Summarizing the Results of a Series of Experiments:
Application to the Effectiveness of Three Software Evaluation Techniques

AUTHOR: BABATUNDE KAZEEM OLORISADE
ACADEMIC ADVISOR: SIRA VEGAS
(Universidad Politécnica de Madrid)
FACULTY REVIEWER: RICHARD TORKAR
(Blekinge Tekniska Högskola)

Madrid, December 2009
Summarizing the Results of a Series of Experiments: Application to Effectiveness of Three Software Evaluation Techniques

Author: Babatunde Kazeem Olorisade
This Thesis work is submitted to the Departamento de Lenguajes y Sistemas Informáticos e Ingeniería del Software (Software Engineering Department) Facultad de Informática, Universidad Politécnica de Madrid, Madrid, Spain and School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Mater of Science in Software Engineering of the European Master on Software Engineering (EMSE), an Erasmus Mundus Program.

Contact Information:
Author(s): Babatunde Kazeem Olorisade
E-mail: qasimbabatunde@yahoo.co.uk

Academic Advisor: Dr. Sira Vegas
Universidad Politécnica de Madrid
Address: Facultad de Informática, Campus de Montegacendo

Faculty Reviewer: Richard Torkar
Blekinge Institute of Technology
Department name: Software Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25
To the memory of my beloved Mum.
Acknowledgement

First and foremost, I am grateful to God almighty that made this endeavor achievable.

To begin with, I would like to express my appreciation to my supervisors, Dr. Natalia Juristo, Dr. Sira Vegas. I have to thank Dr. Sira Vegas for her devotion, patience and guidance in the course of this work. Worth mentioning is how she devoted her time to attend to my concerns, the calmness and warmth she always use to handle all issues related to the work and her approach to problem solving. You made my initiation into the empirical world of software engineering remarkable and interesting. I learnt a lot from you, knowledge and attitude wise. I am not sure how to thank Dr. Natalia Juristo enough, for the series of professional advice, useful personal observations and comments, the teaching, encouragements and support. I also thank Dr. Richard Torkar for accepting to co-supervise this work and believing in us, even when he is not there.

Moreover, I will extend my gratitude to Dr. Oscar Díeste, with whom almost any problem is solvable in the EMSE. I cannot complete the list of how kind, useful and supportive he is, throughout the tenure of the program. I would love you permit me to call you “Dr. Fix it” when it comes to EMSE issues. The appreciation is not complete without mentioning Sr. Nella Alvedo for her support, kindness and assistance over the period of my study at UPM.

I am also grateful to Martin Solari for his assistance on the experiment files used for this work and all the members of the Empirical Software Engineering Group (GriSE) at the Universidad Politécnica de Madrid; I cannot continue to list all of you.

Finally, I am grateful to my mother for giving everything she could when she could and believing in me even when the world seem not to, may your lovely soul rest in peace. I thank my sweetheart for standing by me throughout the period of this work, thank you for your understanding, support and motivation. I thank my brother and friend – Musiliu Akinwande and his wife - Shakirat for their support and motivation. I will not forget my brothers and sisters - Kafayat, Saeed, Ruqayat, Lekan and Igbayilola. Thank you all for being there whenever I need you.
Abstract

Software quality has become and persistently remains a big issue among software users and developers. So, the importance of software evaluation cannot be overemphasized. An accepted fact in software engineering is that software must undergo evaluation process during development to ascertain and improve its quality level.

In fact, there are too many techniques than a single developer could master, yet, it is impossible to be certain that software is free of defects. Therefore, it may not be realistic or cost effective to remove all software defects prior to product release. So, it is crucial for developers to be able to choose from available evaluation techniques, the one most suitable and likely to yield optimum quality results for different products - it bogs down to choosing the most appropriate for different situations.

However, not much knowledge is available on the strengths and weaknesses of the available evaluation techniques. Most of the information related to the techniques available is focused on how to apply the techniques but not on the applicability conditions of the techniques – practical information, suitability, strengths, weaknesses etc. This research focuses on contributing to the available applicability knowledge of software evaluation techniques. More precisely, it focuses on code reading by stepwise abstraction as representative of the static technique, as well as equivalence partitioning (functional technique) and decision coverage (structural technique) as representatives of the dynamic technique.

The specific focus of the research is to summarize the results of a series of experiments conducted to investigate the effectiveness of these techniques among other factors. By effectiveness in this research, we mean the potential of each of the techniques to generate test cases capable of revealing software faults in the case of the dynamic techniques or the ability of the static technique to generate abstractions that will aid the detection of faults. The experiments used two versions of three different programs with seven different faults seeded into each of the programs. This work uses the results of the eight different experiments performed and analyzed separately, to explore this fact. The analysis results were pooled together and jointly summarized in this research to extract a common knowledge from the experiments using a qualitative deduction approach created in this work as it was decided not to use formal aggregation at this stage.
Since the experiments were performed by different researchers, in different years and in some cases at different site, there were several problems that have to be tackled in order to be able to summarize the results. Part of the problems is the fact that the data files exist in different languages, the structure of the files are different, different names is used for data fields, the analysis were done using different confidence level etc.

The first step, taken at the inception of this research was to apply all the techniques to the programs used during the experiments in order to detect the faults. This purpose of this personal experience with the experiment is to be familiarized and get acquainted to the faults, failures, the programs and the experiment situations in general and also, to better understand the data as recorded from the experiments. Afterwards, the data files were recreated to conform to a uniform language, data meaning, file style and structure. A well structured directory was created to keep all the data, analysis and experiment files for all the experiments in the series. These steps paved the way for a feasible results synthesis.

Using our method, the technique, program, fault, program – technique, program – fault and technique – fault were selected as main and interaction effects having significant knowledge relevant to the analysis summary result. The result, as reported in this thesis, indicated that the functional technique and the structural technique are equally effective as far as the programs and faults in these experiments are concerned. Both perform better than the code review. Also, the analysis revealed that the effectiveness of the techniques is influenced by the fault type and the program type. Some faults were found to exhibit better behavior with certain programs, some were better detected with certain techniques and even the techniques yield different result in different programs.
## Contents

1. **Introduction** ........................................................................................................ 1  
   1.1 Study Background ................................................................................................. 2  
   1.2 Motivation .............................................................................................................. 4  
   1.3 Objectives and Research Questions ....................................................................... 4  
   1.4 Approach to the Solution .................................................................................... 6  
   1.5 Thesis Structure ................................................................................................. 7  

2. **Description of the Experiments** ..................................................................... 9  
   2.1 UPM Experiments ............................................................................................... 9  
   2.2 UPV Experiment ................................................................................................. 13  
   2.3 UdS Experiment ................................................................................................. 14  
   2.4 ORT Experiment ............................................................................................... 15  

3. **Description of Programs, Techniques and Faults** ..................................... 17  
   3.1 Description of the Programs ............................................................................... 17  
   3.2 Description of the Evaluation Techniques .......................................................... 20  
   3.3 Description of the Fault Types ........................................................................... 24  
   3.4 Specific Fault Description and Failure Characterization ................................... 25  

4. **Exercising with the Techniques** ................................................................. 47  
   4.1 Test Case Generation ......................................................................................... 47  
   4.2 Failure/Fault Observation .................................................................................. 49  
   4.3 Influence of Learning Effect ............................................................................... 51  
   4.4 Reasons for Missing a Fault or notGenerating a Test Case ............................. 52  
   4.5 General Comments on the Exercise ................................................................... 56  

5. **Common Structure** ...................................................................................... 63  
   5.1 Directory Structure ............................................................................................ 64  

---

Babatunde Kazeem
5.2 The Original Excel Files ........................................................................................................ 65
5.3 Procedure used to Create the Structure of the New Excel Files ........................................ 66
5.4 The Master Excel File ............................................................................................................. 80
5.5 Creation of the Specific Sheets .............................................................................................. 81

6 Global View of the Experiments ......................................................................................... 83
   6.1 A Process to Summarize the Results of Several Experiments ........................................ 83
   6.2 Summary of the Factors’ Significance .................................................................................. 86
   6.3 Classification ....................................................................................................................... 86
   6.4 Ranking of the Classification Table ..................................................................................... 89
   6.5 Deduction ............................................................................................................................ 93

7 Conclusion and Recommendation ......................................................................................... 127
   7.1 Conclusions ......................................................................................................................... 127
   7.2 Further research .................................................................................................................. 128

References ......................................................................................................................................... 130
Appendix A Structure of the Original Excel Files ...................................................................... 132
Appendix B Mapping between New and Original Excel Files .................................................. 135
   B.1 Subject data sheet .............................................................................................................. 135
   B.2 Observable faults sheet .................................................................................................... 139
   B.3 Failure Visibility Sheet .................................................................................................... 142
   B.4 Observed Failure Sheet .................................................................................................... 144
# Table of Contents

1  **Introduction** ................................................................. 1  
   1.1  Study Background ............................................................ 2  
   1.2  Motivation ....................................................................... 4  
   1.3  Objectives and Research Questions .................................. 4  
   1.4  Approach to the Solution ............................................... 6  
   1.5  Thesis Structure ............................................................. 7  

2  **Description of the Experiments** ........................................ 9  
   2.1  UPM Experiments ......................................................... 9  
      2.1.1  Study Objectives ....................................................... 9  
      2.1.2  Hypothesis and Response Variable .......................... 10  
      2.1.3  Factors and Alternatives ....................................... 11  
      2.1.4  Experimental Design ............................................. 12  
      2.1.5  Running the Experiment ....................................... 12  
   2.2  UPV Experiment .......................................................... 13  
   2.3  UdS Experiment .......................................................... 14  
   2.4  ORT Experiment ......................................................... 15  

3  **Description of Programs, Techniques and Faults** .................. 17  
   3.1  Description of the Programs .......................................... 17  
      3.1.1  Cmdline ............................................................... 17  
      3.1.2  Nametbl .............................................................. 18  
      3.1.3  Ntree ................................................................. 19  
   3.2  Description of the Evaluation Techniques .......................... 20  
      3.2.1  Review Technique .................................................. 22  
      3.2.2  Dynamic Techniques ............................................. 23
3.2.2.1 Equivalence Partitioning ................................................................. 23
3.2.2.2 Decision Coverage ........................................................................ 24

3.3 Description of the Fault Types .............................................................. 24

3.4 Specific Fault Description and Failure Characterization ..................... 25

3.4.1 Faults in Cmdline .............................................................................. 25
3.4.1.1 Version 1 ....................................................................................... 25
3.4.1.2 Version 2 ....................................................................................... 28

3.4.2 Faults in Nametbl .............................................................................. 31
3.4.2.1 Version 1 ....................................................................................... 31
3.4.2.2 Version 2 ....................................................................................... 35

3.4.3 Faults in the Ntree ............................................................................ 38
3.4.3.1 Version 1 ....................................................................................... 38
3.4.3.2 Version 2 ....................................................................................... 41

4 Exercising with the Techniques ............................................................ 47

4.1 Test Case Generation ........................................................................... 47
4.1.1 Test Cases using Equivalence Partitioning ....................................... 48
4.1.2 Test Cases using Decision Coverage .............................................. 49

4.2 Failure/Fault Observation .................................................................... 49
4.2.1 Using Equivalence Partitioning ...................................................... 49
4.2.2 Using Decision Coverage ............................................................... 50
4.2.3 Using Code Abstraction ................................................................. 51

4.3 Influence of Learning Effect ............................................................... 51

4.4 Reasons for Missing a Fault or not Generating a Test Case ............... 52
4.4.1 Reasons for not Generating a Test Case .......................................... 52
4.4.1.1 Improper Technique Application ............................................... 52
4.4.1.2 Unclear Reason ......................................................................... 53
4.4.2 Reasons for Missed Faults ...................................................................................................................... 53
4.4.2.1 Using Code Abstraction .................................................................................................................. 54
4.4.2.2 Using Equivalence Partitioning .................................................................................................... 54
4.4.2.3 Using Decision Coverage ............................................................................................................ 56
4.5 General Comments on the Exercise ............................................................................................................ 56
4.5.1 Comment on Code Reading .................................................................................................................. 57
4.5.1.1 Technical Knowledge or Experience ........................................................................................ 57
4.5.1.2 Program Structure and Behavior ................................................................................................ 57
4.5.1.3 Personal Motivation ....................................................................................................................... 58
4.5.2 Comments on Dynamic Techniques .................................................................................................... 58
4.5.2.1 Comments on Equivalence Partitioning ....................................................................................... 59
4.5.2.2 Comments on Decision Coverage ................................................................................................ 60
5 Common Structure ........................................................................................................................................... 63
5.1 Directory Structure .................................................................................................................................. 64
5.2 The Original Excel Files .......................................................................................................................... 65
5.3 Procedure used to Create the Structure of the New Excel Files ................................................................ 66
5.3.1 Glossary of Terms .................................................................................................................................. 66
5.3.1.1 Terms used in the UPM 01 – UPM 04 Files .................................................................................. 67
5.3.1.2 Terms used in UPM 05 File ........................................................................................................... 69
5.3.1.3 Terms used in UPV 05 File .............................................................................................................. 71
5.3.1.4 Terms used in ORT 05 File ............................................................................................................ 72
5.3.1.5 Terms used in the UdS 05 File ........................................................................................................ 73
5.3.2 Abstraction of Relevant Data ............................................................................................................... 74
5.3.2.1 Basic Data Fields .............................................................................................................................. 75
5.3.2.2 Other Accommodated Data Fields ............................................................................................... 76
5.3.2.3 Worksheets Definition ................................................................................................................ 77
5.3.3 Mapping of Data........................................................................................................................... 77
  5.3.3.1 Mapping the New Data Definition with the UPM 01 – UPM 04 Files ....... 78
  5.3.3.2 Mapping the New Definition with the UPM 05 File.............................................. 78
  5.3.3.3 Mapping the New Definition with the ORT 05 File ........................................... 78
  5.3.3.4 Mapping the New Definition with the UPV 05 File........................................ 79
  5.3.3.5 Mapping the New Definition with the US 05 File ........................................... 79

