Comparing the Effectiveness of Equivalence Partitioning, Branch Testing and Code Reading by Stepwise Abstraction Applied by Subjects

N. Juristo[1], S. Vegas[1], M. Solari[2], S. Abrahao[3], I. Ramos[4]
[1] Universidad Politécnica de Madrid, [2]Universidad ORT Uruguay,
[3]Universidad Politécnica de Valencia, [4]Universidad de Sevilla
[1]{natalia,svegas}@fi.upm.es, [2]martin.solari@ort.edu.uy, [3]sabrahao@dsic.upv.es, [4]iramos@us.es

Abstract— **Some verification and validation techniques have been evaluated both theoretically and empirically. Most empirical studies have been conducted without subjects, passing over any effect testers have when they apply the techniques. We have run an experiment with students to evaluate the effectiveness of three verification and validation techniques (equivalence partitioning, branch testing and code reading by stepwise abstraction). We have studied how well able the techniques are to reveal defects in three programs. We have replicated the experiment eight times at different sites. Our results show that equivalence partitioning and branch testing are equally effective and better than code reading by stepwise abstraction. The effectiveness of code reading by stepwise abstraction varies significantly from program to program. Finally, we have identified project contextual variables that should be considered when applying any verification and validation technique or to choose one particular technique.**

Keywords-Verification and validation, experimentation, combination of experimental results.

I. INTRODUCTION

Verification and validation (V&V) is an important but expensive part of the software development process. There are a wide range of V&V techniques, each with different features. It is worthwhile exploring the pros and cons of the techniques, that is, which techniques are better suited for particular software project characteristics, for example, for revealing a particular fault type, for application by a particular developer profile or for a particular software type. These strengths and weaknesses depend on the variables affecting technique effectiveness [4].

Several empirical studies have looked at the behavior of V&V techniques. In most studies, subjects did not apply the techniques, as we will see in the related work section. So far then, the onus of the research has been on examining what we might call *pure* technique behavior. We think that studies should address not only *pure* technique behavior but also the influence of the human factor on technique application. This is especially important for V&V techniques that have not been fully automated.

Most empirical studies with subject participation focus on static techniques [1], [7], [12], [33], [40], [48]. Only a few studies take dynamic techniques into consideration [2], [8], [28], [34], [42]. As a result of differences in contextual conditions and a low rate of internal and external replication, results are extremely immature. For this reason, we cannot differentiate fortuitous events from regular patterns.

In this paper, we examine the effectiveness of three V&V techniques —two dynamic techniques (equivalence partitioning and branch testing) and one static technique (code reading by stepwise abstraction)— applied to three different programs. We have chosen these techniques for two reasons. We wanted: 1) to compare techniques with a very different approach to software V&V, and 2) to use formal enough techniques to guarantee that the results of subjects applying the same technique were comparable. We have replicated the experiment eight times at four different sites. We have found that equivalence partitioning and branch testing are equally effective and better than code reading by stepwise abstraction. Additionally, the effectiveness of code reading by stepwise abstraction varies significantly from program to program. Finally, we have identified some key project contextual variables that influence technique effectiveness.

The paper is organized as follows. Section 2 describes work related to our research. Section 3 explains the research method. Section 4 describes the experiment and replications. Sections 5 and 6 present the results. Section 7 outlines the findings. Finally, Section 8 presents the conclusions.

II. RELATED WORK

There are several recent studies of V&V technique behavior. We divide these studies into three categories: theoretical studies, empirical studies with subjects, and empirical studies without subjects.

The goal of theoretical studies is to examine unadulterated techniques from the viewpoint of logic and deductive reasoning. They analyze techniques based on their theoretical groundwork, studying the effectiveness of code-based [9], [10], [13], [19], [23], [24], [36], [37], [41], [51], [54], [58] or regression techniques [43], [45].

Empirical studies without subjects aim to investigate techniques from the angle of practice and inductive reasoning by simulating technique application. They address different aspects of V&V techniques [27]. Some examine test-case generation and compare the efficiency and effectiveness of specification-based, code-based, and fault based techniques [6], [25], [38], [39], [52], [56]. Others evaluate test sets according to different criteria (data and control flow, mutation, and their variants), measure the criteria coverage level (adequacy) and relate criteria coverage and test case size to effectiveness [15], [16], [17], [18], [30], [53]. Finally, some studies evaluate test case selection approaches (regression techniques [5], [46], [50] filtering [21], [29], [31] or prioritization [14], [44], [47], [55]). Do *et al.* have built an infrastructure to help researchers to run such empirical studies [11].

Empirical studies with subjects take into account how the subject influences technique behavior. Most of these studies evaluate static techniques [1], [7], [12], [33], [40], [48]. Studies evaluating dynamic techniques have addressed test-case generation, comparing the efficiency and effectiveness of specification-based and code-based controlflow techniques applied by subjects [2], [8], [28], [34], [42].

Empirical studies with and without subjects are necessary. Studies with subjects are closer to real situations where techniques are applied by testers, but, they are unable to separate actual technique performance from the influence that the subject might be having on technique effectiveness.

III. Research Method

We have conducted empirical studies with subjects to further our knowledge of three code evaluation techniques. We have run a series of experiment replications. The results of a single experiment could be a mere one-off coincidence. An experiment needs to be repeated to check whether the observed results follow a regular pattern and are reproducible [20].

We have replicated the experiment eight times. Five replications were run at the Universidad Politécnica de Madrid (UPM01, UPM02, UPM03, UPM04 and UPM05, respectively). Additionally, the experiment was conducted at another three different sites: Universidad de Sevilla in Spain (UdS05), Universidad Politécnica de Valencia in Spain (UPV05) and the Universidad ORT in Uruguay (ORT05).

The number of subjects in the experiment replications was 42, 39, 29, 35 and 31 at UPM; 31 at UPV; 172 at UdS; and 76 at ORT.

Additionally, we have held meetings with the researchers present during the experimental operation at each site to discuss deviations from the planned execution. These meetings have helped us to better understand the conditions under which subjects apply the techniques.

To synthesize the results of the eight replications, we first examine the combined results of all the replications. We identify the significance level of the factors (and interactions), and we examine the behavior of the significant factors identified in each replication against the response variable means. To do this, we may need to re-examine some of the results analyzed in each separate experiment or run alternative analyses. Second, we study contextual differences among the replication sites. Finally, we integrate the results of the two stages.

