# State of the Empirical Knowledge on Testing Techniques

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

Engineering disciplines are characterised by the use of mature knowledge by means of which they can achieve 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 analyses the maturity level of the knowledge about testing techniques by examining existing empirical studies about these techniques. For this purpose, we classify testing technique knowledge according to four categories.

Categories and Subject Descriptors: [Software Engineering] – Testing and debugging.
General Terms: Experimentation, Verification.
Additional Key Words and Phrases: Testing techniques, empirical software engineering.

## 1. INTRODUCTION

Engineering disciplines are characterised by using mature knowledge that can be applied to output predictable results. Latour and Woolgor [Latour and Woolgor 1986] discuss a series of intermediate steps on a scale that ranges from the most mature knowledge, considered as proven facts, to the least mature knowledge, composed of beliefs or speculations: facts given as founded and accepted by all, undisputed statements, disputed statements, and conjectures or speculations. The path from subjectivity to objectivity is paved by testing or empirical comparison with reality. Engineering disciplines apply knowledge composed of facts and undisputed statements in order to output products with predictable characteristics.

Unfortunately, software development has been characterised from its origins by a serious lack of empirical facts tested against reality that provide evidence of the advantages or disadvantages of using different methods, techniques or tools to build

Authors' addresses: Facultad de Informática. Universidad Politécnica de Madrid. Campus de Montegancedo, 28660 Boadilla del Monte, Madrid, Spain.

software systems. The knowledge used in our discipline can be considered to be relatively immature, and developers are guided by intuition, fashion or market-speak rather than by the facts or undisputed statements proper to an engineering discipline.

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 [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 analyse the maturity level of the knowledge in this area. For this purpose, we have surveyed the major empirical studies on testing in order to analyse 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 levels 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 uncontrolled 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*.

For this purpose, the paper has been structured as follows. Section 2 presents the chosen approach for grouping the different testing studies. Sections 3, 4, 5, 6 and 7 focus on each of the study categories described in section 2. 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 analysed in detail and, finally, the findings are summarised. Finally, section 8 outlines the practical recommendations that can be derived from these studies, along with knowledge maturity level, that is, how reliable these recommendations are. Section 8 also indicates what aspects should be addressed in future studies in order to increase the body of empirical knowledge on testing techniques.

The organisation of this chapter 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 8, which summarises 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 chapter, 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 8.

## 2. CLASSIFICATION OF TESTING TECHNIQUES

Software testing is the name that identifies a set of corrective practices (as opposed to the preventive practices applied during software construction), whose goal is to determine software systems quality. In testing, quality is determined by analysing the results of exercising the software (there is another type of corrective measures, known as static analysis, that examine the product under evaluation at rest).

Testing techniques determine different criteria for selecting the test cases that are to be run on the software system. These criteria can be used to group the testing techniques by families. Accordingly, techniques belonging to one and the same family are similar as regards the information they need to generate test cases (source code or specifications) or the aspect of code to be examined 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], [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 chapter, and the techniques of which they are composed, the information they require and the aspect of code they examine:

- *Random Testing Techniques*. The random testing techniques family is composed of the oldest and 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, shown in Table 1, are the most commonly used.

Table 1. Random techniques family.

| TECHNIQUE | TEST CASE GENERATION CRITERION |
|---|---|
| Pure random | Test cases are generated at random, and generation stops when there appear to be enough. |
| Guided by the 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 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 behaviour 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 behaviour. 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 behaviour. The techniques of which this family is composed (Table 2) differ from each other in terms of the rigorousness with which they cover the equivalence classes.

Table 2. Functional testing technique family.

| TECHNIQUE | TEST CASE GENERATION CRITERION |
|---|---|
| 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 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. Table 3 shows the techniques of which this family is composed, giving a brief description of the coverage criterion followed, in ascending order of rigorousness.

Table 3. Control flow testing technique family

| TECHNIQUE | TEST CASES GENERATION CRITERION |
|---|---|
| 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 Testing Techniques*. Data flow testing techniques also require knowledge of source code. The objective of this family is to select program paths to explore

[1] A path is a code sequence that goes from the start to the end of the program.

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. Table 4 reflects the techniques, along with their associated coverage criterion.