5.4 The Master Excel File................................................................................................................... 80
  5.4.1 Subject Data Sheet........................................................................................................ 80
  5.4.2 Observable Fault Sheet.................................................................................................. 81
  5.4.3 Failure Visibility Sheet .............................................................................................. 81
  5.4.4 Observed Faults Sheet............................................................................................... 81

5.5 Creation of the Specific Sheets............................................................................................... 81

6 Global View of the Experiments ......................................................................................... 83

6.1 A Process to Summarize the Results of Several Experiments ........................................... 83
  6.1.1 The Extraction Step ..................................................................................................... 84
  6.1.2 The Classification Step ............................................................................................. 84
  6.1.3 The Classification Ranking ....................................................................................... 85
  6.1.4 The Deduction Step ..................................................................................................... 85

6.2 Summary of the Factors’ Significance................................................................................. 86

6.3 Classification ............................................................................................................................. 86

6.4 Ranking of the Classification Table...................................................................................... 89
  6.4.1 Explanation of the Ranks............................................................................................ 89
  6.4.2 Interpolation of the Ranking into the ANOVA Table............................................ 90
  6.4.3 ANOVA Analysis Summary....................................................................................... 91

6.5 Deduction.................................................................................................................................. 93
  6.5.1 Status: Significant.......................................................................................................... 93
6.5.2 Status: Significant Tendency ................................................................. 95
  6.5.2.1 Program ................................................................................................. 95
  6.5.2.2 Program – fault interaction ................................................................. 97
  6.5.2.3 Program – technique interaction ....................................................... 107

6.5.3 Status: Not clear ...................................................................................... 107
  6.5.3.1 Fault ....................................................................................................... 115
  6.5.3.2 Fault – technique interaction .............................................................. 118

6.5.4 Overview of the joint analysis ................................................................. 123

7 Conclusion and Recommendation ............................................................... 127
  7.1 Conclusions ............................................................................................... 127

  7.2 Further research ......................................................................................... 128

References ........................................................................................................ 130
Appendix A Structure of the Original Excel Files ........................................... 132
Appendix B Mapping between New and Original Excel Files ......................... 135

  B.1 Subject data sheet ....................................................................................... 135
  B.2 Observable faults sheet ............................................................................. 139
  B.3 Failure Visibility Sheet ............................................................................. 142
  B.4 Observed Failure Sheet ............................................................................. 144
List of Figures

Figure 3.1: Transparent view in white box testing reveals internal parts .............................................. 21
Figure 3.2: Black box testing uses external product view ....................................................................... 21
Figure 3.3 Summary of steps in reading by abstraction............................................................................ 22
Figure 3.4: Steps in software testing............................................................................................................. 23
Figure 4.1: Situations encountered with application of dynamic techniques ....................................... 48
Figure 5.1: Structure of the directory for the experiments ........................................................................... 65
List of Tables

Table 2.1: Experiment design for experiment II in UPM ................................................................. 12
Table 2.2: Experiment execution for experiment II in UPM .......................................................... 12
Table 2.3: Environmental differences and adjustment in the UPV experiment .............................. 13
Table 2.4: Environmental differences and adjustment in the UdS experiment ............................. 15
Table 2.5: Environmental differences and adjustment in the ORT experiment ............................ 15
Table 3.1: Fault description for cmdline version 1 ........................................................................ 28
Table 3.2: Fault description for cmdline version 2 ........................................................................ 31
Table 3.3: Details of fault in nametbl version 1 .............................................................................. 34
Table 3.4: Description of faults in nametbl version 2 ..................................................................... 37
Table 3.5: Description of faults in ntree version 2 .......................................................................... 41
Table 3.6: Description of faults in ntree version 2 .......................................................................... 45
Table 4.1: Test case generated using equivalence class partition .............................................. 48
Table 4.2: Test case generated using decision coverage ............................................................. 49
Table 4.3: Failure observation using equivalence class partition ............................................... 50
Table 4.4: Failure observation using decision coverage ............................................................. 50
Table 4.5: Fault observation using code abstraction .................................................................... 51
Table 4.6: Faults found influenced by learning effect ................................................................. 51
Table 4.7: Reasons for not generating a test case ........................................................................ 53
Table 4.8: Faults missed during abstraction ................................................................................. 54
Table 4.9: Faults missed due to lack of program understanding using abstraction .................... 55
Table 4.10: Undetected faults due to inattention using equivalent partition .............................. 55
Table 4.11: Undetected faults due to possible masking using equivalent partition ..................... 55
Table 4.12: Undetected faults due to inattention using decision coverage .................................. 56
Table 4.13: Undetected faults due to possible masking using equivalent partition ..................... 56
Table 5.1: File structure for each experiment .............................................................................. 65
Table 5.2: Old and new field names correspondence in the UPM 01 – UPM 04 files ............... 78
Table 5.3: Old and new field names correspondence in the UPM 05 file ..................................... 78
Table 5.4: Old and new field names correspondence in the ORT 05 file ..................................... 79
Table 5.5: Old and new field names correspondence in the UPV 05 file ..................................... 79
Table 5.6: Old and new field names correspondence in the UPV 05 file ..................................... 80
Table 6.1: Overview of the significant main and interaction effects .......................................................... 87
Table 6.2: Coding of the possible combination of treatment based on level significance ..................... 87
Table 6.3: Significance value from the ANOVA table for all experiments ........................................... 88
Table 6.4: Ranking and classification of effect behavior ....................................................................... 89
Table 6.6: Interpretation of ranks in the ANOVA table ......................................................................... 91
Table 6.5: ANOVA summary with the codes and ranks ....................................................................... 92
Table 6.7: Summary of factors’ status ................................................................................................. 93
Table 6.8: SPSS profile plot for technique ............................................................................................ 94
Table 6.9: SPSS profile plots for program ............................................................................................. 96
Table 6.10: Program-fault interaction for UPM 01 .............................................................................. 99
Table 6.11: Program-fault interaction for UPM 02 ........................................................................... 100
Table 6.12: Program-fault interaction for UPM 04 ........................................................................... 101
Table 6.13: Program-fault interaction for UPM 05 ........................................................................... 102
Table 6.14: Program-fault interaction for UPV 05 ........................................................................... 103
Table 6.15: Program-fault interaction for ORT 05 ........................................................................... 104
Table 6.16: Program-fault interaction for UdS 05 ............................................................................. 105
Table 6.17: Output of cumulated cluster analysis for program-fault interaction .................................. 106
Table 6.18: Program-technique interaction for UPM 01 .................................................................... 108
Table 6.19: Program-technique interaction for UPM 02 ................................................................... 109
Table 6.20: Program-technique interaction for UPM 03 .................................................................. 110
Table 6.21: Program-technique interaction for UPM 04 .................................................................. 111
Table 6.22: Program-technique interaction for UPM 05 .................................................................. 112
Table 6.23: Program-technique interaction for UPV 05 .................................................................. 113
Table 6.24: Program-technique interaction for ORT 05 .................................................................. 114
Table 6.25: Cluster analysis of overall program-technique interaction .............................................. 115
Table 6.26: Profile plot for fault ......................................................................................................... 116
Table 6.27: Profile plot for fault ......................................................................................................... 117
Table 6.28: UPM 01 fault-technique interaction .................................................................................. 119
Table 6.29: UPM 03 fault-technique interaction .................................................................................. 120
Table 6.30: UPM 05 fault-technique interaction .................................................................................. 121
Table 6.31: UdS 05 fault-technique interaction .................................................................................. 122
Table 6.32: Cluster behavior of the fault-technique interaction ......................................................... 123
Table A. 1 List, location and explanation of old excel worksheets ......................................................132
Table A. 2: List of data fields and corresponding sheets where they were located .......................133
Table A. 3: Continued list of data fields and corresponding sheets where they were located .........134
Table B. 1: Data fields of ‘subject data’ sheet .................................................................................135
Table B. 2: Data fields of ‘subject data’ sheet (contd.) .....................................................................136
Table B. 3: Data fields of ‘subject data’ sheet (contd.) .....................................................................137
Table B. 4: Data fields of ‘subject data’ sheet (contd.) .....................................................................138
Table B. 5: Data fields in ‘observable faults’ sheet ............................................................................139
Table B. 6: Data fields in ‘observable faults’ sheet (contd) .............................................................140
Table B. 7: Data fields in ‘observable faults’ sheet (contd) .............................................................141
Table B. 8: Data fields in ‘failure visibility’ sheet .............................................................................142
Table B. 9: Data fields in ‘failure visibility’ sheet (contd) .............................................................143
Table B. 10: Data fields in the ‘observed faults’ sheet .....................................................................144
Table B. 11: Data fields in the ‘observed faults’ sheet (contd) .........................................................145
Chapter 1

Introduction

Software quality has become and persistently remains a big issue among software users and developers. So, the importance of software evaluation cannot be overemphasized during software development. Software use has penetrated all phases of human endeavor, from heart monitors in the health sector to auto pilots in the aviation industry. The world continues to depend more and more on safety critical as well as time critical software to conduct its affairs. Thus, the cost of software failures is invaluable [5]. The dream of every software development organization is to develop software that meets the quality expectations of their customers within the quoted budget and time frame.

Software evaluation is the process of assuring or raising the quality standard of software or any of its components. It works by systematically studying the software to gather information about its quality [7]. This process is highly important as put by Harrold [19]. It entails examining and scrutinizing the developed product and judging if it meets the customer desired quality level. If it does, the development continues, otherwise, a rework is ordered to raise the product’s quality [20].

Regrettably, no amount of evaluation could completely assure software developers how their product will perform until they encounter a real situation [5]. Software evaluation is thus seen as an essential tool to demonstrate to or assure the client that the software is actually functional. In fact, it may not be realistic or cost effective to remove all software errors prior to product release [5].

So, it is crucial for developers to be able to choose from available evaluation techniques the one most suitable and likely to yield optimum quality results for different projects.

There are two main strategies of evaluating software:

- Static analysis
- Dynamic analysis or testing.
The term *testing* is often mistakenly used to describe the two strategies [18]. These two techniques differ mainly on the aspect or state of the artifact under evaluation – fixed state or operation mode.

Despite these classifications, it is still difficult today to determine when the use of a certain evaluation technique is more appropriate or advantageous than others in evaluating a piece of software. **This study analyzes and summarizes the results of a series of experiments geared towards determining when a type of software evaluation technique is more effective and thus more appropriate than the other.** This type of knowledge will save the software industry a lot of energy, time and cost.

The static evaluation techniques are commonly referred to as **reviews**. In static analysis, the product under evaluation is examined and scrutinized for inconsistencies and inaccuracy directly at rest. They are used to manually detect possible defects practically from any software artifact e.g., design document, specification document, program code etc. The technique is static in the sense that the hard copy or paper version of the artifact in question is scrutinized by reading through it.

The intention of the exercise is to discover as much of human errors in the artifact as possible. Reviews are useful in that they detect and remove defects, early in the lifecycle of a product. It is estimated that 30% - 70% of design and code defects are detected by review techniques [8]. More so, it is easier and less costly to fix mistakes at the early stage than later when dynamic techniques are applicable.

The static analysis can be implemented by several techniques based on the approach to **reading** the artifact [8].

Dynamic analysis or testing is used to evaluate the dynamic criteria of the product. That is, the product is executed and observed for incorrect and/or unsatisfactory behavior as judged against the expected operational quality level of the product. According to Torkar [21], testing is the process of ensuring that a certain piece of software artifact satisfies its requirements.

Testing is traditionally classified into two areas: structural testing and functional testing; both named based on how the tester viewed the software/artifact under evaluation.

### 1.1 Study Background

Software engineering, like other engineering fields, needs to formalize, standardize, create uniformity and have certain level of predictable functionality as well as accuracy in most of its
tools, methods and procedures. For example, software evaluation techniques as earlier mentioned are broadly divisible into two – static and dynamic techniques, with each having quite a few ways of implementing them. Nevertheless, the absolute benefits and drawbacks of each of the implementation techniques are still quite unknown or at best unclear [23]. Given certain circumstances of the software, the practitioners do not know what evaluation technique is best applicable for optimum quality assurance.

In order to achieve this feat, researchers are aiming for extensive and exhaustive empirical research in all areas, to underpin software engineering [3], since one of the basis for development in any discipline is empirical verification of knowledge [10, 4]. Software engineering researchers and practitioners are now taking advantage of empirical research, to validate their findings and work. Though, the popular strategy still involves (quasi-) controlled experiment and case study [9]. Over the years, the quality of the average empirical study in software engineering is increasing. Empirical study education (theory and practical) as it applies to software engineering is growing among researchers, consequently, the discipline is witnessing increasingly more comprehensive studies conducted on increasingly realistic programs and processes [11]. Most especially, there have been several experiments on software testing techniques as collated by Juristo et al. [23]. Specific guidelines on and introduction to conducting experiments in software engineering are discussed in [3, 4, 12, 13].

In line with this, with respect to code evaluation techniques, Basili and Selby [1, 2] initiated a study which ran through 1982, 83 and 84; the root of which is traceable to Hetzel [2, 17] and Myers [2, 15]. The experiment studied the effectiveness and efficiency of different code evaluation techniques. This experiment was replicated by Kamsties and Lott [2, 14] in 1995 as well as Wood et al in 1997 [2, 16]. This research follows in line, to contribute to advancing the available pragmatic knowledge on software evaluation techniques. It focus on analyzing and summarizing data from a series of experiments performed to compare the effectiveness of code review as an example of static technique and decision coverage and equivalence partitioning as examples of dynamic techniques.