IV. Description of the Experiment

A. Experimental Design

Our experiment and its replications uses the experiment replication packages built by Kamsties and Lott [28], and Roper and colleagues [42], although the material is adapted to our experimental goals and context. The null hypothesis of the experiment is:

H_0: *There is no difference in the effectiveness of equivalence partitioning, branch testing and code reading by stepwise abstraction.*

For equivalence partitioning and branch testing, we measure the response variable **effectiveness** as the *percentage of subjects that are able to generate a test case that uncovers the failure associated with a given fault.* For code reading by stepwise abstraction, we measure effectiveness as the *percentage of subjects that report a given fault.*

The experiment has a factorial design [26]. The **technique** is an experiment factor with three treatments (or levels): *equivalence partitioning* (EP), *branch testing* (BT) and *code reading by stepwise abstractio*n (CR). There follows a brief reminder of these techniques:

- Test case design by *equivalence partitioning*[1] [35] is a two-step procedure: (1) identify the equivalence classes and (2) define the test cases.
  Equivalence classes are identified by taking each input condition (usually a sentence or statement in the specification) and partitioning it into two or more groups. There are two types of equivalence classes: valid classes represent valid program inputs, and invalid classes represent erroneous input values. Both valid and invalid equivalence classes are identified according to a set of predefined heuristics based on whether the input condition specifies a range of values, a number of values, a set of input values or a "must be" situation. If there is any reason to believe that the program does not handle elements in an equivalence class identically, the equivalence class should be split into smaller equivalence classes.
  The second step is to use equivalence classes to identify test cases. Test cases that cover as many of the uncovered valid equivalence classes as possible are written until all valid equivalence classes have been covered by test cases. New test cases that cover one, and only one, of the uncovered invalid equivalence classes are written until all invalid equivalence classes have been covered by test cases.
- White-box testing is concerned with the extent to which test cases exercise the program logic. One possible logic-coverage criterion is *decision coverage* or *branch testing.* This criterion states [3] that enough test cases must be written to assure that every branch alternative is exercised at least once. If enough tests are run to meet this

[1] Some authors believe the term is a misnomer and the technique should be named just *partitioning*. We use the term *equivalence partitioning* in this paper, as it is the name used in the referenced book that we use to teach the technique to the students.

prescription then 100% branch coverage is achieved.
The test case design strategy is as follows. First, an initial test case is generated that corresponds to the simplest, functionally sensible entry/exit path[2]. Extra test cases are then generated. They should differ slightly from previous paths. Some paths should be favoured over others: paths that do not have loops over paths that do, short paths over long paths, simple paths over complicated paths, and paths that make sense over paths that do not. As the test cases are generated, a table showing the coverage status of each decision is built.

- The process of reading a program bottom up is called *code reading by stepwise abstraction* [32]. Subjects begin at the lowest (most detailed) level and replace each prime (consecutive lines of code) in the program by an abstraction (or specification) that summarizes its possible outcomes, irrespective of its internal control structure and data operations. Intermediate abstractions are successively discovered at higher levels by using those already found. Subjects repeat the process until they have abstracted all of the source code. Next, subjects compare the program specification with their abstraction (own specification) to observe inconsistencies between specified and expected program behavior.

The experiment uses three different **programs**, written in C:

- *cmdline* is a parser that reads the input line and outputs a summary of its contents. It has 209 LOC and a cyclomatic complexity of 61.
- *nametbl* implements the data structure and operations of a symbol table. It has 172 LOC and a cyclomatic complexity of 29.
- *ntree* implements the data structure and operations of an n-ary tree. It has 146 LOC and a cyclomatic complexity of 21.

Each program contains seven different **faults**. They are defined according to the six fault types identified in [2]. Each fault type is classed as *omission* (something that is missing) or *commission* (something that is incorrect). The fault types used in our experiment are:

- *Initialization*. Incorrect initialization of a data structure. For example, assigning an incorrect value to a variable when entering a module is a commission fault, whereas failure to initialize when necessary is an omission fault. Our experiment covers initialization faults of both commission (F4) and omission (F3).
- *Control*. The program follows an incorrect control flow path in a given situation. For example, an incorrect predicate in an if-then-else sentence is a commission fault, whereas a missing predicate is an omission fault. Our experiment covers control faults of both commission (F5) and omission (F6).
- *Computation*. These faults lead to an incorrect calculation. For example, an incorrect arithmetic operator on the right-hand side of an assignation is a commission fault. The experiment will cover computation faults of commission (F7).
- *Cosmetic*: Commission faults can result, for example, in a spelling mistake in an error message. Omission faults are faults where an error message should and does not appear. The experiment covers cosmetic faults of both commission (F2) and omission (F1).

[2] Sequence of instructions or statements that starts at the routine's entry point and ends at its exit. A path may pass through several junctions, processes, or decisions, one or more times.

We use two **versions** of each program in order to replicate every fault twice for each program. The programs are small in size, and we cannot seed as many faults as we would like to without violating the fault masking premise. The only difference between two versions of the same program is the fault instance that they contain. The programs always contain the same number and type of faults. All defects have a 100% probability of being detected by all three techniques.

The **experimental procedure** consists of several sessions. Subjects apply every technique once. Subjects work on every program once. Therefore, a subject applies each technique to one program. Subjects perform the following tasks:

- Subjects applying equivalence class partitioning are given the program specification to design test cases. Afterwards, they are given an executable version of the program and they run test cases. Subjects identify failures in terms of incorrect outputs.
- Subjects are expected to get as close as possible to 100% branch coverage. They are given the source code without its specification to design test cases. Afterwards, they are given an executable version of the program to run the test cases. They are then given the program specification to check the test case outputs. Subjects identify failures in terms of invalid outputs.
- Subjects reading the code by stepwise abstraction are given the program source code to identify abstractions and generate a program specification. They are then given the original specification to identify the faults in the program. Subjects identify faults from inconsistencies between the abstracted and original specifications.

