reliably.

11. Conclusions

This paper has presented the current situation of knowledge on testing techniques. For this purpose, we have taken and analyzed a series of empirical studies on testing techniques. With the aim of simplifying the analysis of the knowledge derived from these studies, the studies were first classified according to the techniques they study and then according to the parameters addressed in the study. The analysis involved identifying the environment in which the study was performed in order to classify the knowledge acquired according to the different maturity levels proposed. Finally, the results of the analysis have been presented. The major conclusions are that our current testing technique knowledge is very limited. That is, there is at present no formally tested knowledge and over half of the existing knowledge is based on impressions and perceptions and, therefore, devoid of any formal foundation.

Notes

1. A path is a code sequence that goes from the start to the end of the program.
2. See also Weyuker (1988).
3. See also Offut and Lee (1991).
4. See also Offut et al. (1993).
5. A mutant is killed when a test case causes it to fail.
6. See also Frankl and Weiss (1991a,b).
7. See also Frankl et al. (1994).
8. See also Basili and Selby (1985) and Selby and Basili (1984).

References

Basili, V. R., and Selby, R. W. 1985. Comparing the effectiveness of software testing strategies. Department of Computer Science. University of Maryland. Technical Report TR-1501. College Park.

Basili, V. R., and 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*. International Thomson Computer Press, second edition.

Bible, J., Rothermel, G., and Rosenblum, D. 1999. A Comparative Study of Coarse- and Fine-Grained Safe Regression Test Selection. Computer Science Department, Oregon State University. Technical Report 99-60-05.

Bieman, J. M., and Schultz, J. L. 1992. An empirical evaluation (and specification) of the all-du-paths testing criterion. *Software Engineering Journal* January, 43–51.

Davis, A. 1993. *Software Requirements*: *Objects, Functions and States*. PTR Prentice Hall.

Elbaum, S., Mailshevsky, A. G., and Rothermel, G. 2000. Prioritizing test cases for regression testing. In *Proceedings of the ACM SIGSOFT 2000 International Symposium on Software Testing and Analysis*. Portland, Oregon, USA, August ACM, 102–112.

Frankl, P., and 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, 153–162.

Frankl, P. G., Weiss, S. N., and Hu, C. 1994. All-uses versus mutation: An experimental comparison of effectiveness. Polytechnic University, Computer Science Department. Technical Report. PUCS-94-100.

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

Frankl, P. G., and Weiss, S. N. 1991a. An experimental comparison of the effectiveness of the all-uses and all-edges adequacy criteria. *Proceedings of the Symposium on Testing, Analysis and Verification*. Victoria, BC, Canada, 154–164.

Frankl, P. G., and Weiss, S. N. 1991b. Comparison of all-uses and all-edges: design, data, and analysis. Hunter College, Computer Science Department. Technical Report. CS-91-03.

Frankl, P. G., and 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.

Graves, T. L., Harrold, M. J., Kim, J. M., Porter A., and Rothermel, G. 1998. An empirical study of regression test selection techniques. In *Proceedings of the 1998 International Conference on Software Engineering*. Kyoto, Japan. IEEE Computer Society Press, April, 188–197.

Hamlet, R. 1989. Theoretical comparison of testing methods. In *Proceedings of the ACM SIGSOFT '89 Third Symposium on Testing, Analysis and Verification*. Key West, Florida, ACM, 28–37.

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

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

Kim, J. M., Porter, A., and Rothermel, G. 2000. An empirical study of regression test application frequency. In *Proceedings of the 22nd International Conference on Software Engineering*. Limerick, Ireland: IEEE Computer Society Press, May, 126–135.

Myers, G. J. 1978. A controled experiment in program testing and code walkthroughs/inspections. *Communications of the ACM*. 21(9): 760–768.

Myers, G. J. 1979. *The Art of Software Testing*. Wiley-interscience.

Offut, A. J., Rothermel, G., and Zapf, C. 1993. An experimental evaluation of selective mutation. *Proceedings of the 15th International Conference on Software Engineering*. Baltimore, USA, IEEE, 100–107.