Table 4. Data flow testing techniques

| TECHNIQUE | TEST CASES GENERATION CRITERION |
|---|---|
| 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[2] 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. |

- *Mutation Testing Techniques.* Mutation testing techniques are based on modelling 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 modelled 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. Table 5 shows the techniques of which this family is composed and gives a brief description of the mutant selection criterion.

Table 5. Mutation testing technique family

| TECHNIQUE | TEST CASES GENERATION CRITERION |
|---|---|
| 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, 2, 4 or 6 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. |

Our aim is to review the empirical studies designed to compare testing techniques in order to identify the maturity level of their knowledge, based on the kind of empirical studies performed for getting that knowledge. We have grouped the empirical studies reviewed into several subsets taking into account which techniques they compare:

- *Intra-family studies*, which compare techniques belonging to the same family to find out the best criterion, that is, which technique of all the family members should be used. We have found:
  - Studies on the data flow testing techniques family
  - Studies on the mutation testing techniques family
- *Inter-family studies*, which study techniques belonging to different families to find out which family is better, that is, which type of techniques should be used. We have identified:
  - Comparative studies between the control flow and data flow testing techniques families
  - Comparative studies between the mutation and data flow testing techniques families
  - Comparative studies between the functional and control flow testing techniques families.

[2] There is said to be a c-use of a variable when the variable appears in a computation (right-hand side of an assignation). There is said to be a p-use of a variable when the variable appears as a predicate of a logical expression.

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 analyse the differences between the techniques within the data flow testing techniques family. Table 6 shows which aspects were studied for which testing techniques. The table can be read as follows: Weyuker analysed 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 studied the *number of test cases generated* for the all-du-paths technique alone.

Table 6. Studies on data flow testing techniques.

| | **STUDY** | [Weyuker 1990] | [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 |

Weyuker [Weyuker 1990] (see also [Weyuker 1988]) conducts a quantitative study to check the theoretical relationship of inclusion among the test cases generation criteria followed for each technique. This theoretical relationship can be represented as follows:

all-du-paths ⇒ all-uses

all-uses ⇒ all-c-uses

all-uses ⇒ all-p-uses

all-p-uses and all-c-uses cannot be compared

Which would read as follows. The test cases that comply with the all-du-paths criterion satisfy the all-uses criterion; the test cases that comply with the all-uses criterion satisfy the all-c-uses criterion, and so on. Weyuker's empirical results (obtained by studying twenty-nine programs taken from a book on Pascal with five or more decision sentences) reveal that the following, generally, holds:

all-uses ⇒ all-du-paths

all-p-uses ⇒ all-c-uses

all-p-uses ⇒ all-uses

So, the author establishes an inverse relationship with respect to the theory between: all-uses and all-du-paths and between all-p-uses and all-uses. That is, she concludes that, in practice, the test cases generated to meet the all-uses criterion, also normally comply with all-du-paths, and the test cases generated by all-p-uses also comply with all-uses.

According to these results, it would suffice with respect to criterion compliance to use all-uses instead of all-du-paths and all-p-uses instead of all-uses, as the test cases that meet one criterion will satisfy the other.

However, the number of test cases generated by each criterion needs to be examined to account for the cost (and not only the benefits) of these relationships. Analysing this variable, Weyuker gets the following relationship:

all-c-uses < all-p-uses < all-uses < all-du-paths

which would read as: more test cases are generated to comply with all-p-uses than to meet all-c-uses and fewer than to satisfy all-uses and all-du-paths. Bearing in mind the results concerning the number of generated test cases and criteria compliance, we could deduce that it is better to use all-p-uses than all-uses and it is better to use all-uses than all-du-paths, as the former generate fewer test cases and generally meet the other criterion.

With respect to all-c-uses, although it generates fewer test cases than all-p-uses, the test cases generated by all-c-uses do not meet the criterion of all-p-uses, which means that it does not yield equivalent results to all-p-uses.

Note that the fact that the set of test cases generated for one criterion is bigger than for another does not necessarily mean that the technique detects more faults, as defined in other studies examined later. And the same applies to the relationship of inclusion. The fact that a criterion includes another, does not say anything about the number of faults it can detect.