B. *Description of the Replications*

Each site has some contextual conditions which affect experimental design. TABLE I shows each site's contextual conditions, along with the design decisions. There are some minor differences among the replications at each site:

- **UPM:** As there are no time constraints, the experiment examines all three techniques and uses all three programs. As there are enough computers for all subjects, three sessions are held where subjects individually apply the techniques and, for dynamic techniques, run the test cases. Subjects apply one technique to one program in each session. Subjects apply different techniques in each session. This way, all six possible technique application orders (EP-BT-CR, EP-CR-BT, BT-EP-CR, BT-CR-EP, CR-EP-BT and CR-BT-EP) are covered across the three sessions. All three techniques are exercised during each session, the techniques being applied to the same program. Notice that the day and program are confounded. For this reason, subjects apply programs in different orders in UPM replications (cmdline-ntree-nametbl in UPM01, UPM02 and UPM04, and ntree-cmdline-nametbl in UPM03 and UPM05). Each session has a four-hour duration. This is equivalent to there being no time limit, as it does not take subjects as long to complete the task. Subjects receive three four-hour training sessions to learn how to apply the techniques before the experiment is run.

TABLE I. EXPERIMENT ADAPTATION TO SITES.

VARIABLE	CONDITION	SITE	DESIGN DECISION
Time	Unconstrained	UPM, UdS	3 techniques and 3 programs
	Constrained	UPV, ORT	2 techniques, and 2 or 3 programs
Computer availability	Yes	UPM, UPV	Work individually
	Limited	UdS	Work in pairs
	No	ORT	No test cases run
Subject profile	Familiar with technique	UPV, UdS	Refresher tutorial
	Unfamiliar with the technique	UPM, ORT	Full course on techniques
Training sequence	Sequential	UPM, ORT	First training, then experiment
	Interleaved	UPV, UdS	Training interleaved with experiment
Session length	Unlimited	UPM, ORT	Test cases run with technique application. Application of all techniques in 1 session by different subjects
	Limited	UPV, UdS	Test cases run in a separate session for only 1 program

- **UdS:** As there are no time constraints, the experiment examines all three techniques and uses all three programs. The experiment is run in four two-hour sessions, because session length is limited. During the first three sessions, subjects apply one technique to one program. Subjects work in pairs as there are not enough computers for all subjects. Subjects apply the same technique to different programs in each session, since training has to be interleaved with experiment operation. As session length is limited, subjects run test cases for one of the programs that they have tested with a dynamic technique in the fourth session. As subjects are already acquainted with the techniques, training consists of three brief two-hour tutorials. Each tutorial is held before technique application.
- **UPV:** As there are time constraints, code reading by stepwise abstraction is omitted, but all three programs are used. The experiment is run in three two-hour sessions because session length is limited. As there are enough computers, subjects individually apply one technique to one program during the first two sessions. As training has to be interleaved with experiment operation, subjects apply the same technique to different programs in each session. Subjects run test cases for one of the programs that they have tested with a dynamic technique in session 3. As subjects are already acquainted with the techniques, training consists of two brief two-hour tutorials. Each tutorial is held before technique application.
- **ORT:** As there are time constraints, code reading by stepwise abstraction and the ntree program are omitted. The experiment is run in one session. As session length is unlimited, subjects individually apply the two techniques to the two programs. Subjects apply techniques to different programs and in different orders to cover all four possible application orders (EP/cmdline-BT/nametbl, EP/nametbl-BT/cmdline, BT/cmdline-EP/nametbl, and BT/nametbl-EP/cmdline). Subjects never run test cases, since they do not have access to computers. Subjects receive three four-hour training sessions to learn how to apply the techniques before the experiment is run. Subjects are not very skilled with programming issues.

TABLE II gives a brief description of each replication.

TABLE II. DESCRIPTION OF EACH REPLICATION.

Site	UPM	UdS	UPV	ORT
Techniques	EP, BT, CR	EP, BT, CR	EP, BT	EP, BT
Programs	cmdline nametbl ntree	cmdline nametbl ntree	cmdline nametbl ntree	cmdline nametbl
Subjects	5th-year computing students	5th-year computing students	5th-year computing students	2nd-year computing students
Technique application	Individual	Pairs	Individual	Individual
Training	Full course on techniques	Refresher tutorial	Refresher tutorial	Full course on techniques
Sessions	3	4	3	1
Techniques & programs per session	1 program 3 techniques	1 technique 3 programs	1 technique 3 programs	2 techniques 2 programs
Session workload	1 technique	1 technique	1 technique	2 techniques
Test case execution	Same session	Separate session	Separate session	None

SPSS statistical package has been used for the data analysis. We use analysis of variance (ANOVA) to analyze the data collected in the experiment [26]. We want to find out if a given factor (or combination of factors) is significant at a 0.05 significance level. We can apply ANOVA because the samples of all the replications meet the requirements of normality and homogeneity of variances[3]. We use the Bonferroni multiple comparison test to study the effect of the factor levels on the response variable at a 0.05 significance level. Finally, based on the results of the five UPM replications (where there were no contextual variations), we set a reference value for the mean and standard deviation to determine whether the values of a replication are abnormally high/low. As with outlier detection [57], values that do not fall within the range of the mean ±3 standard deviations are considered to be different (higher/lower) from the reference value.

[3] The complete results of the statistical analyses of can be found at http://www.grise.upm.es/sites/extras/4.

TABLE III. ANOVA SIGNIFICANCE LEVELS (P-VALUES) FOR ALL REPLICATIONS.

	UPM 01	UPM 02	UPM 03	UPM 04	UPM 05	UdS 05	UPV 05	ORT 05
Model	0.000	0.005	0.004	0.014	0.002	0.008	0.036	0.004
Model power	1.000	0.993	0.995	0.979	0.998	0.982	0.862	0.992
Program	0.041	0.390	0.163	0.907	0.002	0.009	0.026	0.003
Technique	0.000	0.000	0.000	0.000	0.000	0.000	0.765	0.003
Version	0.153	0.095	0.123	0.323	0.361	0.070	0.387	0.635
Fault	0.071	0.316	0.882	0.009	0.215	0.046	0.013	0.000
Program * Fault	0.000	0.041	0.057	0.002	0.000	0.023	0.021	0.001
Technique * Fault	0.001	0.269	0.032	0.472	0.028	0.037	0.255	0.264
Version * Fault	0.491	0.152	0.044	0.361	0.670	0.078	0.014	0.092
Program * Technique	0.013	0.017	0.005	0.048	0.040	0.141	0.022	0.014
Program * Version	0.295	0.590	0.811	0.156	0.061	0.760	0.054	0.710
Technique * Version	0.074	0.539	0.459	0.904	0.236	0.034	0.039	0.221
Program * Technique * Fault	0.094	0.564	0.102	0.113	0.060	0.172	0.066	0.028
Program * Version * Fault	0.125	0.708	0.242	0.075	0.245	0.212	0.412	0.019
Technique * Version * Fault	0.348	0.888	0.847	0.779	0.674	0.210	0.516	0.309
Program * Technique *Version	0.410	0.646	0.752	0.800	0.275	0.230	0.999	0.896

Finally, at a meeting with the researchers responsible for running the experiments, we examined possible deviations of real from planned experiment execution and incidents.