In 2000, a group of researchers at the Universidad Politécnica de Madrid (UPM) in Spain started a series of experiments, designed to achieve the same aim as Basili. Eight of these experiments shall form the focus of this study. Five were performed at the UPM in 2001 through 2005, one each in 2005 at the Universidad Politécnica de Valencia (UPV), Spain, Universidad ORT Uruguay and Universidad de Sevilla (UdS), Spain.
1.2 Motivation

There is a need for the existence of a joint view of the results of these experiments. Each of the experiment in the series was conducted by different researchers, different year and in some cases different site; therefore, the result of the whole series exists in different formats – data classification, file structure, language used etc. The result of each of the experiments was independently gathered, analyzed and reported differently, according to the researcher in charge of the experiment. This does not help in having a joint view of all the results.

In view of the above, it can be established that consistency, good organization and structure are key factors that can aid understanding of the results reported from experiments involved in this type of experimental series. In as much as the primary goal is to compare the outcome of each experiment with the other or aggregate and deduce a common inference from them.

As it is, the following specific problems were identified with the current state of the experiment results:

- During the experiment, the subjects records their observation or response in a paper form; the researcher later extracts information from the form as data into excel file. The excel files do not exist in a common format, though they contain same information.
- In order to carry out statistical analysis on the experiments’ data, a pre-processing step is usually required to transform the paper form data into excel files and finally to SPSS.
- There exist report summary for individual experiments, however, there is no joint interpretation for all the experiments.
- The format of the directory for each of the experiment’s data and analysis is quite different from one to the other.
- The current files – excel sheets, does not have any definite structure.
- It is hard to get meaningful information out of the current structure.

1.3 Objectives and Research Questions

The main objectives of this thesis work, is to re-organize the data and analysis of all experiments in the series, interpret the outcome of each experiment and lastly, try to abstract some piece of knowledge based on the pooled result of all the experiments. These shall be achieved as follows:
a. Create a structured directory for the experiment data and analysis files
b. Gather all the experiment result files to a common location
c. Reorganize all the files to conform with a common structure
   i. Translate all file columns recorded in Spanish to English
   ii. Create a dictionary of fields to ensure definition mapping between the existing and new files
   iii. Ensure all files have corresponding fields
d. Create a common view for all the files, by ensuring that the excel files have equal and corresponding number of worksheets
e. Extract the statistical significance value – factors and interactions, from the ANOVA table of each experiment and for all the experiments in the series
f. Study the distribution pattern of all the effects based on whether it is significant at 0.01, 0.05 significant levels or not significant at both
g. Decide on the pattern of further interest to the study
h. Deduce a joint interpretation from the behavior of the main and interaction effects selected for study as mentioned above

During the preliminary work of this study, several questions were formulated in line with the objectives. The two main questions being:

How can the data records and analysis results of the series of experiments be organized to ensure uniformity and regularize deductions? And so far, what piece of information can be jointly deducted from the analysis results of all the experiments?

To be able to answer these two questions, several other questions needed to be answered first. These questions are listed below:

a. What is a suitable directory structure format for output from the experiments –data and analysis?
b. What can be done to make the results of the experiments similar and thus comparable?
c. What is the outcome of each replication of the experiment?
d. Is a joint overview of the ANOVA result for all the experiments possible?
e. What behavior is discernible in each factor across all the experiments?
f. What preliminary conclusions could be drawn from the joint analysis or abstraction of the replication results?

1.4 Approach to the Solution

The thesis will use the qualitative analysis approach which shall be in the following four steps:

1. Familiarize with the subject by reading texts on techniques and their application
2. Get familiar with the experiments. This will be achieved by running the three different evaluation techniques on the two versions of the three programs.
3. Familiarization with required statistical tools
4. Reorganize and re-structure all the existing result files.
5. Generate a common view by analyzing the results of the SPSS analysis for the experiments.
6. Obtain some conclusions through abstract interpretation of the joint data

It should be noted that in the process of obtaining the conclusion, we did not intend to apply a formal aggregation technique at this stage.

At the end of the study, we will be able to answer all the sub questions and invariably the two main questions. Below are the expected milestones or deliverables that will assist in this regard.

a. A uniform structured directory for all the experiments’ data and analysis.
b. Reorganization and restructuring of the existing data files for all the experiments to conform to a uniform naming convention, data interpretation and language use
   - A mapping and definition of fields between the existing excel files and the new ones – table.
   - Index of location for the different data fields – table.
c. Set of uniform, similar and comparable experiment result files – excel files.
d. Tables summarizing the significance values of the main and interaction effects from all the experiments.
e. Tables presenting the preliminary result of each factor’s behavior across all the experiments.
f. Joint abstractive interpretation from the analysis of the SPSS results
1.5 Thesis Structure

The rest of this report is structured as below:

In Chapter 2, some insight is given into the details of the design and conduct of the experiments. Also, we shall give a brief description of the experimental designs of each experiment.

Chapter 3 presents more details on the programs, techniques and faults inclusion and distribution; as factors of the experiments.

Chapter 4 shall present the report of the researcher’s personal experience, running the experiment to grasp the basics and understand the experiment better.

In Chapter 5, the common structure for the experiment data will be highlighted, also we shall present the structure of directories for the experiment data, the definition table for all data fields as well as the new look of the file format for the excel sheets which shall form the basis for the data extraction process. It will also present the pooled ANOVA result from SPSS analysis, qualitative analysis of the result and evidences generated from the result.

The outcome of the joint analysis of the experiments’ results shall be the focus of Chapter 6, while Chapter 7 will be focused on conclusions and recommendations for future research.
Summarizing the Results of a Series of Experiments: Application to Effectiveness of Software Evaluation Techniques

Babatunde Kazeem
Chapter 2

Description of the Experiments

The Universidad Politécnica de Madrid (UPM) started a series of experiments in 2001, to further investigate the effectiveness and efficiency of three software evaluation techniques. The material used for the experiments was extracted from Kamsties and Lott’s experiment [2], conducted in 1995. There are eight experiments in the series covered by this work. The series is termed Experiment II. There are other experiment series, for example, Experiment I and III which are outside the scope of this research. As mentioned in Chapter 1, three (UPV 05, ORT 05 and UdS 05) of the experiments in the series used for this research took place outside of the UPM. In each of these cases, the experimental settings (environment) were slightly different from the UPM. Therefore, some settings-induced changes were made as appropriate to the UPM experiment in order to adapt it. The design approach used for the experiments is termed Cross-over design, which is adapted to site. The UPM experimental package and those of UPV, UdS and ORT are further highlighted in the following sections.

2.1 UPM Experiments

The series of experiments in Experiment II was run to study the effectiveness and efficiency of the three evaluation techniques, most importantly, effect of the subjects on the effectiveness of the techniques. That is, to distinguish between situations when a subject failed to notice an exposed failure and when the technique is unable to produce a test case capable of exposing the failure.

2.1.1 Study Objectives

The experiment II was set up to investigate the under-listed three objectives:

- Study the influence of failure visibility: The non detection of a failure is usually due to either of two factors - the technique did not produce a test case capable of exposing the failure/fault or the subject (tester) did not noticed the fault when it showed up. Most
times, it is not clear which of these happened during testing exercise. Therefore, in order to establish situations when the non detection of a failure is due to the technique or the subject (in this case), subjects are given a set of pre-generated test cases, to expose all the faults. In this way, it will be clear what fault was exposed by each of the test cases generated through each of the techniques and how many were caught by each subject.

- Study fault detection ability of the techniques (effectiveness): Due to the modification above, it will be possible to monitor more reliably, any interaction between fault and technique. After the experiment, the original test cases developed by each subject are inspected to know what faults they can detect and this is recorded. This record is then compared to the actual failures the subjects observed and recorded while running the pre-generated test cases given to them.

- Study efficiency of the techniques: The efficiency of the techniques will also be studied in the experiments. The type of fault each detects most and how each technique behaves with the different types of programs and faults used for the experiment.

2.1.2 Hypothesis and Response Variable

The main aim of these series of experiments is to investigate the hypothesis whether or not the effectiveness of code evaluation techniques has anything to do with the fault types present in the program. Therefore, the response variable is effectiveness which is measured in terms of the number of subjects who detect a given fault for each fault in the program.

There are three basic factors used in Experiment II, technique, fault type and program type. However, since the program size was considered small, a fourth factor – version was introduced to increase the number of fault seeded into the programs.

To investigate how the combination of these factors influence fault detection, using each of the techniques. The experiments’ hypotheses were set as below [2]:

\[ H_{01} : \text{The fault detection technique, program and fault type do not impact effectiveness.} \]

\[ H_{11} : \text{The fault detection technique, program and fault type has an impact on effectiveness.} \]

\[ H_{02} : \text{The failures generated by the faults have no impact on its visibility.} \]

\[ H_{12} : \text{The failures generated by the faults have an impact on visibility.} \]

\[ H_{03} : \text{The fault detection technique, program and fault type do not impact efficiency.} \]

\[ H_{13} : \text{The fault detection technique, program and fault type has an impact on efficiency.} \]
2.1.3 Factors and Alternatives

As mentioned, the factors for this experiment are: fault type, program type, technique and version. Each is discussed in more details below:

- **Fault types:** The fault classification used in experiment II is based on the classification used by Basili in his experiment [2]. The introduction of program version facilitated the elimination of fault repetition, thus, each program contained seven distinct faults. The faults are:
  - F1: Cosmetic, omission.
  - F2: Cosmetic, commission.
  - F3: Initialisation, omission.
  - F4: Initialisation, commission.
  - F5: Control, commission.
  - F6: Control, omission.
  - F7: Computation, commission

See chapter three for more details on the fault categorization and inclusion in each program.

- **Technique:** The techniques for experiment II are code review by stepwise abstraction for the static technique, equivalence partitioning for the functional technique and decision coverage for the structural technique. The dynamic techniques were used in the following manner in experiment II:
  - The subjects applied the technique to generate test cases. These test cases were later analyzed to determine what faults they can detect.
  - The subjects were given test cases supplied by the experimenters to execute. This was to eliminate any bias in the earlier results obtained on the structural and the functional techniques.

- **Program:** Three programs were used in experiment II – *cmdline*, *nametbl* and *ntree* – these three came with the original experimental material used for the experiments [2].

- **Version:** Each program was seeded with the same number of faults (seven). Due to the fact that the program size is small, not much fault could be inserted into each. Therefore, two versions of each program marked by the fault they contain were produced for the experiments.
2.1.4 Experimental Design

The subjects are fifth-year computer science students, who were considered familiar with the techniques because of a course they must have taken in the fourth-year. The experiment was conducted using a design approach termed *Cross-over design*, as earlier mentioned. In this design, each subject will apply all the three techniques. The design is presented in Table 2.1 and 2.2 below. The subjects were in a group of 7 or 8. Each group was randomly assigned a program and a technique but the group members worked independently.

<table>
<thead>
<tr>
<th>Program</th>
<th>cmdline</th>
<th>ntree</th>
<th>nametbl</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique</td>
<td>CR</td>
<td>S</td>
<td>F</td>
</tr>
<tr>
<td>Group 1</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Group 2</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Group 3</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Group 4</td>
<td>-</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Group 5</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>Group 6</td>
<td>-</td>
<td>-</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2.1: Experiment design for experiment II in UPM

In table 2.1, ‘CR’ stands for code review, ‘S’ for structural technique and ‘F’ for functional technique. Also, ‘X’ indicates the assignment of each technique to the different groups. Apparently, all the groups exercised with all the techniques. One technique on one program executed as shown in Table 2.2.

<table>
<thead>
<tr>
<th>Day</th>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>cmdline</td>
<td>ntree</td>
<td>nametbl</td>
</tr>
<tr>
<td>Group 1, Group 2</td>
<td>Review</td>
<td>Structural</td>
<td>Functional</td>
</tr>
<tr>
<td>Group 3, Group 4</td>
<td>Review</td>
<td>Structural</td>
<td>Functional</td>
</tr>
<tr>
<td>Group 5, Group 6</td>
<td>Review</td>
<td>Structural</td>
<td>Functional</td>
</tr>
</tbody>
</table>

Table 2.2: Experiment execution for experiment II in UPM

2.1.5 Running the Experiment

The experiments were organized in three different sessions as shown in Table 2.2, the reason for the experiments was clear to all the subjects and they are also aware that the result will form part of their performance grading. The students were handed required documentation and asked to study
them, the training for the exercise was given as part of a course. No student had a prior knowledge of what technique or program they were going to work with before the experiment.

2.2 UPV Experiment

In order to conduct the same experiment at the UPV, all documentations used for the UPM experiment were transferred to the researchers at the UPV. Additionally, a series of meetings took place between the researchers of both schools to iron out gray areas and facilitate similarities in the replication of the experiments. Nevertheless, a few environmental constraints enforced some alteration in the design and execution of the experiment compared to the UPM’s package. The study objectives, hypothesis and response variable as well as factors and alternatives were the same as that of the UPM.

The following differences existed between the UPV environment and UPM environment as far as the experiment is concerned:

- Subjects are already acquainted with the techniques
- Less time was available to perform the experiment
- Less time available per session
- Training and execution of the experiment cannot be sequential

These four constraints lead to the four new conditions as shown in Table 2.3.

Due to the change in time, one of the techniques – code review – was left out of the experiment. Also, the test case execution was restricted to only one program. In terms of sessions, each session lasted for a maximum of two hours instead of four hours in UPM. Therefore, the test case generation and execution were divided into two sessions.