Another of Weyuker's results is that the number of test cases generated by all-du-paths, although exponential in theory, is in practice linear with respect to the number of program decisions. Bieman and Schultz [Bieman and Schultz 1992] partly corroborate these results using real industrial software system, deducing that the number of test cases required to meet this criterion is reasonable. Bieman and Schultz indicate that the number of cases in question appears to depend on the number of lines of code, but they do not conduct a statistical analysis to test this hypothesis, nor do they establish what relationship there is between the number of lines of code and the number of generated test cases.

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, using toy programs, and Bieman and Schultz, 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 generalised to real practice.
- Bieman and Schultz 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 and Bieman and Schultz 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 summarised in Table 7.

## 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 8.

Table 7. 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<br>- All-uses includes all-du-paths | - |
| | **Number of test cases generated** | - All-c-uses generates fewer test cases than all-p-uses<br>- All-p-uses generates fewer test cases than all-uses<br>- All-uses generates fewer test cases than all-du-paths<br>- 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<br>- 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.*<br>*- 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.*<br>*- 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.<br>- The results of Bieman and Schultz's study have to be corroborated using formal statistical analysis techniques.<br>- 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.<br>- 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. | | |

As shown in Table 8, the efficiency of these techniques is measured differently. So, whereas Offut and Lee [Offut and Lee 1994] (see also [Offut and Lee 1991]) and Offut *et al.* [Offut *et al.* 1996] (see also [Offut *et al.* 1993]) measure efficiency as the percentage of mutants killed by each technique, Wong and Mathur [Wong and Mathur 1995] measure it as the percentage of generated test set cases that detect at least one fault. On the other hand, all the studies consider the cost of the techniques identified as the number of generated test cases and/or the number of generated mutants.

The results of the three studies appear to corroborate each other as regards mutation being much more costly than any of its variants, while there does not appear to be too drastic a loss of effectiveness for the variants as compared with strong mutation.

After analysing 11 subroutines of no more than 30 LOC, Offut and Lee indicate in this respect that, for non-critical applications, it is recommendable to use weak as opposed to strong mutation, because it generates fewer test cases and kills a fairly high percentage of

Table 8. Studies on mutation testing techniques.

| | STUDY | [Offut and Lee 1994] | [Offut *et al.* 1996] | [Wong and Mathur 1995] |
|---|---|---|---|---|
| ASPECT STUDIED | % mutants killed[3] by each technique | X | X | |
| | No. of generated cases | X | X | |
| | No. of generated mutants | | X | X |
| | % generated sets that detect at least 1 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 | | |
| | 2-selective mutation | | O | |
| | 4-selective mutation | | O | |
| | 6-selective mutation | | O | |
| | Random selected 10% mutation | | | O |
| | Constrained (abs/ror) mutation | | | O |

mutants. In particular, they suggest that bb-weak-1 and st-weak kill a higher percentage of mutants, but they also generate more test cases.

Furthermore, Offut *et al*. analyse 10 programs (9 of which were studied in Offut and Lee, of no more than 48 LOC) and find that the percentage of strong mutation mutants killed by each selective variant is over 99% and is, in some cases, 100%. Therefore, the authors conclude that selective mutation is an effective alternative to strong mutation. Additionally, selective mutation cuts test costs substantially, as it reduces the number of generated mutants.

As regards Wong and Mathur, they compare strong mutation with two selective variants (randomly selected 10% mutation and constrained mutation, also known as abs/ror mutation). They find, on 10 small programs, that strong or standard mutation is equally as or more effective than either of the other two techniques. However, these results are not supported statistically, which means that it is impossible to determine whether or not this difference in effectiveness is significant.

Finally, Wong and Mathur refer to other studies they have performed, which determined that abs/ror mutation and 10% mutation generate fewer test cases than strong mutation. This gain is offset by a loss of less than 5% in terms of coverage as compared with strong mutation. This could mean that many of the faults are made in expressions

[3] A mutant is killed when a test case causes it to fail.

and conditions, which are the questions evaluated by abs/ror mutation. For this reason and for non-critical applications, they suggest the possibility of applying abs/ror mutation to get good cost/benefit performance (less time and effort with respect to a small loss of coverage).

In summary, 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 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 generalised to industrial cases, and a replication in this context is required to get greater results reliability.
- Offut *et al.* and Wong and Mathur 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.
- 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 9 shows the results of this set of studies.

## 5. COMPARATIVE STUDIES BETWEEN THE DATA FLOW, CONTROL FLOW AND RANDOM TESTING TECHNIQUES FAMILIES

The objective of this series of studies is to analyse 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 10 shows the studies considered, the aspects studied by each one and for which testing techniques.