Our study has some **validity threats** that indicate that we have to be careful about how the results are used, as:

- Results have been obtained from junior testers. If the subjects were experienced practitioners, results could be different.
- Results have been obtained from three programs written in C. Were the programs to be larger, or written in a different programming language, results could be different.
- Results have been obtained from a certain mode of applying the techniques. For instance, no dynamic analyser is used to apply branch testing.
- The variables identified during the meeting with researchers running the experiments that could have a possible influence on results should be further explored in future experiments to check whether they do or do not have an influence.

V. EXPERIMENT RESULTS

TABLE III illustrates the significance level (p-value) of the ANOVA for each replication, as outputted by SPPSS. The first two rows in TABLE III respectively denote that the ANOVA model is valid (its value is less than 0.05 in all cases) and its statistical power is high (its value is greater than 0.8 in all cases). The shaded cells indicate the significant factors or combination of factors, that is, factors whose value is lower than 0.05.

To interpret the results of the eight replications jointly, we label each factor or combination of factors with a value that summarizes its significance trend. A factor or combination of factors has a significant trend if it is significant in 67%-100% of the replications (6-8); has an ambiguous trend when it is significant in 34%-66% of the replications (3-5); and does not have a significant trend if it is significant in 0%-33% of the replications (0-2). TABLE IV shows the significance trends. We decided to use this approach and not meta-analysis, because the raw data of the experiment was available. This approach would allow us to get more insights on the results than just examining means and standard deviations (as meta-analysis does). Let us discuss significant and ambiguous factors or combinations of factors.

TABLE IV. SIGNIFICANCE TRENDS.

Significant (67%-100%)	**Ambiguous (34%-66%)**	**Not significant (0%-33%)**
Technique Technique*Program Program*Fault	Program Fault Technique*Fault	Version Program*Version Technique*Version Fault*Version Three-way interactions

A. *Factors and Combinations with a Significant Trend*

The factors that have a significant trend are technique, and technique*program and program*fault combinations.

Figure 1 shows the mean effectiveness value for each **technique** at a 95% confidence level in each replication. The statistical analyses show that:

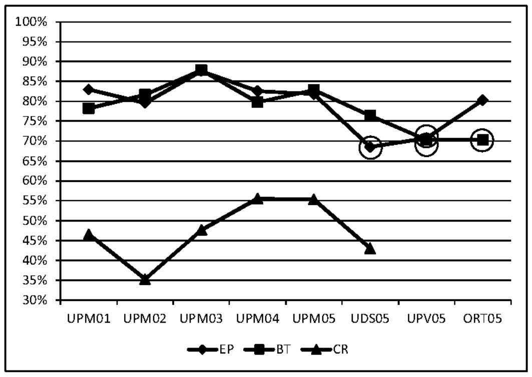

Figure 1. Technique effectiveness values.

- Even in a best-case scenario, none of the techniques comes close to 100% effectiveness, where the mean effectiveness among replications is 79.26% for equivalence partitioning, 78.43% for branch testing and 47.23% for code reading by stepwise abstraction.
- Code reading by stepwise abstraction is less effective than equivalence partitioning and branch testing in all the replications that exercise the reading technique.

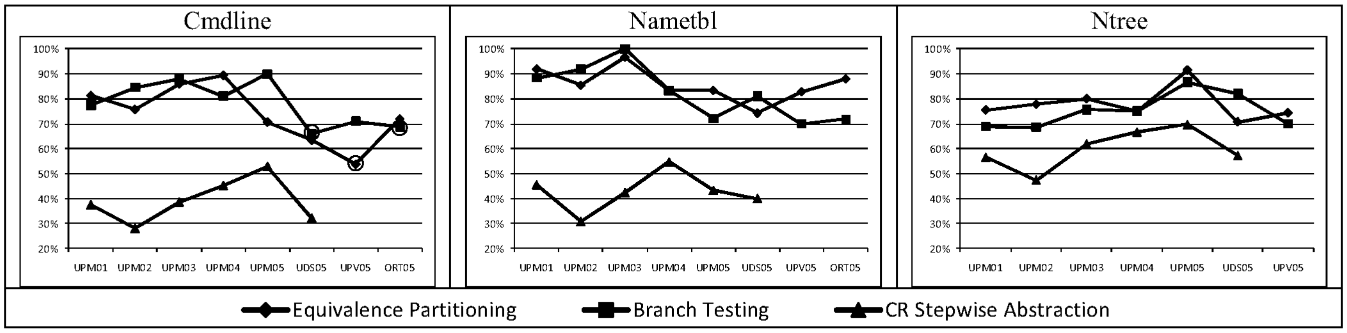

Figure 2. Technique effectiveness values by program.

- The effectiveness of equivalence partitioning and branch testing is similar, except in ORT05.

Some low effectiveness values (calculated as explained in Section 4) are circled in Figure 1: equivalence partitioning in UdS05 and UPV05, and branch testing in UPV05 and ORT05. This could be a sign that the site has some sort of influence on technique effectiveness.

Figure 2 shows the mean value of **technique effectiveness by program** in each replication. The technique*program interaction is significant in all replications except UdS05. The interaction is ordinal (treatment effects are not equal across all levels of another treatment, but they are always in the same direction [22]), meaning that the main effects can be interpreted. The statistical analyses show that:

- Code reading by stepwise abstraction behaves worse than equivalence partitioning and branch testing for all programs, where effectiveness is much worse for cmdline and nametbl and only slightly worse for ntree.
- Equivalence partitioning and branch testing tend to behave equally for all programs, except for cmdline in UPM05 and UPV05, and nametbl in ORT05.
- Some low effectiveness values for cmdline are circled in Figure 2: branch testing in UdS05 and ORT05, and equivalence partitioning in UPV05.