<table>
<thead>
<tr>
<th>NEW CONDITION</th>
<th>CHANGE ON EXPERIMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less time available</td>
<td>Code review technique left out</td>
</tr>
<tr>
<td></td>
<td>Test cases executed for just one of the programs</td>
</tr>
<tr>
<td>Two hours per session</td>
<td>Test case generation will be performed in one session</td>
</tr>
<tr>
<td></td>
<td>and test case execution in another session</td>
</tr>
<tr>
<td>Training and operation not sequential</td>
<td>One technique applied on all three programs in one session after the training on such technique</td>
</tr>
<tr>
<td>Subjects already acquainted with techniques</td>
<td>Refresher tutorial of 4 hours rather than 16 hours of lessons with the same material</td>
</tr>
</tbody>
</table>

Table 2.3: Environmental differences and adjustment in the UPV experiment [6]
Training was organized in the form of a refresher tutorial on the techniques in form of a practical exercise; this was merged with experiment execution [6].

2.3 UdS Experiment

As in the case of the UPV, the same set of documentation, used for the UPM experiments was transferred to researchers at UdS. Also, a series of meetings and discussions took place between the researchers at UPM and UdS to explain any ambiguity in the materials to enhance understanding. The unique environment of the UdS also forced the researchers to alter the UPM design to adapt it to their own environment. Apart from the environmental constraints, other things like the experiment objectives, hypothesis, factors and response variables remained as it was in UPM.

Five conditions were different between the UPM and UdS environment regarding this experiment. These are:

- Less time available to run the experiment
- Less time available per session of the experiment
- Training and operation not sequential
- Subjects were already acquainted with the techniques
- Insufficient computer systems

The test case execution could only be accommodated for just one of the programs due to less time available to conduct the experiment. Since only two hours is available per session of the experiment, test case generation will be performed separately to execution. The subjects had to work in pairs because there were not enough computers that could go round all subjects individually. Moreover, as against the UPM, in one session, training is given on one technique (dynamic) and the technique is subsequently applied on all the three programs. Lastly, only four hours of refresher tutorial was given to subjects because they were already familiar with the techniques.

The summary of conditions and changes is shown in Table 2.3 below:
Chapter 2: Description of the Experiments

2.4 ORT Experiment

As in the UdS and UPV experiments, the UPM package was used for this experiment with modification as required due to new environmental conditions.

In this case, there were three constraints:

- Less time available
- Computer room not available
- Subjects are junior with no programming experience

Due to short time available for the experiment, code review was left out from the techniques, so, only the dynamic techniques were used. Also, one program – ntree - was left out. Moreover, the experiment took place in only one session with unlimited time given to the subjects to apply two techniques on two programs [6]. The test case execution part of the experiment could not be conducted because the computer rooms were not available.

<table>
<thead>
<tr>
<th>NEW CONDITION</th>
<th>CHANGE ON EXPERIMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less time available</td>
<td>Test cases executed for just one of the programs</td>
</tr>
<tr>
<td>Two hours per session</td>
<td>Test case generation will be performed in one session and test case execution in another session</td>
</tr>
<tr>
<td>Training and operation not sequential</td>
<td>One technique applied on all three programs in one session after the training on such technique</td>
</tr>
<tr>
<td>No sufficient computer systems</td>
<td>Subjects work in pairs</td>
</tr>
<tr>
<td>Subjects already acquainted with techniques</td>
<td>Refresher tutorial of 4 hours rather than 16 hours of lessons with the same material</td>
</tr>
</tbody>
</table>

Table 2.4: Environmental differences and adjustment in the UdS experiment [6]

<table>
<thead>
<tr>
<th>NEW CONDITION</th>
<th>CHANGE ON EXPERIMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less time available</td>
<td>Code review technique left out</td>
</tr>
<tr>
<td></td>
<td>Experiment ran in one session</td>
</tr>
<tr>
<td></td>
<td>One of the programs left out</td>
</tr>
<tr>
<td>Computer room not available</td>
<td>Test case execution left out</td>
</tr>
<tr>
<td>Junior subjects without programming language experience</td>
<td>Junior subjects</td>
</tr>
</tbody>
</table>

Table 2.5: Environmental differences and adjustment in the ORT experiment [6]

The subjects are junior students with no prior knowledge of programming; there was no way to adjust to this difference [6].

The new conditions and adjustments is shown in Table 2.5
Chapter 3

Description of Programs, Techniques and Faults

As mentioned in the previous chapter, three techniques – code review, equivalence portioning and decision coverage; three programs – *cmdline*, *nametbl* and *ntree* – and seven fault types were used in experiment II that is analyzed in this study. In this chapter, the programs shall be described in more detail and a further description of the faults and the expected failure shall be given.

3.1 Description of the Programs

The three programs used in these experiments were the ones created by Kamsties and Lotts [2], also used by Wood et al., [2]. The programs are basically of two types: functional (*cmdline*) and data (*nametbl* and *ntree*) programs. For the sake of these experiments, each program type used is further classified into two versions – version 1 and 2 – marked by the fault each contained. The programs are described in the following sections.

3.1.1 Cmdline

This is a program that reads an input line and output the summary of its input. It analyzes the input for syntactic and partially for semantic correctness. The user is expected to input at least one ‘measure’ option and one file to the program. In addition, one ‘search’ option can also be included in the input. The order of the search option and the measure option does not matter. The first input that is not a search or measure option is treated as a file. This program is 308 source lines of code (LOC) long, including whitespaces. The input line reads as shown below:

```
cmdline – measure <MEASURE> [Search-option] File [File...]
```

The inputs in square bracket are optional. If the program found the input to be correct, a summary of the arguments is printed out. The summary will consist of the measure option given,
the search option if any and the list of the filenames. Otherwise, an explanatory error message is displayed.

The input commands are:

- `-me[asure] <MEASURE>`: The acceptable measures are: GKO[M], LKO[M], GKH[M], LKH[M], GI[HE], and LI[HE].
- Search options: The valid search option input are: -al[l], -ma[x], -mi[n], -av[erge], -be[low] <value> and -ab[ove] <value>.
- Help: The help commands are: -? and –h[elp].

The characters in square brackets are optional. Also, the value in search option must be a real number.

### 3.1.2 Nametbl

This program implements the data structure of a symbol table, as well as its operations for a certain computer language. It reads commands from a file and processes them to test a few functions. The symbol table stores the following information:

- Symbol name.
- Object type of the symbol: {OT_NO_INF, SYSTEM, RESOURCE}
- Resource type of the symbol: {RT_NO_INF, RT_SYSTEM, FUNCTION, DATA}

The command-line input reads:

```
nametbl <input-file>
```

The following are the input data:

- `ins <symbol>`: This command inserts the symbol into the table. If it already exists, an error message is displayed. The default object and resource type for the symbol is: OT_NO_INF and RT_NO_INF respectively.
- `tot <Symbol> <ObjTyp>`: This command enters the object type (SYSTEM or RESOURCE) for the <symbol>. If any other value is provided or the symbol is not in the table, a corresponding error message will be generated.
• trt <Symbol> <ResTyp>: This enters the resource type (RT_SYSTEM, FUNCTION or DATA) for the <symbol>. The value RT_SYSTEM is used only when the object type has the value SYSTEM. An error message is generated if any other value is used or the symbol is not found in the table.

• sch <Symbol>: It searches for <symbol> in the table and prints out corresponding information about it. If not found an error message is printed.

• prt: Prints out the content of the table.

The length of this program is 342 LOC including whitespaces.

3.1.3 Ntree

This program implements the data structure of n-ary tree, as well as its operations. The program together implements the functions of a tree, in which each node can have any number of child nodes. A node of the tree holds two strings (key and content). This program is 271 LOC in length including whitespaces.

It reads its input from a file supplied in the command-line argument:

ntree <input-file>

The commands for manipulating the programs are:

• root <key> <content>: This command must be the first line of the input file. It creates the root of the tree.

• child <father-key> <child-key> <child-content>: Creates a child and attaches it to the father node. If the father is not found, an error message is displayed. If the key to the father node is duplicated in the tree, the child is attached to the first node found.

• search <key>: This command searches the tree using depth-first search. If the node with the given key is found, the contents are printed out, if not, an error message is displayed. In case there are two nodes with the key, only the contents of the first one are printed.

• sibs <key1> <key2>: Checks the tree whether the nodes with key1 and key2 shares the same father node. If any of the nodes is not found, an error message is displayed.
• print: Prints the contents of the tree. Each node (key, contents and depth) is printed on a new line. Each line is indented to correspond to the depth of the node (4 blank space per level)

The contents of the file including the commands must not exceed 1024 characters. Also the program is case sensitive to the commands, each command must start on a new line and the content string cannot contain white space because it is used as command separator in the program.

3.2 Description of the Evaluation Techniques

Generally speaking as discussed in Chapter 1, software evaluation techniques is broadly divisible to static and dynamic analysis techniques.

The main classifications of the static analysis according to Juristo et al. [8] are:

• Informal review: This is just an informal exchange of opinions among colleagues.
• Inspection: Otherwise known as reading. It takes an organized and systematic approach to evaluating any artifact, involving several participants with defined roles. A report of the process is produced at the end of the activity.
• Walkthrough: A type of review that involved simulation of test cases for the program under evaluation.
• Audits: Audits evaluates software artifacts against established standards and organizational tradition.

Each of these classifications has several techniques that use the approach. Reading by stepwise abstraction – an inspection approach was the review technique used in the experiments of this study.

The dynamic analysis otherwise known as testing is traditionally classified into two: structural testing and functional testing based on how the tester viewed the software artifact under evaluation.

Structural testing, also known as white box testing, is a family name for several techniques based on the white box approach.

In structural testing, the tester uses his knowledge of the software internal structure, to figure out test data and develop test cases (Figure 3.1). A few of the techniques under this family include:

• Path coverage: Testing of all the possible paths across the program. A path refers to a linked set of statements performing a focused task from entry to exit point.
• Decision coverage: Testing each decision point in the program for a false or true situation
• Statement coverage: Testing of all the program statements at least once.

![Decision coverage example](image)

**Figure 3.1:** Transparent view in white box testing reveals internal parts [21]

Functional testing otherwise known as black box testing uses the opposite approach to test software. The external or general knowledge of the software is used to develop test cases for the software. No recourse is made to the internal structure of the product. The product is viewed as a whole (Figure 3.2).

![Functional testing example](image)

**Figure 3.2:** Black box testing uses external product view [21]

All the techniques based on this method have this approach in common. Few of them are:

• Boundary value analysis: This approach develops test cases by using the boundary or edge values of the input class. That is, values around the smallest and the highest input values.
• State diagram: A type of diagram used to give an abstract description of the system behavior. It is expected that the system must consist of a finite number of states.
• Cause effect graph: This method uses a set of directed graphs to map a set of inputs to corresponding set of outputs.
Partition testing: In this technique, the input space of a system is classified into data partitions. Test cases are the developed to cover each partition at least once.

3.2.1 Review Technique

Reading by abstraction or abstract interpretation was the review technique used during the experiments. This method converts the code into textual statements of what the program does. This text is then compared to the original program specification to establish correctness, completeness and consistency.

3.2.1.1 Abstractive Reading

In inspection, a team of experts charged with the responsibility of ensuring the quality of a software product choose a reading technique for the purpose of detecting defects. A defect in this sense is any deviation from predefined quality properties of the product. There are several types of reading techniques, one of which is abstractive reading. This method is applicable only to codes.

In abstractive reading, the purpose is to compare the program specification to what the program actually does [8]. This comparison is impossible unless the program (code) is converted to specification – a text format. The summary (Figure 3.3) of the conversion steps are:

- Read the code to grab the general idea
- Establish dependencies among program functions
- Understand what each function does
- Correlate each function’s behavior with the established relationship to understand the behavior of the whole program described in text.
- Compare the description with the program specification. Note discrepancies and list them.

![Figure 3.3 Summary of steps in reading by abstraction](image-url)
A detailed description of these steps can be found in [8].

3.2.2 Dynamic Techniques

As discussed in Chapter 1, the dynamic techniques are used to exercise the dynamic behavior of the software product. It simulates the real life situation for the program. The idea is to nip any impending problem in the program on the bud, using testing before it gets to the field.

Generally, testing is an activity to evaluate the reliability of a software system or any of its component part, by running it under specific conditions, to observe how it behaves. Its behavior is recorded and compared to its expected behavior to determine failures. The failures are then traced to faults in the code for debugging (Figure 3.4). This usually necessitates further testing.

The two dynamic techniques used in the experiments are: decision coverage and equivalent class partition.

![Figure 3.4: Steps in software testing](image)

3.2.2.1 Equivalence Partitioning

The equivalent class partition testing [4, 21], strives to be as frugal as possible with test case. A tester’s dream is to be as exhaustive as possible with minimal or no redundancy. This is something achievable with partition testing, without losing the feeling of completeness. It works by dividing the program inputs space into partitions. All the elements in each partition are considered similar, as far as the program operations is concerned. Therefore, a member is selected and tested in each partition, with the assumption that testing all will yield the same behavior. Thus, any fault, not uncovered by a member, cannot be uncovered by any member of the partition.

The basic steps of this technique are two:
1. Identify the classes
2. Develop test case based on each class

3.2.2.2 Decision Coverage

In decision coverage [4], the decision points in the program are of interest. This includes conditional statements and loop constructs. Test cases are developed based on the predicates of the decision statements. Each decision point is exercised at least once for true and false conditions.

3.3 Description of the Fault Types

The fault types in the programs used in these experiments followed the classification used by Basili in the first experiment of the series [2]. He classified faults into six types, under two basic categories. He basically made a distinction between something that is missing which he called omission and something that is incorrect which he tagged commission [1]. Therefore, he identified six fault types of omission and six fault types of commission. Four of these fault types were used in Experiment II. The four fault types used in are highlighted below; all faults are described for commission and omission:

- **Initialization**: The initialization fault corresponds to an incorrect initialization of a variable or data structure. For example, assigning a value erroneously to a variable within a module would be an initialization error of commission, while failure to initialize when required would be an initialization error of omission.

- **Cosmetic**: Cosmetic faults usually result from unwarranted mistakes in operations or messages within a program. For example, a spelling mistake in an error message would be a cosmetic error of commission; whereas, the absence of error messages, where there should be one, would be a cosmetic error of omission.