Table 9. Results of the studies on mutation testing techniques

| | STUDY | [Offut and Lee 1994] | [Offut *et al.* 1996] | [Wong and Mathur 1995] |
|---|---|---|---|---|
| ASPECT STUDIED | **% 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<br>- 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)<br>- Abs/ror generates from 50 to 100 times fewer mutants than standard mutation |
| | **% sets generated that detect at least 1 fault** | - | - | - Standard mutation is more effective than 10% mutation in 90% of cases and equal in 10%<br>- Standard mutation is more effective than abs/ror in 40% of the cases and equal in 60%<br>- 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.*<br>*- 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.*<br>*- 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 7 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.<br>- The results of the studies by Offut *et al.* and Wong and Mathur should be corroborated using formal techniques of statistical analysis.<br>- It remains to compare the variants of strong mutation with each other.<br>- 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 10. Comparative studies of the data flow, control flow and random testing technique families.

| | STUDY | [Frankl and Weiss 1993] | [Hutchins *et al.* 1994] | [Frankl and Iakounenko 1998] |
|---|---|---|---|---|
| ASPECT STUDIED | Number of test cases generated | X | X | |
| | No. of sets with at least 1 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 |

Frankl and Weiss [Frankl and Weiss 1993] (see also [Frankl and Weiss 1991] and [Frankl and Weiss 1991]) and Frankl and Iakounenko [Frankl and Iakounenko 1998] study the effectiveness of the all-uses, branch testing and random testing techniques in terms of the probability of a set of test cases detecting at least one fault, measured as the *number of sets of test cases that detect at least one fault/total number of sets generated.*

Frankl and Weiss use nine toy programs containing one or more faults to measure technique effectiveness. The results of the study indicate that the probability of a set of test cases detecting at least one fault is greater (from a statistically significant viewpoint) for all-uses than for all-edges in five of the nine cases. Additionally, all-uses behaves better than random in six of the nine cases and all-edges behaves better than random testing in five of the nine cases.

Analysing the five cases where all-uses behaves better than all-edges, Frankl and Weiss find that all-uses provides a greater probability of a set of cases detecting at least one fault with sets of the same size in four of the five cases. Also, analysing the six cases where all-uses behaves better than random testing, the authors find that all-uses provides a greater probability of a set of cases detecting at least one fault in four of these cases. That is, all-uses has a greater probability of detecting a fault not because it works with sets containing more test cases than all-edges or random testing, but thanks to the very strategy of the technique. Note that the diffrence in the behaviour of the techniques (of nine programs, there are five for which a difference is observed and four for which none is observed for all-uses and six out of nine for random testing) is not statistically significant, which means that it cannot be claimed outright that all-uses is more effective than all-edges or random testing.

Analysing the five cases where all-edges behaves better than random testing, Frankl and Weiss find that in no case does all-edges provide a greater probability of a set of cases detecting at least one fault with sets of the same size. That is, in this case, all-edges has a greater probability of detecting a fault than random testing because it works with larger sets.

Frankl and Weiss also discern a relationship between technique effectiveness and coverage, but they do not study this connection in detail. Frankl and Iakounenko, however, do study this relationship and, as mentioned above, again define effectiveness as the probability of a set of test cases finding at least one fault, measured as the number of sets of test cases that detect at least one fault/total number of generated test cases. Frankl and Iakounenko deal with eight versions of an industrial program, each containing a real fault. Although the study data are not analysed statistically and its conclusions are based on graphical representations of the data, the qualitative analysis indicates that, as a general rule, effectiveness is greater when coverage is higher, irrespective of the technique. However, there are occasions where effectiveness is not 1, which means that some faults are not detected, even when coverage is 100%. This means that coverage increases the probability of finding a fault, but it does not guarantee that it is detected. Additionally, both all-uses and all-edges appear to behave similarly in terms of effectiveness, which is a similar result to what Frankl and Weiss found. For high coverage levels, both all-uses and all-edges behave much better than random testing. Indeed, Frankl and Weiss believe that the behaviour of random testing is unrelated to coverage. Hence, as random testing does not improve with coverage, it deteriorates with respect to the other two.