The program*fault interaction is disordinal (the differences between the levels of one treatment change depending on how they are combined with levels of another treatment [22]), and the main effects of the interaction cannot be interpreted separately. The disordinal interaction signifies that two identical faults do not follow the same pattern in two different programs. Nonetheless, we looked for a behavioral pattern for the program*fault interaction across the different replications.

Again we were unsuccessful. The only plausible explanation for this disordinal interaction is that the fault classification used in the experiment is not discriminative. Even though we take care to use the same type of faults (e.g., omission, control), they behave differently in combination with the techniques across programs. The techniques are able to detect the defects in some cases and not in others, for which there is no plausible fault type-related explanation.

B. *Factors and Combinations with an Ambiguous Trend*

The factors that do not have a clearly significant or not significant trend are program, fault and the technique*fault interaction.

Figure 3 shows the mean effectiveness value for each **program** for a 95% confidence interval in each of the replications. As the program*fault interaction is disordinal, the findings are limited. The program is significant in the UPM01, UPM05, UdS05, UPV05 and ORT05 replications. However, the multiple comparison tests reveal that the three programs behave equally in UPM01 and UPV05. Even though the significance level is less than 0.05, the high variability in program behavior at the above sites means that no differences are found among the programs. Consequently we find differences in program behavior in only three out of the eight replications: UPM05, UdS05 and ORT05. We find that:

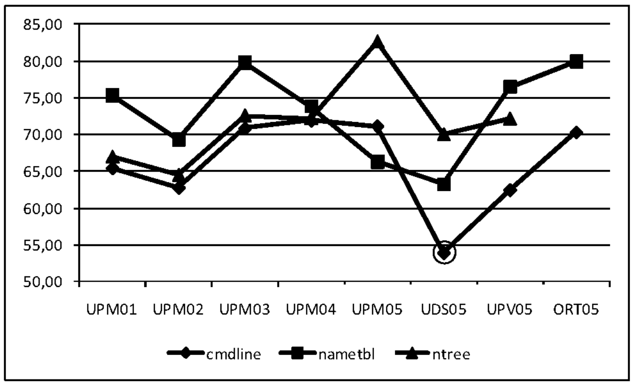

Figure 3. Program effectiveness values.

- There does not appear to be a behavioral pattern in replications in which the relative effectiveness of the programs is significant.
- There is a low effectiveness value circled in Figure 3: cmdline in UdS05.

Fault was significant in the UPM04, UdS05, UPV05 and ORT05 replications. As the program*fault interaction is disordinal, the findings are again limited. However, the multiple comparison tests reveal that faults behave equally in UPM04 and UdS05. Even though the significance level is less than 0.05, the high variability in program behavior at

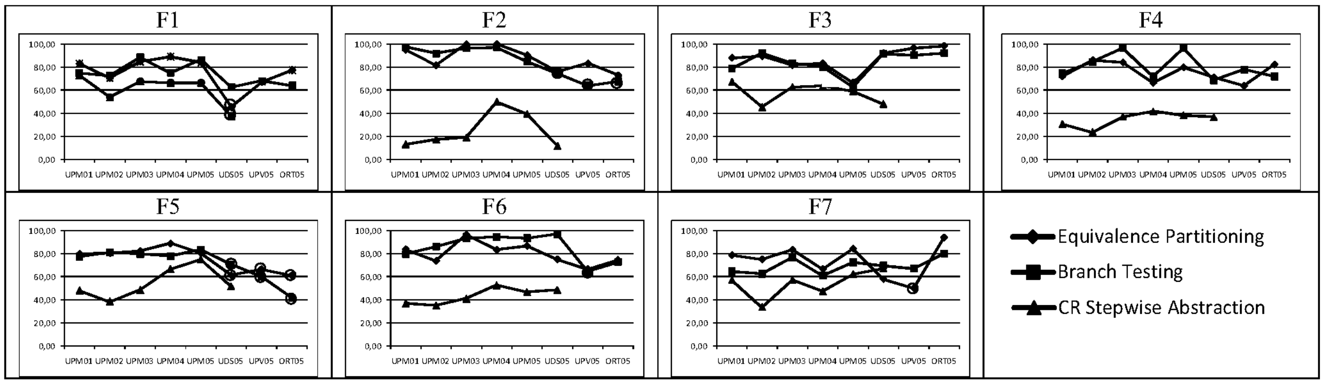

Figure 4. Technique effectiveness values by fault.

the above sites means that no differences are found among the programs. Consequently, we find differences in program behavior in only two out of the eight replications: UPV05 and ORT05. Note that the fault trend is no longer ambiguous, it is not significant. Therefore it makes no sense to analyze this trend.

Figure 4 illustrates the mean value of **technique effectiveness by fault** in each replication. The interaction is significant in the UPM01, UPM03, UPM05, and UdS05 replications. From multiple comparison tests we find that:

- Equivalence partitioning and branch testing are equally effective irrespective of the fault type.
- Code reading by stepwise abstraction is causing the interaction, as it does not behave equally for all faults in all replications. We find that there is a tendency for faults F1 (cosmetic, omission), F3 (initialization, omission), F5 (control, commission) and F7 (computation, commission) to be more effective than faults F2 (cosmetic, commission), F4 (initialization, commission) and F6 (control, omission).
- There are some extreme effectiveness values for all faults except F3 and F4. However, there is no clear behavioral pattern.

TABLE V summarizes the relevant results.

TABLE V. Summary of Relevant Results.

	UPM	UdS	UPV	ORT
Tech.	(EP=BT) >CR	(EP=BT) > CR	EP=BT	EP>BT
		EP ↓	EP ↓, BT↓	BT ↓
Tech. x Prog.	(EP=BT) > CR nt: CR ↑ UPM05: BT>EP	(EP=BT) > CR nt: CR ↑	cm: EP<BT na, nt: EP=BT	cm: EP=BT na: EP>BT
		cm: BT↓	cm: EP↓	cm: BT↓
Tech. x Fault	EP=BT irrespective of fault type CR: (F1, F3, F5, F7) > (F2, F4, F6)			

Legend: *cm* cmdline, *na* nametbl, *nt* ntree,
EP Equivalence partitioning, *BT* Branch testing, *CR: CR stepwise abs.*
↓ drops, ↑ increases, = equal, > better, < worse

VI. Differences of Context

From the analysis of the changes in the replication designs, we find that the following variables could be influencing the results.