- **Control**: In any given situation, if the program follows an incorrect control flow path, results in a control fault. It is a control fault of commission if the predicate in an if-then-else statement is incorrect. It is omission if the predicate is missing.

- **Computation**: These faults have to do with inaccurate calculations. The use of a wrong arithmetic operator would result in an error of commission while the absence of operator within a mathematical expression will be an omission.
The omission error of computation was not used in Experiment II. There are other fault types in the classification of Basili but are not included in this enumeration because they were not used in the experiments. These are interface and data error (commission and omission).

3.4 Specific Fault Description and Failure Characterization

Each of the programs used contains seven faults, faults 1 to 7 namely: Omission, cosmetic; Commission, cosmetic; Omission, initialization; Commission, initialization; Commission, control; Omission, control and Omission, computation respectively. The following subsections will describe the fault distribution and failure characterization in each program.

3.4.1 Faults in Cmdline

In this section, a further insight will be provided as to the particular faults seeded into the two versions of the cmdline program used in the experiment.

3.4.1.1 Version 1

The seven faults in the version 1 of the cmdline program are as follows:

- **Fault 6 (Control, omission):** In the version 1 of the program, this fault is located on line 25. The ‘-1’ at the end of the statement (Listing 3.1) is unnecessary.

```
[..........]
24 static int keyword_table_size = sizeof(keyword_table) /
25       sizeof(struct keyword_entry)-1;
[..........]
```

Listing 3.1: Fault 6 in cmdline version 1

This will reduce the options stored in the table by one. The last entry will never be stored. Therefore, the search command ‘-above’ which is the last in the list, is not recognized by the program.

- **Fault 2 (Cosmetic, commission):** In line 31 of the cmdline program, the word ‘option’ in the error message was misspelled. In version 1 (Listing 3.2), the letter ‘p’ is missing, notable also is the fact that the excerpt is part of a method with 6 LOC.
.. code-block:: c

    void usage(char *progn)
    {
        fprintf(stderr,
            "Usage: %s -measure <MEASURE> [ Search-otion ] File [ File ... ]\n", prog);
        exit(1);
    }

Listing 3.2: Fault 2 in `cmdline` version 1

The message containing this error pops up whenever the user request for help message on the usage of the command line arguments.

- **Fault 7 (Computation, commission):** In line 68 of version 1, the search data structure was assigned an unnecessary ‘null’ value (Listing 3.3).

    .. code-block:: c

        void print_command(struct command *cmd)
        {
            if ((cmd->measure == NULL) && (cmd->search != NULL))
                fprintf(stderr, "No measure option given\n");
            else if (strcmp(cmd->measure, "-help") == 0)
                usage("cmdline");

Listing 3.3: Fault 7 in `cmdline` version 1

This assignment renders the conditional statement useless; a missing ‘measure’ option is not recognized as an error, if a search option is used. Hence, the error message will not be displayed.

- **Fault 4 (Initialization, commission):** Inside a ‘switch’ statement on line 139 of version 1, a data structure was wrongly assigned the command ‘LKOM’ instead of ‘LKH’ (Listing 3.4).

    .. code-block:: c

        case LKHM:
            cmd->measure = "LKOM";
            break;

Listing 3.4: Fault 4 in `cmdline` version 1
Therefore, the ‘switch case’ for ‘LKHM’ is always wrongly processed. Report of ‘LKOM’ will always be given in lieu of ‘LKHM’.

- **Fault 3 (Initialization, omission):** A ‘break’ statement was omitted inside a ‘switch’ block. This error occurred on line 193 (Listing 3.5), ‘switch case’ for the ‘minimum’ search option. An attempt to use this search option will result in an error message, saying too many search options.

```c
186 case MIN:
187     if (cmd->search != NULL) {
188         rc = -1;
189     }
190  
191     else
192         cmd->search = "-min";
193  
194 case BELOW: [......]
```

Listing 3.5: Fault 3 in *cmdline* version 1

- **Fault 5 (Control, omission):** On line 188 of version 1, using the code excerpt for fault 3 (Listing 3.5), an error message – “too many search options given”, is missing. Thus, using several search options with ‘minimum’ will not output an error message. Nevertheless, this error is concealed by fault 5.

- **Fault 1 (Cosmetic, omission):** A counter variable ‘i’ was not initialized to 0 before being used on line 89 (Listing 3.6).

```c
87    { 
88      int i;
90      if (argi == 0)
91          printf("No input files given\n");
92      else if (argi > 0)
93          { 
94          printf("Number of input files: %d\n", argc - argi);
95          printf("Input files are:\n");
96          while (i < argc)
97              printf(" %s\n", argv[i++]);
98          }
[......]
```

Listing 3.6: Fault 7 in *cmdline* version 1
This causes the program to print the names of the input files incorrectly.

A summarized description of the faults and associated failures is shown in Table 3.1.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault type</th>
<th>Fault position</th>
<th>Fault description</th>
<th>Expected failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cosmetic, omission</td>
<td>Line 89</td>
<td>The variable ‘i’ is not initialized to 0</td>
<td>Prints the names of the input files incorrectly</td>
</tr>
<tr>
<td>2</td>
<td>Cosmetic, commission</td>
<td>Line 31</td>
<td>In the word “Option”, the ‘p’ is missing</td>
<td>Use message contain spelling mistake</td>
</tr>
<tr>
<td>3</td>
<td>Initialization, Omission</td>
<td>Line 193</td>
<td>The “break” statement is missing</td>
<td>If the “minimum” option is given, an error message complaining about too many</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>search options appears</td>
</tr>
<tr>
<td>4</td>
<td>Initialization, commission,</td>
<td>Line 139</td>
<td>‘LKOM’ is assigned instead of ‘LKHM’</td>
<td>The switch LKHM is not processed correctly; instead, the user always receives</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>a report for LKOM</td>
</tr>
<tr>
<td>5†</td>
<td>Control, commission</td>
<td>Line 188</td>
<td>Error message “Too many search options given” is missing.</td>
<td>The “minimum” option with several search options does not output error</td>
</tr>
<tr>
<td>6</td>
<td>Control, omission</td>
<td>Line 25</td>
<td>The “-1” is superfluous</td>
<td>The last entry in the table is never found. The option “-above” is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>recognised</td>
</tr>
<tr>
<td>7</td>
<td>Computation, commission</td>
<td>Line 68</td>
<td>The expression (cmd-&gt;search == NULL) is superfluous</td>
<td>A missing measurement option is not recognized as an error if a search option is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>used</td>
</tr>
</tbody>
</table>

Table 3.1: Fault description for `cmdline` version 1

### 3.4.1.2 Version 2

In this section, we will describe the corresponding seven faults in the `cmdline` version 2.

- **Fault 6 (Control, omission):** In version 2, the fault is located on line 98. Pre-incrementing ‘i’ (Listing 3.7) is wrong; it was supposed to be post incremented.

```
96 for (i = argi ; i < argc; )
97   printf(" %s\n", argv[++i]);
98 }
```

Listing 3.7: Fault 6 in `cmdline` version 2

† This failure is masked by fault 5
• **Fault 2 (commission, cosmetic):** In line 244 (Listing 3.8) of the `cmdline` program, the word ‘option’ in the error message was misspelled. The error message concerned is part of a method with 152 LOC. The length of the method might have effect in code reading or decision coverage. The word is part of error message displayed as a result of an incorrect use of the command arguments.

```c
[......]
243 if ((rc < 0) || (position == 0))
244    fprintf(stderr, "Otion arguments are incorrect
");
[......]
```

Listing 3.8: Fault 2 in `cmdline` version 2

• **Fault 4 (Initialization, commission):** On line 243 (Listing 3.8), a variable ‘position’ was assigned 0. This assignment was unnecessary and thus, the program will fail to output an error message if it is not given any input file.

• **Fault 7 (Computation, commission):** On line 89 (Listing 3.9), the variable ‘number’ was not initialized before being used. Therefore, a wrong number – system’s default for integer - is always reported as the number of input files for the program.

```c
[......]
88 int i, number;
89
90  if (argi == 0)
91      printf("No input files given\n");
92   else if (argi > 0)
93   {
94      printf("Number of input files: %d\n", number); [......]
```

Listing 3.9: Fault 7 in `cmdline` version 2

• **Fault 3 (Initialization, omission):** A ‘break’ statement was omitted inside a ‘switch’ block on line 118 (Listing 3.10) for the ‘help’ command. An attempt to use this search option will result in an error message, saying too many search options.

```c
[......]
115 case HELP:
116    cmd->measure = ":-help";
117    position = -1;
118```
Whenever the ‘-help’ command is used the message: “too many search options given and Option arguments are incorrect” is output. This indicates that the ‘-help’ measure is handled incorrectly.

- **Fault 5 (Control, commission):** This fault is on line 151 (Listing 3.11), an error message, “fprintf(stderr, “Invalid measure option given\n”)” is missing in the default case of the ‘switch’ block.

```c
default:
151
152  rc = -1;
153  break;
```

Listing 3.11: Fault 5 in `cmdline` version 2

Therefore, no error message is displayed if an invalid measure option is used.

- **Fault 1 (Cosmetic, omission):** A data structure was created on line 7 to 22 as shown in Listing 3.12, to define the program commands. The entry: {“-superior”, 3, SUP} is missing, thus, the program will not recognize ‘-above’ as a measure option, it is rather treated as a file name.

```c
static struct keyword_entry keyword_table[] = {
7  {"-?", 2, HELP},
8  {"-help", 2, HELP},.................
9  {"-min", 3, MIN},
10 {"-below", 3, BELOW},
11 };
```

Listing 3.12: Fault 7 in `cmdline` version 2

The summary of the faults, fault type, fault location and the expected failure associated with the failures of the `cmdline` version 2 is shown in Table 3.
### Table 3.2: Fault description for `cmdline` version 2

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault type</th>
<th>Fault position</th>
<th>Fault description</th>
<th>Expected failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cosmetic, omission</td>
<td>Line 21</td>
<td>The last table entry is missing [“-superior”, 3, SUP]</td>
<td>Does not recognise the “-above” option</td>
</tr>
<tr>
<td>2</td>
<td>Cosmetic, Commission</td>
<td>Line 244</td>
<td>In the word “Option”, the ‘p’ is missing</td>
<td>Error message containing spelling mistake</td>
</tr>
<tr>
<td>3</td>
<td>Initialization, omission</td>
<td>Line 118</td>
<td>The “break” statement is missing</td>
<td>The “-help” option is incorrectly interpreted</td>
</tr>
<tr>
<td>4</td>
<td>Initialization, commission</td>
<td>Line 241</td>
<td>The expression (position ==0) is superfluous</td>
<td>Does not output error message when there is no input file</td>
</tr>
</tbody>
</table>
| 5     | Control, commission | Line 151       | The sentence fprintf(stderr, “Invalid measure option given
”) is missing          | Does not output error message when there is an invalid measure option |
| 6     | Control, commission | Line 98        | ‘i’ should not be pre-incremented, but post-incremented                          | Prints the names of the input file incorrectly         |
| 7     | Computation, commission | Line 89      | Variable number should be initialized to ‘arge-argi’                            | Prints the number of input files incorrectly           |

#### 3.4.2 Faults in Nametbl

As mentioned earlier, this program is a data program which implements the data structure of a symbol table and its operations. The program reads commands from a file and processes them in order to test a few functions. It uses a symbol table to store:

- Symbol name
- Object type of the symbol: {OT_NO_INF, SYSTEM, RESOURCE}
- Resource type of the symbol: {RT_NO_INF, RT_SYSTEM, FUNCTION, DATA}

This section will present the faults and expected failures in both versions of the program.

#### 3.4.2.1 Version 1

Below is the list of the seven faults in the version 1 of the nametbl program.

- **Fault 1 (Cosmetic, omission):** An error message was missed in line 189 (Listing 3.13). The code piece is extracted from the method that processes the resource type. Due to the missing message, no error message will be displayed if the resource type given is invalid.

```c
[.......]
185 else if (strcmp("FUNCTION", new_rt_char) == 0)
186     new_rt = FUNCTION;
```
Fault 1 (commission, cosmetic):

In version 1 of the `nametbl` program, the word “Table” was spelled wrongly (Listing 3.14) in an error message. Therefore, if an object type is assigned to an element that was not in the table, this error message pops up with table spelled wrongly.

```c
187 else if (strcmp("DATA", new_rt_char) == 0)
188     new_rt = DATA;
189
190 if (new_rt == -1) {
191  .......
Listing 3.13: Fault 1 in `nametbl` version 1
```

Fault 2 (commission, cosmetic):

In version 1 of the `nametbl` program, the word “Table” was spelled wrongly (Listing 3.14) in an error message. Therefore, if an object type is assigned to an element that was not in the table, this error message pops up with table spelled wrongly.

```c
164 else {
165    result = set_obj_type(nt, new_name, new_ot);
166    if (result)
167        printf("The name \"%s\" is not in the Tabl.\n", new_name);
169 }
[........]
Listing 3.14: Fault 2 in `nametbl` version 1
```

Fault 3 (Initialization, omission):

On line 98 (Listing 3.15) of the version 1, the variable ‘result’ was supposed to be initialized before use. The variable is a placeholder for stored object type. Consequently, an object type is stored but an error message saying that the element is not in the table is returned anytime the user attempts to retrieve the stored object type.