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

Offut, A. J., and Lee, D. 1991. How strong is weak mutation? *Proceedings of the Symposium on Testing, Analysis, and Verification*. Victoria, BC, Canada, ACM, 200–213.

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

Rothermel, G., Untch, R. H., Chu, C., and Harrold, M. J. 1999. Test case prioritization: An empirical study. In *Proceedings of the International Conference on Software Maintenance*. September, 179–188.

Rothermel, G., and Harrold, M. J. 1998. Empirical studies of a safe regression test selection technique. *IEEE Transactions on Software Engineering* 24(6): 401–419, June.

Selby, R. W., and Basili, V. R. 1984. Evaluating software engineering testing strategies. *Proceedings of the 9th Annual Software Engineering* Workshop. NASA/GSFC, Greenbelt, MD, 42–53.

Vokolos, F., and Frankl, P. G. 1998. Empirical evaluation of the textual differencing regression testing technique. In *Proceedings of the International Conference on Software Maintenance (ICSM-98)*. November, Bethesda, Maryland: IEEE Computer Society Press.

Weyuker, E. 1988. An empirical study of the complexity of data flow testing. *Proceedings 2nd Workshop on Software Testing, Verification and Analysis*. Banff, Canada, 188–195.

Weyuker, E. J. 1990. The cost of data flow testing: An empirical study. *IEEE Transactions on Software Engineering* 16(2): 121–128.

Wong, W. E., Horgan, J. R., London, S., and Agrawal, H. 1998. A study of effective regression testing in practice. In *Proceedings of the 8th International Symposium on Software Reliability Engineering* (ISSRE'97). Bethesda, Maryland: IEEE Computer Society Press, November, 264–274.

Wong, E., and 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., and Miller, J. 1997. Comparing and combining software defect detection techniques: A replicated empirical study. *Proceedings of the 6th European Software Engineering Conference*. Zurich, Switzerland.

Dr. Natalia Juristo is a professor of software engineering with the Computer Science School at the Technical University of Madrid (UPM) in Spain. Since 1992 she is the Director of the UPM MSc in Software Engineering. Dr. Juristo has a BS and a PhD in Computing from UPM. She was fellow of the European center for Nuclear Research (CERN) in Switzerland in 1988, and staff of the European Space Agency (ESA) in Italy in 1989 and 1990. During 1992 she was resident affiliate of the Software Engineering Institute. She was program chair for SEKE97 and general chair for SEKE01 and SNPD02. Prof. Juristo has been key speaker for CSEET03. She has been guest editor of special issues in several journals, including the Journal of Software and Systems, Data and Knowledge Engineering and the International Journal of Software Engineering and Knowledge Engineering Dr. Juristo has been member of several editorial boards, including IEEE Software and the Journal of Empirial Software Engineering. She is senior member of IEEE.

Dr. Ana M. Moreno is Associate Professor with the School of Computing at the Universidad Politecnica de Madrid. Dr. Moreno has a BS and a PhD in Computing. Since 2001 she is Director of the MSc in Software Engineering. Dr. Moreno has been visiting scholar at the Vrije Unviersiteit (Amsterdam, The Netherlands) and visiting professor at the Unviersity of Colorado at Colorado Springs (USA). She was program chair for NLDB'01 and SNPD'02 and general chair for CSEET03. She has been guest editor of a special issues in several journals including Data & Knowledge Engineering and International Journal of Software Engineering and Knowledge Engineering; and reviewer journals like ACM Computing Reviews, IEEE Transactions on Software Engineering, IEEE Computer or IEEE Software. In 2001 she has published a book titled ''Basics on Software Engineering Experimentation''.

Dr. Sira Vegas is assistant professor with the Computer Science School at the Universidad Politécnica de Madrid in Spain. She was Summer Student at the European Center for Nuclear Reseach (Geneva) in 1995. From 1998 to 2000 she has been a regular visiting scholar at the University of Maryland. In summer 2002 she has been research visitor at the Fraunhofer Institute for Experimental Software Engineering in Kaiserslautern (Germany). Sira has a BS and PhD in computer science from the Technical University of Madrid. She is a member of IEEE Computer Society and ACM.