- *Copying.* Subjects apply the same technique to different programs in each experimental session at UdS and UPV. As a result, subjects might swap information about the programs and their faults at the end of each session. As a result of copying, the techniques applied in sessions 2 and 3 could be more effective at UdS and UPV than at other sites.
- *Experience in programming issues.* At ORT, experimental subjects have hardly any programming experience. Branch testing generates test cases from source code, whereas equivalence partitioning generates them from the specification. As a result of programming experience, branch testing could be less effective at ORT than at other sites.
- *Tiredness.* At ORT, subjects complete the whole experiment in a single session, where they have to apply two techniques. Subjects could be tired by the time they come to apply the second technique. This effect is cancelled out by considering all possible application orders. As a result of tiredness, however, techniques could be less effective at ORT than at other sites.
- *Work in pairs.* At UdS, subjects apply the techniques in pairs rather than individually. As a result of pair work, techniques could be more effective at UdS than at other sites.

From the post-execution meetings held with researchers present during the experiment execution, we identified the following variables that could be influencing the observed results. For a detailed description of how all these variables have been obtained, see [49]:

- *Training.* At UdS and UPV, the subjects are familiar with the technique and receive only a refresher tutorial. At UPM and ORT, subjects receive a full course. We discovered that the tutorials did not ensure that subjects were equally knowledgeable about the techniques. As a result of training, techniques could less effective at UdS and UPV than at other sites.
- *Motivation.* Subjects at UdS and UPV are not as highly motivated because the experiment has hardly any impact on the grade for the course they are taking. At UPM and ORT subjects are graded on the experimental tasks. As a

result of motivation, techniques could be less effective at UdS and UPV than at other sites.

- *Time pressure*. UPV subjects did not have enough time to apply branch testing. It was the first technique applied, and some time was spent on explaining the experiment procedure to the subjects. Subjects were working under pressure to get the task done on time. As a result of time pressure, branch testing could be less effective at UPV than at other sites.

TABLE VI summarizes the values at each site for the possible influencing variables.

TABLE VI. POSSIBLE INFLUENTIAL VARIABLES.

	UPM	UdS	UPV	ORT
Training	Full course	Refresher tutorial	Refresher tutorial	Full course
Motivation	High	Low	Low	High
Time pressure	No	No	Yes (BT)	No
Copying	No	Yes	Yes	No
Experience in programming	Yes	Yes	Yes	No
Tiredness	No	No	No	Yes
Work in pairs	Individual	Pair	Individual	Individual

These variables can be grouped into three categories depending on the type of influence they have on the results: variables that influence all techniques; variables that influence a particular technique, and variables that have no influence. General-purpose guidelines on how to apply V&V to improve performance can be formulated from variables that influence either all or none of the techniques. Guidelines about how to select the best technique(s) for a given project context can be formulated from variables that influence a particular technique.

VII. FINDINGS

We group our findings by their source: experiment results, and combination of experiment results with contextual variables. Depending on the reliability of the results, they are also classed as evidence or signs.

A. *Experimental Results*

We have found statistical evidence that:

- **Equivalence partitioning and branch testing are equally effective.** Effectiveness is similar in seven out of the eight replications (the exception being 0RT05).
- Equivalence partitioning and branch testing **are more effective when different subjects apply the same technique** than when **the same subject applies different techniques**. Techniques are not 100% effective (theoretical likelihood) in any of the replications.
- **Equivalence partitioning and branch testing effectiveness is not dependent on the program**. This is true in five out of the eight replications (the exceptions being UPV05, UPV05 and ORT05).
- **Equivalence partitioning and branch testing are more effective than code reading by stepwise abstraction.** Code reading by stepwise abstraction behaves worse than equivalence partitioning and branch testing in all six replications evaluating this technique.
- **Code reading by stepwise abstraction should be applied in conjunction with equivalence partitioning or branch testing,** as the effectiveness of code reading by stepwise abstraction is fault dependent. F1 (cosmetic, omission), F3 (initialization, omission), F5 (control, commission) and F7 (computation, commission) generally tend to behave better than F2 (cosmetic, commission), F4 (initialization, commission) and F6 (control, omission) in all replications, although the difference varies from one replication to another.
- **Fault detection effectiveness appears to be program dependent**. The program was influential in seven out of the eight replications, although we were unable to find a behavioral pattern. This suggests that the fault classification that we used was not discriminative, as supposedly identical faults (all F1s, all F2s, etc.) did not behave equally in all the programs. An alternative fault classification scheme is required to study effectiveness. Neither is fault behavior consistent across replications. Faults that subjects found very easy (very hard) to detect in one replication were not in others. This suggests that fault behaviors are unrepeated and erratic, and there is no pattern.
- **Program appears to influence the behavior of code reading by stepwise abstraction**. In all six replications examining this technique, it is more effective with ntree than with the other two programs. A detailed analysis of ntree does not reveal any special feature that might single it out. Whereas cmdline might be expected to behave differently, nametbl and ntree are in many respects alike: both are programs that implement a data structure, etc. The difference is more likely to be due to the faults seeded in ntree code, which, for some, as yet unknown, reason, are easier for subjects to find than others. It is surprising that the effectiveness results for code reading by stepwise abstraction were different even with extremely similar programs. This would suggest that the technique is highly sensitive to program or fault characteristics.
- **Technique effectiveness is independent of the fault type** in all replications. However, subjects do not detect the same faults, since all faults are being detected by some subjects.

B. *Experimental Results and Context*

There are signs that:

- **Programming inexperience decreases the effectiveness of branch testing**. In ORT05, branch testing was less effective than equivalence partitioning. ORT05 subjects were not experienced with programming issues. This difference is greater for the cmdline program, which is the program with the greatest cyclomatic complexity.
- **Technique knowledge could affect technique effectiveness**. Subject training in UPV05 and UdS05 is not as thorough as at UPM and in ORT05. This could explain why both techniques are less effective in UPV05, and equivalence partitioning underperforms in UdS05.