Note that even when technique coverage is close to 100%, there are programs for which the technique's fault detection effectiveness is not close to 1. This leads us to suspect that there are techniques that work better or worse depending on the fault type. The better techniques for a given fault type would be the ones for which effectiveness is 1, whereas the poorer ones would be the techniques for which effectiveness is not 1, even though coverage is optimum. However, Frankl and Weiss do not further research this relationship.

The study by Hutchins *et al.* [Hutchins *et al.* 1994] compares all-edges with all-dus and with random testing. As shown in Table 10, the authors study the *number of test cases generated* by each technique, the effectiveness of the techniques, again measured as

the number of sets that detect at least one fault/the total sets, as well as the relationship to coverage.

Hutchins *et al*. consider seven toy programs (with a number of lines of code from 138 to 515), of which they generate versions with just one fault. The results of the study show that the greater coverage is, the more effective the techniques are. While there is no evidence of a significant difference in effectiveness between all-edges and all-dus, there is for random testing.

Furthermore, the authors study the sizes of the test cases generated by all-edges and all-dus, and how big a set of cases generated using random testing would have to be for a given coverage interval to be equally effective. They reach the conclusion that the sizes generated by all-edges and all-dus are similar and that the increase in the size of one set of cases generated by random testing can vary from 50 to 160% for high coverages (over 90%).

The authors further examine the study, analysing the fault types detected by each technique. They find that each technique detects different faults, which means that although the effectiveness of all-edges and all-dus is similar, the application of one instead of the other is not an option, as they find different faults.

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 analyse 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.* 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 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 and Hutchins *et al.* use relatively simple, non-industrial programs, which means that the results cannot be directly generalised to real practice.
- Of the three studies, Frankl and Iakounenko 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 and Frankl and Iakounenko should also study technique complementarity (as in Hutchins *et al.*) , 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 summarised in Table 11.

Table 11. 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<br>- 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<br>- 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 1 fault/no. of sets generated** | - There is no convincing result regarding all-uses being more effective than all-edges and random:<br>• In approximately 50% of the cases, all-uses is more effective than all-edges and random, and all-edges is more effective than random<br>• 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<br>- 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)<br>- 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.*<br>*- 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.*<br>*- 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*. and Frankl and Ianounenko in industry.<br>- The results of the studies by Frankl and Weiss should be corroborated using formal techniques of statistical analysis.<br>- 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.<br>- 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. | | | |

## 6. 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 12.

Table 12. Comparative studies between the mutation and data flow testing techniques families.

| | **STUDY** | **[Frankl *et al.*1997]** | **[Wong and Mathur 1995]** |
|---|---|---|---|
| **ASPECT STUDIED** | % mutants killed by each technique | X | |
| | Ratio of generated sets that detect at least 1 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 |

Frankl *et al.* [Frankl *et al.* 1997] (see also [Frankl *et al.* 1994]) compare the effectiveness of mutation testing and all-uses. They study the ratio between the sets of test cases that detect at least one fault vs. total sets for these techniques. The effectiveness of the techniques is determined at different coverage levels (measured as the percentage of mutants killed by each technique). The results for 9 programs at high coverage levels with a number of faults of no more than 2 are as follows:

- Mutation is more effective for 5 of the 9 cases
- All-uses is more effective than mutation for 2 of the 9 cases
- There is no significant difference for the other two cases.

With regard to Wong and Mathur [Wong and Mathur 1995], they compare strong mutation, as well as two variants of strong mutation (randomly selected 10% and constrained mutation, also known as abs/ror mutation), with all-uses, studying again the ratio between the sets of test cases that detect at least 1 fault vs. total sets. For this purpose, the authors study 10 small programs, finding that the mutation techniques behave similarly to all-uses.

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.* 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 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 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 analyse 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 summarised in **¡Error! La autoreferencia al marcador no es válida.**.

## 7. COMPARISONS BETWEEN THE FUNCTIONAL AND CONTROL FLOW TESTING TECHNIQUES FAMILIES

The four studies of which this group is composed are reflected in Table 14. 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 analysis

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

Table 13. Comparisons between mutation and all-uses

| | STUDY | [Frankl *et al* 1997] | [Wong and Mathur 1995] |
|---|---|---|---|
| ASPECT STUDIED | **% 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 1 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%<br>- Abs/ror is more effective than all-uses in 50% of the cases, equally effective in 30% and less effective in 20%<br>- 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.*<br>*- 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.<br>- 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.<br>- 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.<br>- The cost of technique application should be studied. | | |

In Myers' study [Myers 1978], inexperienced subjects choose to apply one control flow and one functional testing technique, which they apply to a program taken from a programming book, analysing the variables: number of faults detected, time to detect faults, time to find a fault/type, number of faults detected combining techniques, and time taken to combine techniques/fault type.