```c
98 int result;
99  NTE *nte;
102 nte = retrieve_entry(nt, search_name);
103 if (nte != NULL)
104    nte->ot = new_ot;
104 else result = -1;
108 return result; [......]
Listing 3.15: Fault 3 in `nametbl` version 1
```

Fault 4 (Initialization, commission):

In an attempt to define the data structure in the header file of the program, two variable names were swapped for each other. On line 22 and 23(Listing 3.16), ‘FUNCTION’ was swapped for ‘DATA’. If the user assigned the resource type of a symbol to be ‘FUNCTION’, ‘DATA’ is assigned by the program. Therefore, when
the program prints the input summary, ‘DATA’ will be printed for a symbol of resource type ‘FUNCTION’.

[......]
14 enum resource_type {
15    RT_SYSTEM,
16    FUNCTION,
17    DATA,
18    RT_NO_INF};
20 static char *resource_type_name[] = {
21    "RT_SYSTEM",
22    "DATA",
23    "FUNCTION",
24    "RT_NO_INF"};
[......]
Listing 3.16: Fault 4 in nametbl version 1

• **Fault 5 (Control, commission):** On line 158 (Listing 3.17), the word “RESOURCE” was misspelled in the predicate of an ‘if’ statement. Thus the program will not recognize a ‘RESOURCE’ object type.

[......]
158 else if (strcmp("RESSOURCE", new_ot_char) == 0)
159       new_ot = RESOURCE;
[......]
Listing 3.17: Fault 5 in nametbl version 1

• **Fault 6 (Control, omission):** In version 1, a curly bracket was missing (Listing 3.18) after the ‘if’ statement in a method used to set the resource type of the symbol. As a result, the program will always assign a resource type to a non-existing symbol at the request of the user.

[.....]
120 nte = retrieve_entry(nt, search_name);
121 if (nte != NULL)
122   result = 0;
123   nte->rt = new_rt;
[.....]
Listing 3.18: Fault 6 in nametbl version 1
- **Fault 7 (Computation, commission):** In an attempt to increment the number of elements in the table, the developer multiplied the element content by 1 (Listing 3.19) instead of adding 1 to the content. This causes the program to keep wrong count of the input symbols and thus output a wrong value for the elements in the table.

```c
[.....]
59  { ..........
63  assert(nte != NULL);
65  nte->name = strdup(new_name);
66  assert(nte->name != NULL);
67  nte->ot = new_ot;
68  nte->rt = new_rt;
69  (nt->numitems) *= 1;[.....]
```

Listing 3.19: Fault 7 in `nametbl` version 1

Below, the faults description location and associated failures is presented in Table 3.3.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault name</th>
<th>Fault position</th>
<th>Fault description</th>
<th>Expected failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Omission, cosmetic</td>
<td>Line 189</td>
<td>Missing error message</td>
<td>No error message is printed if the resource type given is unknown</td>
</tr>
<tr>
<td>2</td>
<td>Commission, cosmetic</td>
<td>Line 168</td>
<td>The word “Table” is misspelled</td>
<td>If an object type is assigned to an element that is not in the table, the error message is misspelled</td>
</tr>
<tr>
<td>3</td>
<td>Omission, initialization</td>
<td>Line 98</td>
<td>The variable “results” should be initialized</td>
<td>The value for the object type is successfully stored, but an error message is printed saying that the element is not in the table</td>
</tr>
<tr>
<td>4</td>
<td>Commission, initialization</td>
<td>Line 22 &amp; 23 in <code>file.h</code></td>
<td>The constants “FUNCTION” and “DATA” are swapped</td>
<td>If the resource type of a symbol is entered to be “FUNCTION”, the assigned value is “DATA”</td>
</tr>
<tr>
<td>5</td>
<td>Commission, control</td>
<td>Line 158</td>
<td>The word “RESOURCE” is misspelled</td>
<td>Object type “RESOURCE” is not recognised</td>
</tr>
<tr>
<td>6</td>
<td>Omission, control</td>
<td>Line 122 &amp; 130</td>
<td>The curly brackets are missing</td>
<td>The resource type is always assigned, even though the symbol is not in the table</td>
</tr>
<tr>
<td>7</td>
<td>Omission, computation</td>
<td>Line 69</td>
<td>The number of elements in the table is not incremented</td>
<td>Prints incorrectly the number of elements in the table</td>
</tr>
</tbody>
</table>

Table 3.3: Details of fault in `nametbl` version 1
3.4.2.2 Version 2

Like the previous section, the faults in the version two of the program are described in this section.

- **Fault 1 (Cosmetic, omission):** An error message was missed on line 162 (Listing 3.20). The message was to be displayed for an invalid object type.

  ```c
  new_ot = -1;
  if (strcmp("SYSTEM", new_ot_char) == 0)
    new_ot = SYSTEM;
  else if (strcmp("RESOURCE", new_ot_char) == 0)
    new_ot = RESOURCE;
  if (new_ot == -1) {
    result = -1;
  }
  ```

  Listing 3.20: Fault 1 in namethl version 2

- **Fault 2 (Cosmetic, commission):** The word “Table” was spelled wrongly (Listing 3.21) in an error message on line 197. In this case, the message is expected to be displayed when a resource is assigned to an element that was not in the table.

  ```c
  else {
    result = set_res_type(nt, new_name, new_rt);
    if (result)
      printf("The name `%s' is not in the Table.\n", new_name);
  }
  ```

  Listing 3.21: Fault 2 in namethl version 2

- **Fault 3 (Initialization, omission):** On line 116 (Listing 3.22), the variable ‘result’ was supposed to be initialized before use. The variable is a placeholder for stored resource type. Consequently, an error message saying a stored resource type is not in the table is printed during an attempting to retrieve it.

  ```c
  int result;
  NTE *nte;
  ```
Babatunde Kazeem

Listing 3.22: Fault 3 in nametbl version 2

• Fault 4 (Initialization, commission): In an attempt to define the data structure in the header file of the program, two variable names were swapped for each other. On line 10 and 11 (Listing 3.23), ‘SYSTEM’ was swapped for ‘RESOURCE’. If the user assigned the object type of a symbol to be ‘RESOURCE’, ‘SYSTEM’ is assigned by the program. Therefore, when the program prints the input summary, ‘SYSTEM’ will be printed for a symbol of object type ‘RESOURCE’.

Listing 3.23: Fault 4 in nametbl version 2

• Fault 5 (Control, commission): On line 185 (Listing 3.24), the word “FUNCTION” was spelled wrongly in the predicate of an ‘if’ statement. An object type ‘FUNCTION’ will not be recognized by the program.

Listing 3.24: Fault 5 in nametbl version 2
• **Fault 6 (Control, omission):** On line 104 – 105 (Listing 3.25), a curly bracket was missing after the ‘if’ statement in a method that assigns object type to existing symbols in the table. As a result, the program will always assign an object type to symbols that do not exist in the table.

    [.....]
    102  nte = retrieve_entry(nt, search_name);
    103  if (nte != NULL)
    104    result = 0;
    105    nte->ot = new_ot;
    [.....]

Listing 3.25: Fault 6 in *nametbl* version 2

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault name</th>
<th>Fault position</th>
<th>Fault description</th>
<th>Expected failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Omission, cosmetic</td>
<td>Line 162</td>
<td>Missing error message</td>
<td>No error message is printed if the object type given is unknown</td>
</tr>
<tr>
<td>2</td>
<td>Commission, cosmetic</td>
<td>Line 197</td>
<td>The word “Table” is misspelled</td>
<td>If an resource type is assigned to an element that is not in the table, the error message is misspelled</td>
</tr>
<tr>
<td>3</td>
<td>Omission, initialization</td>
<td>Line 116</td>
<td>The variable “results” should be initialized</td>
<td>The value for the resource type is successfully stored, but an error message is printed saying that the element is not in the table</td>
</tr>
<tr>
<td>4</td>
<td>Commission, initialization</td>
<td>Line 10 &amp; 11 in file.h</td>
<td>The constants “SYSTEM” and “RESOURCE” are swapped</td>
<td>If the resource type of a symbol is entered to be “SYSTEM”, the assigned value is “RESOURCE”</td>
</tr>
<tr>
<td>5</td>
<td>Commission, control</td>
<td>Line 185</td>
<td>The word “FUNCTION” is misspelled</td>
<td>Object type “FUNCTION” is not recognised</td>
</tr>
<tr>
<td>6</td>
<td>Omission, control</td>
<td>Line 104 - 105</td>
<td>The curly brackets are missing</td>
<td>The object type is always assigned, even though the symbol is not in the table</td>
</tr>
<tr>
<td>7</td>
<td>Omission, computation</td>
<td>Line 69</td>
<td>The number of elements in the table is not incremented</td>
<td>Prints incorrectly the number of elements in the table</td>
</tr>
</tbody>
</table>

Table 3.4: Description of faults in *nametbl* version 2

• **Fault 7 (Computation, commission):** The number of elements in the table was incorrectly incremented (Listing 3.26). ‘2’ was used as the incrementing factor instead of ‘1’. Therefore, the number of items in the table is incremented by two each an element is added resulting in a wrong value for the table elements.

Babatunde Kazeem
The description of the faults and associated failures is summarized in Table 3.4.

### 3.4.3 Faults in the Ntree

The *ntree* program reads command from a file and processes them in order to test a few functions. The functions simulate the implementation of a tree whereby each node can have unlimited number of child nodes. Each node holds two strings - key and content. The faults in the program are described in more details in the following two subsections.

#### 3.4.3.1 Version 1

In this section, the faults in the version 1 of the ntree program are looked at more closely.

- **Fault 1 (Cosmetic, omission):** For fault 1, there was a missing message on line 135 (Listing 3.27). A command ‘sibs’ of the program requires two arguments and checks if the arguments are siblings. The missing error message was supposed to display if the first argument is missing. Consequently, no message is displayed if the first argument of command ‘sibs’ is missing.

```
[....]
133 node1 = find_node(tree->root, key1);
134 if (node1 == NULL) {
135  rc = -1;
136 } else {
137  node2 = find_node(tree->root, key2);
[....]
```

Listing 3.27: Fault 1 in *ntree* version 1
• **Fault 2(Cosmetic, commission):** In version 1, the word “contents” was spelled wrongly on line 114 (Listing 3.28). Therefore, a message with a misspelled word is displayed if the user searched the program for an existing node.

```c
112 node = find_node(tree->root, key);
113 if (node != NULL) {
114    printf("The contents are: %s\n", node->data);
115    rc = 0;
116 }

Listing 3.28: Fault 2 in ntree version 1
```

• **Fault 3(Initialization, omission):** The variable ‘parent’ which represents the parent node was not stored in the data structure on line 19 (Listing 3.29). Thus, if both nodes used with command ‘sibs’ exists, they are never recognized as siblings because the value of their parent field was not initialized.

```c
16 assert(node != NULL);
17  node->key = key;
18  node->data = data;
19  node->n_children = 0;
20  node->max_children = INITIAL_CAPACITY;

Listing 3.29: Fault 3 in ntree version 1
```

• **Fault 4 (Initialization, commission):** In a method that prints the tree, the variable ‘i’ on line 162 (Listing 3.30) was initialized to 1 instead of 0. The leftmost node of the tree will never be printed because of this error in initialization.

```c
152 static void print_tree_nodes(TN *node,
153                             int level)
154 {
155  int i;
156
157  assert(node != (TN *)0);
158  for (i=0; i<level; i+=2)
159    printf("   ");
160  printf("Node (level %d): key '%s', Contents '%s'\n",

Listing 3.30: Fault 4 in ntree version 1
```
161      level, node->key, node->data);
162  for (i = 1; i < node->n_children; ++i) {
163    print_tree_nodes(node->children[i], level + 1);
164  }
165 }

Listing 3.30: Fault 4 in ntrees version 1

- Fault 5 (Control, commission): On line 138 (Listing 3.31), the expression should read (node2 == NULL). Therefore, if the second node belongs to the tree, the program displays an error message that it does not. On the other hand, if it does not, no message is reported.

[........]
134 if (node1 == NULL) {
135  rc = -1;
136 } else {
137  node2 = find_node(tree->root, key2);
138  if (node2 != NULL) {
139    printf("Node with key %s is not found\n", key2);
140    rc = -1;
141  }
[........]

Listing 3.31: Fault 5 in ntrees version 1

- Fault 6 (Control, omission): The code for checking the return value during a search operation is not complete. An error message is missing on line 116 (Listing 3.32). No error message is printed if the searched node cannot be found.

[........]
113 if (node != NULL) {
114  printf("The cotents are: %s\n", node->data);
115  rc = 0;
116 }
118  return rc;
[........]

Listing 3.32: Fault 6 in ntrees version 1

- Fault 7 (Computation, commission): The variable “i” on line 158 (Listing 3.33) should be incremented by 1 instead of 2. Thus, each level of the tree is indented wrongly.
Listing 3.33: Fault 7 in \textit{ntree} version 1

These faults are summarized in Table 3.5.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault name</th>
<th>Fault position</th>
<th>Fault description</th>
<th>Expected failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Omission, cosmetic</td>
<td>Line 135</td>
<td>The error message about not finding the nodes is missing</td>
<td>No error message is printed if the first argument of command “sibs” is not found while checking for siblings. An error message was expected</td>
</tr>
<tr>
<td>2</td>
<td>Commission, cosmetic</td>
<td>Line 114</td>
<td>The word “contents” is incorrectly typed</td>
<td>Misspelled word in the output</td>
</tr>
<tr>
<td>3</td>
<td>Omission, initialization</td>
<td>Line 19</td>
<td>Parameter “parent” is not stored in the structure</td>
<td>If both nodes used in the “sibs” command exist, then they are never recognized as siblings, because the value of the parent field is never initialized</td>
</tr>
<tr>
<td>4</td>
<td>Commission, initialization</td>
<td>Line 162</td>
<td>The variable “I” should be initialized to 0 instead of 1</td>
<td>The leftmost node is never printed</td>
</tr>
<tr>
<td>5</td>
<td>Commission, control</td>
<td>Line 138</td>
<td>The expression (node2 != NULL) should be . (node2 == NULL)</td>
<td>If the second node belongs to the tree, prints an error saying that it does not, and if does not belong to the tree, no error message is printed</td>
</tr>
<tr>
<td>6</td>
<td>Omission, control</td>
<td>Line 116</td>
<td>Code for checking the return values is incomplete</td>
<td>No error message is printed if the node cannot be found</td>
</tr>
<tr>
<td>7</td>
<td>Omission, computation</td>
<td>Line 158</td>
<td>Variable “i” should be incremented one unit</td>
<td>Each level in the tree is not properly indented.</td>
</tr>
</tbody>
</table>

Table 3.5: Description of faults in \textit{ntree} version 2

3.4.3.2 Version 2

In version 2, the faults are characterized as highlighted below:

- **Fault 1 (Cosmetic, omission):** There was a missing message on line 135 (Listing 3.34). A command ‘sibs’ of the program requires two arguments and checks if the arguments are siblings. The missing error message was supposed to display if the second argument is missing.
  Therefore, no error message is displayed if the second ‘sib’ argument is missing.
Babatunde Kazeem

Listing 3.34: Fault 1 in ntree version 2

- **Fault 2 (Cosmetic, commission):** On line 82 in (Listing 3.35), the word “is” was typed wrongly. If the user tries to add a child a non-existing parent in the table, the message on line 82 (Listing 3.35) is displayed with a wrongly spelled “is”.

Listing 3.35: Fault 2 in ntree version 2

- **Fault 3 (Initialization, omission):** Variable ‘sibling’ was not initialized on line 127 (Listing 3.36). Therefore, a check on two sibling nodes will result in a message saying they are not siblings.

Listing 3.36: Fault 3 in ntree version 2

- **Fault 4 (Initialization, commission):** On line 58 (Listing 3.37), the variable ‘i’ was supposed to be initialized to 0 instead of 1. This mistake has effect on the “child”, “search” and “sibs” commands.
As for the “child” command, the nodes in the sub-tree of the parent’s leftmost child are not found during a search operation. Also, no new node is inserted into the tree, instead an error message that the parent cannot be found is printed.

In case of the “search” command, a node which is supposed to exist is not found and no message/error message is printed.

Two sibling nodes are reported as none siblings when using the “sibs” command.

- **Fault 5 (Control, commission):** On line 113 (Listing 3.38), the expression should read (node != NULL). In this case, the contents of a node reported not found in the tree are printed.