- **Techniques might be less effective if subjects are not motivated**. The subjects in UPV05 and UdS05 are less motivated than UPM and ORT05 subjects. This could explain why equivalence partitioning and branch testing are less effective in UPV05, and equivalence partitioning underperforms in UdS05.
- **Techniques are more effective with pair work**. Although motivation and training could be having a negative influence, only equivalence partitioning behaves worse in UdS05 than in the other replications. This suggests that pair work might be offsetting any effects that motivation and training have on branch testing and code reading by stepwise abstraction that were applied in pairs (not so in the case of equivalence partitioning).
- **Time pressure does not appear to influence branch testing effectiveness**. The effectiveness of equivalence partitioning and branch testing is similar in UPV05. If work under pressure had been influential, branch testing should have been less effective since subjects had less time to apply this technique than equivalence testing.
- **Tiredness does not appear to influence technique effectiveness**. If tiredness were to affect effectiveness, the techniques should have been less effective in ORT05, but they were not.
- **Privileged information about the program does not appear to influence technique effectiveness**. In UPV05 and UdS05 subjects had the chance to swap information on the programs at the end of each session. If privileged information were to be influential, equivalence partitioning should have behaved better than branch testing in UPV05 and UdS05, and code reading by stepwise abstraction better than the dynamic techniques in UdS05.

VIII. Conclusions

We have run eight replications of the same experiment (some at other sites) to try to understand the effectiveness of three V&V techniques: two dynamic techniques (equivalence partitioning and branch coverage) and one static technique (code reading by stepwise abstraction). We analyzed the results by jointly interpreting results. We also conducted a study of contextual differences between the different sites at which the replications were run.

The findings reveal that equivalence partitioning and branch testing are similarly effective and more effective than code reading by stepwise abstraction. Likewise, we discovered that equivalence partitioning and branch testing are independent of the fault type. This suggests that it might be worthwhile combining subjects using the techniques to increase technique effectiveness.

On the other hand, we identified contextual variables that could be influencing the results. Training, motivation, and pair work could have an effect on technique effectiveness, and programming language experience could have an effect on branch testing's effectiveness. Translating these results to industry, training, motivation and pair work could improve testing effectiveness, whereas language experience could influence the decision on whether to use equivalence partitioning or branch testing.

More experiments with other techniques would show whether the behavior of equivalence partitioning and branch testing can be generalized to black-box and white-box testing techniques.

References

[1] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sorumgard, M. V. Zelkowitz, The Empirical Investigation of Perspective Based Reading. *Journal of Empirical Software Engineering*. 1(2):133–164, 1996.

[2] V.R. Basili, R.W. Selby. Comparing the Effectiveness of Software Testing Strategies. *IEEE Transactions on Software Engineering,* 13(2):1278-1296. 1987.

[3] B. Beizer. *Software Testing Techniques*. International Thomson Computer Press, Second Edition. 1990.

[4] A. Bertolino. Guide to the Knowledge area of Software Testing. *Software Engineering Body of Knowledge*. IEEE Computer Society. 2004.

[5] J. Bible, G. Rothermel, D. Rosenblum. A Comparative Study of Coarse- and Fine-Grained Safe Regression Test Selection. *ACM Transactions on Software Engineering and Methodology* 10(2):149-183. 2001.

[6] J.M. Bieman, J.L. Schultz. An Empirical Evaluation (and Specification) of the All-du-paths Testing Criterion. *Software Engineering Journal*, January 1992, pp. 43–51.

[7] S. Biffl, Analysis of the Impact of Reading Technique and Inspector Capability on Individual Inspection Performance. 7th *Asia-Pacific Software Engineering Conference*, pp. 136–145. 2000.

[8] L.C. Briand, M. Penta, Y. Labiche. Assessing and Improving State-Based Class Testing: A Series of Experiments. *IEEE Transactions on Software Engineering*. 30(11):770-793. 2004.

[9] T.Y. Chen, Y.T. Yu. On the Relationships between Partition and Random Testing. *IEEE Transactions on Software Engineering*. 20(12): 977-980, 1994.

[10] L.A. Clarke, A. Podgurski, D.J. Richardson, S.J. Zeil. A Formal Evaluation of Data Flow Path Selection Criteria. *IEEE Transactions on Software Engineering*. 15(11):1318-1332, 1989.

[11] H. Do, S. Elbaum, G. Rothermel. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact. *Empirical Software Engineering*. 10:405–435, 2005.

[12] A. Dunsmore, M. Roper, M. Wood. Further Investigations into the Development and Evaluation of Reading Techniques for Object-oriented Code Inspection. *24th International Conference on Software Engineering*, pp. 47–57.2002.

[13] J.W. Duran, S.C. Ntafos. An Evaluation of Random Testing. *IEEE Transactions on Software Engineering*. SE-10(4): 438—444, 1984.

[14] S. Elbaum, A.G. Mailshevsky, G. Rothermel. Prioritizing Test Cases for Regression Testing. *International Symposium on Software Testing and Analysis*. 2000, pp. 102–112.

[15] P.G. Frankl, Y. Deng. Comparison of Delivered Reliability of Branch, Data Flow and Operational Testing, a case study. *International Symposium on Software Testing and Analysis*. 2000.

[16] P. Frankl, O. Iakounenko. Further Empirical Studies of Test Effectiveness. *International Symposium on Foundations on Software Engineering*. Pp 153-162. 1998

[17] P.G.Frankl, S.N. Weiss. An Experimental Comparison of the Effectiveness of Branch Testing and Data Flow Testing. *IEEE Transactions on Software Engineering*, 19(8):774-787. 1993.

[18] P.G. Frankl, S.N. Weiss, C. Hu. All-Uses vs Mutation Testing: An

Experimental Comparison of Effectiveness. *Journal of Systems and Software*, 38:235-253. 1997.

[19] P.G. Frankl, E.J. Weyuker. Assessing the Fault--Detecting Ability of Testing Methods. *International Conference on Software for Critical Systems*, pp. 77—91. 1991.

[20] O.S. Gómez, N. Juristo, S. Vegas. Replication Types in Experimental Disciplines. *International Symposium on Empirical Software Engineering and Measurement (ESEM'10)*. Pages 19-28. September 16-17, 2010. Bolzano, Italy.

[21] T.L. Graves, M.J. Harrold, J. Kim, A. Porter, G. Rothermel. An Empirical Study of Regression Test selection Techniques. *ACM Transactions on Software Engineering and Methodology*, 10(2):184-208. 2001.

[22] J.F. Hair, W.C Black, B.J. Babin, R.E. Anderson, R.L. Tatham. *Multivariate Data Analysis 6th edition*. Prentice Hall. 2006.

[23] R. Hamlet. Probable correctness theory. *Information Processing Letters*. 25:17-25, 1987.

[24] D. Hamlet, R. Taylor. Partition Testing does not Inspire Confidence. *IEEE Transactions on Software Engineering*. 16(12): 1402—1411, 1990.