Myers does not specify which particular techniques were used, which means that this study does not provide very practical results. One noteworthy result, however, is that the author does not find a significant difference as regards the number of faults detected by both technique types. However, the author indicates that different methods detect some fault types better than others (although this study is not performed statistically).

Table 14. Comparative studies of functional and control testing techniques.

| | STUDY | [Myers 1978] | [Basili and Selby 1987] | [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 |

Myers also studies fault detection efficiency combining the results of two different people. Looking at Table 14, we find that Wood *et al.* [Wood *et al.* 1997] also address this factor. The conclusions are similar in the two studies, that is, more faults are detected combining the faults found by two people. However, there are no significant differences between the different technique combinations.

Of the studies in Table 14, we find that Basili and Selby [Basili and Selby 1987] (see also [Basili and Selby 1985] and [Selby and Basili 1984]) and Wood *et al.* [Wood *et al*. 1997] use almost the same response variables: number of detected faults, percentage of detected faults, time to detect faults, number of faults detected per hour and percentage of faults detected per hour for Basili and Selby and number of detected faults, number of faults detected combining techniques, number of faults detected per hour, and percentage of detected faults for Wood *et al*.

Apart from these results, Kamsties and Lott [Kamsties and Lott 1995] also take an interest in the faults that cause the different failures, studying another set of variables, as

shown in Table 14: number of faults isolated per hour, percentage of faults isolated per type, time to isolate faults, percentage of faults isolated and total time to detect and isolate faults, but also time to detect faults, number of faults detected per hour, percentage of faults detected by type and percentage of detected faults.

Whereas Basili and Selby replicate the experiment with experienced and inexperienced subjects (two and one replications, respectively), Wood *et al*., like Kamsties and Lott, use only inexperienced subjects.

This means that Basili and Selby can further examine the effect of experience on the fault detection rate (number of faults detected per hour) or the time taken to detect faults. As regards the first variable, the authors indicate that the fault detection rate is the same for experienced and inexperienced subjects for both techniques (boundary value analysis and sentence coverage), that is, neither experience nor the technique influences this result. With respect to time, Basili and Selby indicate that the experienced subjects take longer to detect a fault with using the functional technique than with sentence coverage. This means that experienced subjects detect fewer faults with the structural technique than with the functional testing technique within a given time. For inexperienced subjects, on the other hand, the findings are inconclusive, as the results of the replications are not the same (in one replication no differences were observed between the techniques and in the other, the functional testing technique took longer to detect faults).

Also as regards time, the study by Kamsties and Lott (who, remember, worked with inexperienced subjects) indicates that the total time to detect and isolate faults is less using the functional testing technique than with condition coverage. As these authors studied the time to detect and isolate faults separately, the authors were able to determine statistically that it takes longer to isolate the fault using the functional technique than with condition coverage, but the time to detect the fault is less. Note that this result cannot be directly compared with the findings of Basili and Selby, where the functional technique did not take less time to detect faults, as the two consider different structural testing techniques: sentence coverage (Basili and Selby) and condition coverage (Kamsties y Lott).

As regards efficiency, Kamsties and Lott indicate that the fault detection rate was greater for the functional testing technique than for condition coverage.

Kamsties and Lott note that there were no significant differences between the percentage of isolated and detected faults, that is, both techniques behaved similarly, because the program was the influential factor. This result was corroborated by studies by

Basili and Selby and Wood *et al.*, who claim that the percentage of detected faults depends on the program and, according to Wood *et al.*, more specifically, on the faults present in these programs.

Basili and Selby and Kamsties and Lott have also studied the percentage of faults detected by the techniques according to fault type. In this respect, whereas Basili and Selby claim that the functional technique detects more control faults than sentence coverage, Kamsties and Lott indicate that, generally, there are no significant differences between the functional testing technique and condition coverage with regard to the percentage of isolated and detected faults by fault type.