```
[......]
103 int t_search(TREE *tree, char *key)
105 {
106    int rc;
107    TN *node;
108    rc = -1;
109    assert(tree != NULL);
110    assert(key != NULL);
111    node = find_node(tree->root, key);
112    if (node == NULL) {
113        printf("The contents are: %s\n", node->data);
114        rc = 0;
115    } else printf("Node with key %s is not found\n", key);
116    return rc;
117 }
[......]
Listing 3.38: Fault 5 in ntree version 2
```
Fault 6 (Control, omission): The loop for indenting the tree during a print operation is missing on line 158 (Listing 3.39). The tree structure is wrongly indented.

```

[......]
153 static void print_tree_nodes(TN *node, int level)
154 {
155      int i;
156
157     assert(node != (TN *)0);
158     printf("    ");
159     printf("Node (level %d): key '", node->key, node->data);
160     printf(" Content '%s'
", level, node->key, node->data);
161     for (i = 0; i < node->n_children; ++i) {
162         print_tree_nodes(node->children[i], level);
163     }
[......]
```

Listing 3.39: Fault 6 in ntree version 2

Fault 7 (Computation, commission): On line 162 (Listing 3.39), the variable method print_tree_nodes is called with “level” instead of “level +1”. Therefore, the tree is wrongly indented.

The faults are summarized in Table 3.6.
Table 3.6: Description of faults in ntree version 2

<table>
<thead>
<tr>
<th>Fault</th>
<th>Fault name</th>
<th>Fault position</th>
<th>Fault description</th>
<th>Expected failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Omission, cosmetic</td>
<td>Line 140</td>
<td>The error message about not finding the nodes is missing</td>
<td>No error message is printed if the second argument of command “sibs” is not found while checking for siblings. An error message was expected</td>
</tr>
<tr>
<td>2</td>
<td>Commission, cosmetic</td>
<td>Line 82</td>
<td>The word “is” is incorrectly typed</td>
<td>Misspelled word in the output</td>
</tr>
<tr>
<td>3</td>
<td>Omission, initialization</td>
<td>Line 127</td>
<td>Variable “siblings” is not initialized</td>
<td>When both nodes belong to the tree, a message is printed saying that they are not siblings</td>
</tr>
</tbody>
</table>
| 4     | Commission, initialization| Line 58     | Variable “i” should be initialized to 0                                           | Nodes in the sub-tree of the father’s leftmost child are not found during the search. The leftmost sub-tree of the root is not searched, and so forth. Causes problems for commands “child”, “search” and “sibs” as follows:  
  - child: No new node is inserted into the tree. An unexpected error message is printed that the father node could not be found.  
  - search: The node is not found and no message is printed (not even an error message).  
  - sibs: A message is printed stating that they are not siblings |
| 5     | Commission, control     | Line 113       | The expression (node == NULL) should be (node != NULL)                           | The contents of the node are printed when it is not found in the tree                                                                            |
| 6     | Omission, control       | Line 158       | The loop for indenting the tree is missing                                        | The tree is not indented properly                                                                                                               |
| 7     | Omission, computation   | Line 163       | The function is called with level instead of level+1                             | The tree is not indented properly                                                                                                               |
Chapter 4

Exercising with the Techniques

At the beginning of this study, the experiment was re-run by the researcher, by applying the three techniques on the two versions of the programs - cmdline, ntree and nametbl, as was done during the experiments. The experience of the re-run process is the theme of this chapter.

In order to determine the effectiveness of the evaluation techniques, the relative effectiveness of the subjects is also important. This was part of the study focus of Experiment II and was attested as part of the analysis of the discoveries during the personal running of the experiments in the course of this study. It was discovered that testers may fail to develop test cases because of their own negligence or little knowledge of the technique application. Also, there are situations when the tester will miss an exposed fault because of non attentiveness or he does not understand the specification enough to know it was a fault. These situations as encountered in the running of the experiment are presented in the following sub sections.

4.1 Test Case Generation

The application of the dynamic techniques presents three basic different situations (Figure 4.1).

- **Failure detected**: This happens when the technique is correctly applied, a test case is generated and the tester paid enough attention while executing the test cases to detect the failures.

- **Failure missed**: This is when everything happens as it does in ‘failure detected’ but the tester did not pay enough attention to observe, record and/or uncover the failure.

- **No test case generated**: This situation occurs as a result of improper application of the technique. Therefore, the tester failed to develop any test case capable of exposing a particular fault.

The test case generated and no test case generated situations (second level nodes in Figure 4.1) as it applies to each of the dynamic techniques are the theme of the following two subsections.
The third level nodes of will be further discussed in Section 4.2

![Diagram showing the process of test case generation and failure detection.]

**Figure 4.1:** Situations encountered with application of dynamic techniques

### 4.1.1 Test Cases using Equivalence Partitioning

In Table 4.1, the number of test cases generated for each version of the programs using equivalent class partition is presented. A column marked with ‘Yes’ indicates that a test case was generated for that particular fault of the program version, while a ‘No’ indicated otherwise. With test cases, we mean test cases that are actually capable of exposing each of the seven faults in the program.

<table>
<thead>
<tr>
<th>Program/fault</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Nametbl</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ntree</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Version 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Nametbl</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ntree</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.1: Test case generated using equivalence class partition
A test case developed that is not capable of this, does not count.

In all, 6 test cases were generated for the *cmdline* version 1, 7 for both *nametbl* and *ntree*, while it was 7 for *cmdline* version 2, 6 each for both *nametbl* and *ntree*.

### 4.1.2 Test Cases using Decision Coverage

Using decision coverage, the same information as above is presented in Table 4.2. In version 1, 6 test cases each was developed for *cmdline* and *nametbl*, and 7 for *ntree*. Version 2 has complete test cases developed for all the programs.

<table>
<thead>
<tr>
<th>Program/fault</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Cmdline</em></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><em>Nametbl</em></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><em>Ntree</em></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Version 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Cmdline</em></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><em>Nametbl</em></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><em>Ntree</em></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.2: Test case generated using decision coverage

### 4.2 Failure/Fault Observation

As stated earlier, when the test cases developed as shown in Table 4.1 and Table 4.2 are executed, the tester was expected to note the failure and record it. But sometimes s/he completely fails to notice such failures. Also, all faults were supposed to be detected by using the code abstraction technique. The situations when the tester reported a failure/fault or missed a failure/fault during the simulation experiment are presented in the following subsections.

#### 4.2.1 Using Equivalence Partitioning

While working with the equivalent partition technique, Table 4.3 presents the failures observed and those that were missed. Each of the columns is the result of a logical ‘AND’ between test case generation and failure detection. For example, if for a fault F1, a test case was generated and the corresponding failure detected, the column is marked ‘Yes’; but if the failure was not detected, the column is marked ‘No’.
A column is marked ‘N/A’ in a case where no test case was developed for the fault, definitely, no failure should be detected and if the failure was detected, it was not taken into account in this table.

One fault was missed with `cmdline` version 1 and none with version 2. Two faults were missed with `nametbl` version 1 and one with version 2. One fault each was missed with `ntree` version 1 and 2.

<table>
<thead>
<tr>
<th>Program/fault</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Nametbl</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ntree</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Version 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Nametbl</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ntree</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.3: Failure observation using equivalence class partition

### 4.2.2 Using Decision Coverage

The result of failure observation using decision coverage is presented in Table 4.4. The columns are the same as explained in section 4.2.1.

<table>
<thead>
<tr>
<th>Program/fault</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Nametbl</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ntree</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Version 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Nametbl</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ntree</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.4: Failure observation using decision coverage

In `cmdline` version 1, two faults were missed and none was missed in version 2. One fault was missed in `nametbl` version 1 and two in version 2. While one fault each was missed in `ntree` version 1 and version 2.

If this result is typical of the average performance of subjects in our experiment, it shows that the subject has an average 1 out of 6 (approximately 20%) chance of affecting the effectiveness of the evaluation techniques.
4.2.3 Using Code Abstraction

While using the code abstraction technique, the tester still failed to notice some of the faults in the code. The result of the abstraction exercise is presented in Table 4.5. In the table, a column with ‘Yes’ indicates the fault was detected and a ‘No’ indicates otherwise.

<table>
<thead>
<tr>
<th>Program/fault</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Nametbl</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Ntree</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Version 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Nametbl</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ntree</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4.5: Fault observation using code abstraction

4.3 Influence of Learning Effect

It was discovered during the exercise that some faults were likely to have been detected due to learning effect. Learning effect in this case is a situation whereby the tester became cautious to observe the presence some faults or fault types due to experience from previous similar situation. Most especially, in this case where fault types are replicated in all the programs. The likely result of learning effect during the exercise is presented in Table 4.6.

The columns marked ‘Yes’ are the only faults the tester might have had to learn from past mistakes.

<table>
<thead>
<tr>
<th>Program/fault</th>
<th>Technique</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cmdline</td>
<td>Equivalence partition</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Nametbl</td>
<td>Equivalence partition</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Decision coverage</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Code abstraction</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Version 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ntree</td>
<td>Equivalence partition</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Decision coverage</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Code abstraction</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Nametbl</td>
<td>Code abstraction</td>
<td>-</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.6: Faults found influenced by learning effect
Obviously F2 is the only fault greatly influenced by this effect. This is because it is a spelling mistake (cosmetic, commission) which was noticed from applying the review technique on \textit{cmdline}. The tester does not count it significant initially as a fault, until it was pointed out at the end of the exercise. From then on, error messages are always double checked for typographical mistakes.

4.4 Reasons for Missing a Fault or not Generating a Test Case

The probable reasons why a test case was not generated for any fault whose column is marked ‘No’ in Table 4.1 and Table 4.2, as well as the missed faults/failures in Tables 4.3, 4.4 and 4.5 are given in this section. Regarding the code abstraction, the reasons for missing a fault are two: lack of attention to detail (attention) and lack of comprehensive understanding of the program (Understanding). The same applies to the dynamic techniques as well: lack of attention to detail and mask effect or unknown reason (Unclear). However, improper application of the technique mostly account for why a test case was not generated.

4.4.1 Reasons for not Generating a Test Case

According to Table 4.1 and 4.2, no test case was generated for five faults. The reasons are basically two: improper application of the technique or reason unknown (Table 4.7).

4.4.1.1 Improper Technique Application

Under misapplication of the technique we have the following faults:

- \textbf{Equivalence partitioning:} Using the equivalent class partition four test cases were not developed due to improper application of the technique. The cases are:
  
  - \textbf{Fault 4} (\textit{cmdline}, version 1): The program input was wrongly partitioned; therefore this fault could not be exercised by any test case.
  
  - \textbf{Fault 2} (\textit{namebl}, version 2): The test cases developed does not cover all the partition for the input domain. The tester does not pay much attention to be aware of this before the start of test case execution.
  
  - \textbf{Fault 1} (\textit{ntree}, version 2): The same situation applies here as well; the test case developed does not cover the input domain.
- **Fault 6** (*cmdline*, version 1): The input space was inadequately covered due to wrong partition. Therefore, no test case was developed to expose this fault.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Program</th>
<th>Version</th>
<th>Technique</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4</td>
<td>Cmdline</td>
<td>1</td>
<td>Equivalence partition</td>
<td>Incorrect application of the technique. The four character ‘measures’ were grouped all in a class.</td>
</tr>
<tr>
<td>F3</td>
<td>Cmdline</td>
<td>1</td>
<td>Decision coverage</td>
<td>All possible options for the test cases were not exhausted. Also, there was a mistake in implementing the program’s command syntax.</td>
</tr>
<tr>
<td>F3</td>
<td>Nametbl</td>
<td>1</td>
<td>Decision coverage</td>
<td>It was not clear how to show the existence of this fault.</td>
</tr>
<tr>
<td>F2</td>
<td>Nametbl</td>
<td>2</td>
<td>Equivalence partition</td>