[25] M. Hutchins, H. Foster, T. Goradia, T. Ostrand. Experiments on the Effectiveness of Dataflow- and Controlflow-Based Test Adequacy Criteria. *16th International Conference on Software Engineering*. Pp. 191-200. 1994.

[26] Juristo N, Moreno A.M. Basics of Software Engineering Experimentation. Kluwer. 2001.

[27] N. Juristo, A.M. Moreno, S. Vegas, M. Solari. In Search of What We Experimentally Know about Unit Testing. *IEEE Software*. 23(6):72-80. 2006.

[28] E. Kamsties, C.M. Lott. An Empirical Evaluation of Three Defect-Detection Techniques. *Fifth European Software Engineering Conference*. 1995.

[29] J.M. Kim, A. Porter, G. Rothermel. An Empirical Study of Regression Test Application Frequency. *22nd International Conference* on *Software Engineering*. Pp.126-135. 2000.

[30] H. Lee, Y. Ma, Y. Kwon. Empirical Evaluation of Orthogonality of Class Mutation Operators. *11th Asia-Pacific Software Engineering Conference*. 2004, pp. 512–518.

[31] D. Leon, W. Masri, A. Podgurski. An Empirical Evaluation of Test Case Filtering Techniques Based on Exercising Complex Information Flows. *27th International Conference on Software Engineering*. 2005.

[32] R.C. Linger, H.D. Mills, B.I. Witt. *Structured Programming: Theory and Practice*. Addison-Wesley, 1979.

[33] J. Maldonado, J. Carver, F. Shull, S. Fabbri, E. Dória, L. Martimiano, M. Mendonça, V. Basili. Perspective-based Reading: A Replicated Experiment Focused on Individual Reviewer Effectiveness. *Empirical Software Engineering*. 11(1):119–142, 2006.

[34] G.J. Myers. A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections. *Communications of the ACM*. 21(9): 760-768, 1978.

[35] Myers G.J. T. Badgett, C. Sandler. *The Art of Software Testing*. Wiley-Interscience. 2nd edition. 2004.

[36] S. Ntafos. A comparison of some structural testing strategies. *IEEE Transactions on Software Engineering*. 14(6):368-874, 1988.

[37] S.C. Ntafos. On Random and Partition Testing. *International Symposium on Software Testing and Analysis*, pp. 42-48. 1998.

[38] A.J. Offutt, S.D. Lee. An Empirical Evaluation of Weak Mutation. *IEEE Transactions on Software Engineering,* 20(5):337-344. 1994.

[39] A.J. Offutt, A. Lee, G. Rothermel, RH. Untch, C. Zapf. An Experimental Determination of Sufficient Mutant Operators. *ACM Transactions on Software Engineering and Methodology,* 5(2):99-118. 1996.

[40] A. Porter, L. Votta, V.R. Basili. Comparing Detection Methods for Software Requirements Inspection: A Replicated Experiment. *IEEE Transactions on Software Engineering*, 21(6): 563–575, 1995.

[41] S. Rapps, E. Weyuker. Selecting Software Test Data Using Data Flow Information. *IEEE Transactions on Software Engineering*. SE-11(4):367-375, 1985.

[42] M. Roper, M. Wood, J. Miller. An empirical evaluation of defect detection techniques. *Information and Software Technology*. 39:763-775. 1997.

[43] D.S. Rosenblum, E.J. Weyuker. Using Coverage Information to Predict the Cost-Effectiveness of Regression Testing Strategies. *IEEE Transactions on Software Engineering*. 23(3):146—156, 1997.

[44] G. Rothermel, S. Elbaum, A.B. Malishevsky, P. Kallakuri, X. Qiu. On Test Suite Composition and Cost-Effective Regression Testing. *ACM Transactions on Software Engineering and Methodology*. 13(3):277-331. 2004.

[45] G. Rothermel, M.J. Harrold. Analyzing Regression Test Selection Techniques. *IEEE Transactions on Software Engineering*. 22(8):529-551, 1996.

[46] G. Rothermel, M.J. Harrold. Empirical Studies of a Safe Regression Test Selection Technique. *IEEE Transactions on Software Engineering*. 24(6):401-419. 1998.

[47] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold. Test Case Prioritization: An Empirical Study. *International Conference on Software Maintenance*. Pp. 179-188. 1999.

[48] T. Thelin, P. Runeson, C. Wohlin, T. Olsson, C. Andersson. Evaluation of Usage-Based Reading—Conclusions after Three Experiments. *Empirical Software Engineering*. 9:77–110, 2004.

[49] S. Vegas, N. Juristo, A.M. Moreno, M. Solari, P. Letelier. Analysis of the Influence of Communication between Researchers on Experiment Replication. *International Symposium on Empirical Software Engineering*. Pp. 28-37. 2006.

[50] F. Vokolos, P.G. Frankl. Empirical Evaluation of the Textual differencing Regression Testing Technique. *International Conference on Software Maintenance*. 1998.

[51] M.D. Weiser, J.D. Gannon, P.R. McMullin. Comparison of Structural Test Coverage Metrics. *IEEE Software*. 2(3):80-85, 1985.

[52] E.J. Weyuker. The Complexity of Data Flow Criteria for Test Data Selection. *Information Processing Letters*. 19(2):103-109. 1984.

[53] E.J. Weyuker. The Cost of Data Flow Testing: An Empirical Study. *IEEE Transactions on Software Engineering*. 16(2): 121–128, 1990.

[54] E.J. Weyuker, B. Jeng. Analyzing Partition Testing Strategies. *IEEE Transactions on Software Engineering*. 17(7): 703—711, 1991.

[55] W.E. Wong, J.R. Horgan, S. London, H. Agrawal. A Study of Effective Regression Testing in Practice. *8th International Symposium on Software Reliability Engineering*. Pp.264-274. 1998

[56] E. Wong, A.P. Mathur. Fault Detection Effectiveness of Mutation and Data-flow Testing. *Software Quality Journal*, 4:69-83. 1995.

[57] T.W Wright. A Treatise on the Adjustment of Observations by the method of Least Squares. Van Nostrand. 1884.

[58] S.J. Zeil. Selectivity of Data--Flow and Control--Flow Path Criteria. *2nd Workshop on Software Testing, Verification and Analysis*, pp. 215—222. 1988.