Finally, it should be mentioned that Wood *et al*. also focus on the study of the number of detected faults using each technique individually and combining the results of two people applying the same or different techniques. Individually, they reach the conclusion that it is impossible to ascertain which technique is more effective, as the program (fault) is also influential. On the other hand, they find that the number of different faults detected is higher combining the results of different people, instead of considering only the results of the individual application of each technique. However, a formal analysis of the data could show that there is no significant difference between two people applying the same or different techniques, which might suggest that it is the people and not the techniques that find different faults (although this claim would require further examination).

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, Kamsties and Lott and Wood *et al.* find effectiveness-related differences between functional and structural techniques depending on the program to which they are applied. Wood *et al.* 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 and Myers 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 point out

that the functional technique detects more control faults. Myers also discerns a difference as regards different faults, but fails to conduct a statistical analysis. Finally, Kamsties and Lott 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 generalise the results obtained.
- Finally, it would be recommendable to unify the techniques under study in future replications to be able to generalise 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 generalising 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, Kamsties and Lott and Wood *et al.* 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 summarised the results of this group in Table 15.

Table 15. 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<br>- Inexperienced subjects:<br>• In one case, there is no difference between structural and functional techniques<br>• In the other, the functional technique detects more faults than the structural technique | - | The number of detected faults depends on the program/technique combination |
| | **% faults detected** | - Experienced subjects: The functional technique detects more faults than the structural technique<br>- Inexperienced subjects:<br>• In one case, there is no difference between the structural and functional techniques<br>• 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 |
| | **% faults detected/type** | - Boundary value analysis detects more control faults than sentence coverage<br>- 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<br>- Inexperienced subjects: Boundary value analysis takes as long or longer than sentence coverage | - Inexperienced subjects:<br>• Boundary value analysis takes less time than condition coverage<br>• The time taken to faults also depends on the subject | - |

| | STUDY | [Basili and Selby 1987] | [Kamsties and Lott 1995] | [Wood *et al.* 1997] |
|---|---|---|---|---|
| | **No. faults detected/hour** | - The fault rate with boundary value analysis and sentence coverage does not depend on experience<br>- 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 |
| | **% 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 | - |
| | **% 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<br>- 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.*<br>*- 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.*<br>*- There appears to be a dependency on the subject as regards technique application time, fault detection and fault isolation.*<br>*- There appears to be a dependency on the program as regards the number and type of faults detected.*<br>*- More faults are detected by combining subjects than techniques of the two families.*<br>*- 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.*<br>*- 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.<br>- 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. | | | |

## 8. DISCUSION ON TESTING TECHNIQUE KNOWLEDGE MATURITY LEVEL

Despite the difficulties encountered, **practitioners** can find interesting recommendations in Table 16, Table 17, Table 18, Table 19 and Table 20. 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 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 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.

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.*<br>- ***Confirmed*** *formally.* | - Find out the difference in terms of effectiveness between all-c-uses, all-p-uses, all-uses and all-du-paths.<br>- Compare with the other techniques in the family. |
| | 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.*<br>- ***Pending*** *field study.* | - 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.*<br>- ***Confirmed*** *with field study.*<br>- ***Pending*** *formal analysis.*<br>- ***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. | - ***Confirmed*** *with laboratory programs and faults.*<br>- ***Confirmed*** *formally.*<br>- ***Pending*** *laboratory replication.*<br>- ***Pending*** *field study.* | - Compare the different mutation variants with each other.<br>- Use another metric type for effectiveness, as the number of mutants killed by a technique is only useful for relative comparisons between mutation techniques. |
| | 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. | - ***Confirmed*** *with laboratory programs and faults.*<br>- ***Pending*** *formal analysis.*<br>- ***Pending*** *laboratory replication.*<br>- ***Pending*** *field study.* | |
| | 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 7 points. If time is not a critical factor, it is preferable to use standard mutation. | | |

Table 18. 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.*<br>- ***Confirmed*** *formally.*<br><br>- ***Pending*** *laboratory replication.*<br>- ***Pending*** *field study.* | - Compare with other techniques of the family.<br>- 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.*<br>- ***Confirmed*** *by field study.*<br><br>- ***Pending*** *formal analysis.*<br>- ***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.*<br>- ***Confirmed*** *formally.*<br><br>- ***Pending*** *laboratory replication.*<br>- ***Pending*** *field study.* | - Compare with other techniques of the family.<br>- Use a better metric for effectiveness. |

Table 19. 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.*<br>- ***Pending*** *formal analysis.*<br>- ***Pending*** *laboratory replication.*<br>- ***Pending*** *field study.* | - Find out whether the cases in which mutation is more effective than all-uses is due to the fault type.<br>- Study the costs of both techniques in terms of application time.<br>- 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 20. 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.*<br>- ***Confirmed*** *formally.*<br><br>- ***Pending*** *laboratory replication.*<br>- ***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. |
| | There appears to be a dependency on the program as regards the number and type of faults detected. | | - Further examine the combination of fault and failure.<br>- Check whether it is true for all techniques. |
| | More faults are detected by combining subjects than techniques of the two families. | | - Further examine the type of faults detected by each technique.<br>- 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). | | |

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 summarised 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 generalised. Most of the statements about the techniques are beleaguered by one or more of the following limitations which makes testing knowledge quite immature:

- 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).

After analysing 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.

## BIBLIOGRAPHY


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.* Pages 1278-1296. SE-13 (12).

BEIZER, B. 1990. *Software Testing Techniques*. International Thomson Computer Press, second edition.

BIEMAN, J.M., and SCHULTZ, J.L. 1992. An Empirical Evaluation (and specification) of the All-du-paths Testing Criterion. *Software Engineering Journal.* Pages 43-51, January.

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

FRANKL, P. and IAKOUNKENKO, O. 1998. Further Empirical Studies of Test Effectiveness. In *Proceedings of the ACM SIGSOFT International Symposium on Foundations on Software Engineering*, pages 153-162, Lake Buena Vista, Florida, USA.

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*. Volume 38. Pages 235-253. September.

FRANKL, P.G., and WEISS, S.N. 1991. An Experimental Comparison of the Effectiveness of the All-uses and All-edges Adequacy Criteria. *Proceedings of the Symposium on Testing, Analysis and Verification*. Pages 154-164. Victoria, BC, Canada.

FRANKL, P.G. and WEISS, S.N. 1991. 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*. Volume 19 (8). Pages 774-787.

HAMLET, R. 1989. Theoretical Comparison of Testing Methods. In *Proceedings of the ACM SIGSOFT '89 Third Symposium on Testing, Analysis and Verification*. Pages 28-37, Key West, Florida, ACM.

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.* Pages 191-200. Sorrento, Italy. IEEE.

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.

LATOUR, B., and WOOLGOR, D. 1986. *Laboratory Life. The Construction of Science Facts*. Princeton, USA: Princeton University Press.

MYERS, G.J. 1978. A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections. *Communications of the ACM*. Vol. 21 (9). Pages 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*. Pages 100—107. Baltimore, USA. IEEE.

OFFUT, A.J., LEE, A., ROTHERMEL, G., UNTCH, RH., and ZAPF, C. 1996. An Experimental Determination of Sufficient Mutant Operators. *ACM Transactions on Software Engineering and Methodology.* Volume 5 (2). Pages 99-118.

OFFUT, A.J., and LEE, D. 1991. How Strong is Weak Mutation?. *Proceedings of the Symposium on Testing, Analysis, and Verification*. Pages 200—213. Victoria, BC, Canada. ACM.

OFFUT, A.J., and LEE, S.D. 1994. An Empirical Evaluation of Weak Mutation. *IEEE Transactions on Software Engineering.* Vol. 20(5). Pages 337—344.

SELBY, R.W., and BASILI, V.R. 1984. Evaluating Software Engineering Testing Strategies. *Proceedings of the 9th Annual Software Engineering* Workshop. Pages 42—53. NASA/GSFC, Greenbelt, MD.

WEYUKER, E. 1988. An Empirical Study of the Complexity of Data Flow Testing. *Proceedings 2nd Workshop on Software Testing, Verification and Analysis*. Pages 188—195. Banff, Canada.

WEYUKER, E.J. 1990. The Cost of Data Flow Testing: An Empirical Study. *IEEE Transactions on Software Engineering.* Volume 16 (2). Pages 121—128.

WONG, E., and MATHUR, A.P. 1995. Fault Detection Effectiveness of Mutation and Data-flow Testing. *Software Quality Journal*. Volume 4. Pages 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.