<td>The tester was not thorough enough. The test cases options were not exhausted.</td>
</tr>
<tr>
<td>F1</td>
<td>Ntree</td>
<td>2</td>
<td>Equivalence partition</td>
<td>Improper application of the technique. Test cases did not cover all situations.</td>
</tr>
<tr>
<td>F6</td>
<td>Cmdline</td>
<td>1</td>
<td>Equivalence partition</td>
<td>This fault was missed due to wrong application of the technique. The search options were grouped together.</td>
</tr>
</tbody>
</table>

Table 4.7: Reasons for not generating a test case

- **Decision coverage:** While using decision coverage technique we have:
  - **Fault 3** (*cmdline*, version 1): The input space was wrongly used; therefore, some input were left uncovered by the test cases developed.

4.4.1.2 Unclear Reason

For unknown reason, only one test case was not developed. This happens with the decision coverage technique. The test case was for:

- **Fault 3** (*nametbl*, version 1): In this case, the fault was exercised but the test case failed to reveal it. It was unclear how to really expose this fault. It is probably masked by another fault.

4.4.2 Reasons for Missed Faults

In total, 29 faults were missed out of 126 during the evaluation exercise (Table 4.3, 4.4 and 4.5). A fault is considered as missed because it was exposed during the evaluation exercise but the tester either failed to notice them, they were noticed but not considered as faults or the faults were not exposed by any of the test cases. The specific reason for each of the faults is presented below based on the technique used.
4.4.2.1 Using Code Abstraction

As mentioned above, the reasons why faults were missed while using this technique is classifiable into two:

- Attention to detail: A situation where the tester does not pay enough attention to the task at hand. For example, missing a spelling mistake in an error message is not attributable to anything but lack in attention.
- Comprehensive understanding of the program: Usually, to do a good job with code reading, the reader must have good knowledge of the programming language used as well as the specifications of the program.

4.4.2.1.1 Attention to Detail

Inattention to important details while reading the code lead to failure to notice the faults listed in Table 4.8. Most of the faults are spelling mistakes or swapped variables.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Program</th>
<th>Version</th>
<th>Specific reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>Cmdline</td>
<td>Version 1</td>
<td>No attention was paid to the error message wherein the spelling mistake was contained.</td>
</tr>
<tr>
<td>F6</td>
<td>Cmdline</td>
<td>Version 2</td>
<td>The data structure definition in the header file was not mapped to the implementation.</td>
</tr>
<tr>
<td>F7</td>
<td>Nametbl</td>
<td>Version 1</td>
<td>The required action was not specifically mentioned in the program specification.</td>
</tr>
<tr>
<td>F2</td>
<td>Ntree</td>
<td>Version 1</td>
<td>Spelling mistake not noticed.</td>
</tr>
<tr>
<td>F3</td>
<td>Ntree</td>
<td>Version 1</td>
<td>Tree initialization was overlooked</td>
</tr>
</tbody>
</table>

Table 4.8: Faults missed during abstraction

4.4.2.1.2 Program Understanding

The difficulty in understanding the program constructs, in order to comprehend the implementation lead to missing the faults in Table 4.9.

4.4.2.2 Using Equivalence Partitioning

As mentioned in Section 4.4, the reasons why failures were missed (not detected) during the application of dynamic techniques, even though, test cases were developed for them are classifiable into two:

- Attention to details: This happens when a tester failed to notice a failure that was exposed by any of the test cases.
• Mask effect: This is a situation where one fault covers up the manifestation of the other.

<table>
<thead>
<tr>
<th>Faults</th>
<th>Program</th>
<th>Version</th>
<th>Specific reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3</td>
<td>Cmdline</td>
<td>1</td>
<td>Failure to understand the implication of conditional statement’s predicate</td>
</tr>
<tr>
<td>F6</td>
<td>Cmdline</td>
<td>1</td>
<td>The missing error message was not detected</td>
</tr>
<tr>
<td>F3</td>
<td>Cmdline</td>
<td>2</td>
<td>Usage of the method was not completely confirmed.</td>
</tr>
<tr>
<td>F1</td>
<td>Nametbl</td>
<td>1</td>
<td>The need for the missing error message was not specifically mentioned in the specification.</td>
</tr>
<tr>
<td>F6</td>
<td>Nametbl</td>
<td>1</td>
<td>Incomplete understanding the behavior of program’s functions and inter relationship</td>
</tr>
<tr>
<td>F1</td>
<td>Nametbl</td>
<td>2</td>
<td>Insufficient background information from the specification</td>
</tr>
<tr>
<td>F6</td>
<td>Nametbl</td>
<td>2</td>
<td>Lack of understanding of what the function was doing due to intensive use of pointers</td>
</tr>
<tr>
<td>F6</td>
<td>Ntree</td>
<td>1</td>
<td>Unaware that a program part was missing since the function works just fine</td>
</tr>
<tr>
<td>F3</td>
<td>Ntree</td>
<td>2</td>
<td>Insufficient understanding of the code. wrong initialization was not suspected</td>
</tr>
<tr>
<td>F6</td>
<td>Ntree</td>
<td>2</td>
<td>Incomplete understanding of the code’s implementation</td>
</tr>
</tbody>
</table>

Table 4.9: Faults missed due to lack of program understanding using abstraction

4.4.2.2.1 Attention to Detail

Due to insufficient attention to details during the exercise, the faults (table 4.10) were not detected while using the equivalent partition technique.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Program</th>
<th>Version</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>Ntree</td>
<td>1</td>
<td>The fault showed up but the tester failed to notice it. The misspelling escaped the tester’s attention.</td>
</tr>
<tr>
<td>F5</td>
<td>Ntree</td>
<td>2</td>
<td>This was not regarded as a specific fault, it was considered as a ripple effect of another fault.</td>
</tr>
</tbody>
</table>

Table 4.10: Undetected faults due to inattention using equivalent partition

4.4.2.2.2 Mask Effect

The faults in this category as discussed earlier were actually tested but not uncovered. This is probably due to the fact that another fault in its line of execution is actually covering it up (Table 4.11).

<table>
<thead>
<tr>
<th>Fault</th>
<th>Program</th>
<th>Version</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>F3</td>
<td>Nametbl</td>
<td>1</td>
<td>It did not show up during testing.</td>
</tr>
<tr>
<td>F3</td>
<td>Nametbl</td>
<td>2</td>
<td>The fault did not show.</td>
</tr>
<tr>
<td>F6</td>
<td>Nametbl</td>
<td>1</td>
<td>It did not manifest during testing. It was probably masked by another fault on the variable concerned.</td>
</tr>
</tbody>
</table>

Table 4.11: Undetected faults due to possible masking using equivalent partition
4.4.2.3 Using Decision Coverage

The category of reasons under this technique is the same as it is for equivalent partition.

4.4.2.3.1 Attention to Detail

The faults (Table 4.12) below were not observed during test case execution due to inattention.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Program</th>
<th>Version</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>Cmdline</td>
<td>1</td>
<td>This fault was not noticed. The tester was not paying attention to spellings of output messages. He was rather focused program mal-functions.</td>
</tr>
<tr>
<td>F2</td>
<td>Ntree</td>
<td>1</td>
<td>The tester failed to notice the misspelling. He was focused seeing a failure.</td>
</tr>
<tr>
<td>F5</td>
<td>Ntree</td>
<td>2</td>
<td>This was not regarded as a specific fault, it was considered as a ripple effect of another fault.</td>
</tr>
</tbody>
</table>

Table 4.12: Undetected faults due to inattention using decision coverage

4.4.2.3.2 Mask Effect

The following faults (Table 4.13) are likely not exposed due to mask effect.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Program</th>
<th>Version</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>F6</td>
<td>Cmdline</td>
<td>1</td>
<td>The fault did not show up. Probably it is masked by other fault.</td>
</tr>
<tr>
<td>F6</td>
<td>Nametbl</td>
<td>1</td>
<td>The fault did not show up. It is probably masked by other fault on same variable.</td>
</tr>
<tr>
<td>F3</td>
<td>Nametbl</td>
<td>2</td>
<td>The fault did not show.</td>
</tr>
<tr>
<td>F6</td>
<td>Nametbl</td>
<td>2</td>
<td>The fault failed to show up.</td>
</tr>
</tbody>
</table>

Table 4.13: Undetected faults due to possible masking using equivalent partition

4.5 General Comments on the Exercise

The simulation was performed in a more relaxed environment than a normal laboratory. Though, efforts were made to be as disciplined as possible in terms of timing and working on the exercises. Like a normal experiment, the solution to the problem was not known until after the exercise, it was mentioned wherever the prior knowledge of working with a program or exposure to the solutions might have influenced the outcome of any exercise.

The following comments are subjective remarks abstracted based on the experience while applying the techniques.
4.5.1 Comment on Code Reading

The general discussion on code reading will be based on four subjects: technical experience and/or knowledge, functional structure and behavior of the program and personal motivation.

4.5.1.1 Technical Knowledge or Experience

The technical knowledge is divisible into two: expertise/knowledge of the technique and the programming language.

- **Programming language knowledge**: The knowledge or experience of the tester, in the programming language used, in the program development, plays a vital role in the understanding of the code. Since in code reading, the tester has to read every piece of line and try to follow the logic and do mental simulation of the program, if he is less familiar with the programming language or the language constructs used, he will not be able to make much sense out of the program. He can only assume or guess what the program is doing and where it is doing it. Therefore, he may miss out on vital faults, since his understanding is partial, most especially those fault that are as a result of method or procedure interaction.

- **Technique application knowledge**: Experience with the technique can also play vital role in what fault is detected. An experienced code reader will have more enthusiasm reading a code, also he will have some strategy to determine likely portion that are prone to error e.g. variable initialization, loop control variables and conditions, parameter passing, type mismatch etc. this will aid his detection capacity from the abstraction. He will be adapted to mental simulation of codes, an exercise that new readers may find difficult if he lose track of events often.

4.5.1.2 Program Structure and Behavior

The program structure, length, complexity and time are factors that might also influence outcome in code reading. The lengthier or more complex the code, the harder it will be to follow. This may favor fault positioning. That is, faults positioned in inner depth of a complex or lengthy program may go undetected because the reader might have lost track of what is actually happening (the pre-condition and the post-condition) at that point or he might just be weary.
Connecting related parts of codes may be tiresome and this may be needed to detect any logical or operational fault. This has to do with the extent of coupling in the code being read. For example, if a part of the code that implements a method1 and method1 also uses a method 2 is being read. The reader has to trace all this dependencies before he can predict the output of that part of the program. Along the line, a reader may lose track of what he is doing or become dissociated from the reading exercise.

4.5.1.3 Personal Motivation

The tester’s mind frame goes a long way to affect the success of the testing exercise. Whether distracted or under pressure; also included is emotional stability and psychological state of the mind. A tester under pressure (e.g. of what mark to score in the testing exercise or of performance evaluation) may get too nervous that he cannot do a good job.

Code reading was found to be generally boring and uninteresting; this has effect on the tester’s enthusiasm, focus of attention to minute details, concentration and interest in the code to be read. For these reasons it was impossible to do a proper or thorough mental simulation of the program. Thus, faults as simple and glaring as spelling mistakes were missed out.

Also, early success (fault detection) in error prone areas is a good stimulant to the enthusiasm and concentration of a new reader, a factor that may not necessarily affect an experienced reader. He will be motivated to continue reading with a stronger inner assurance that he will find more faults. A motivation he is not likely to have if he could not pin point a fault after reading a piece of code for say over 30 minutes, with several pages left to read.

4.5.2 Comments on Dynamic Techniques

The author found dynamic testing to be generally more interesting as it involve interaction with the program with a set of input. Based on the input the program gives some output which may aid intuitive visualization of what is happening within the program. The interest in the technique is a motivation for concentration which may lead to good performance.

More so, dynamic techniques require less detailed knowledge of the software piece to be tested. Rather, it needs the mastery of the technique.
Knowledge of the programming language is not relevant to the effectiveness of the technique. This is because the test cases will be developed and implemented without the physical code but the object code. This helps limit the factors associated with program complexity and structural understanding that might make the tester lose interest, most especially if he does not understand a lot of things in the code. This is an aid factor to successful fault finding since practically, anybody can use the technique and detect faults.

Dynamic testing probably operates at a better abstraction level of the program; the tester does not have to worry about the length or complexity of the program. This is a good stimulant to work with, it is possible to detect a couple of faults with a single test case and the faults may span methods and several lines of codes. This situation is impossible in code reading - where the tester has to read through each line and methods and also establish the relationship between the methods to reason out any fault.

4.5.2.1 Comments on Equivalence Partitioning

A good knowledge of the technique is required to make a good application/implementation of the technique. That is, to determine partition classes for each program input. The knowledge or level of experience will determine the quality of partitions and how successful or useful the equivalence classes developed by the tester would be to reveal faults. For example, this will aid in how he will decide to partition the input classes like the “<measure>” option in cmdline, should they be categorized as one or split as individual class of input.

The understanding of the program input, how each input is used and handled by the program is also vital, in order to determine equivalence class limits or partition boundaries as an aid to the point mentioned above.

As revealed in some test cases, it is sometimes difficult to categorize the inputs; this is because sometimes, the inputs do not fall in a range of value, but are a set of independent nominal variables probably treated in the same way by the program. The technique allows grouping inputs that are treated in a similar manner in the same partition. However, nominal input variables, if grouped together may mask faults in one another. So it is a mistake or wrong use of the technique to group nominal input variables together. This goes back to knowledge, understanding and experience with the technique implementation, as well as understanding of the input usage. For
example, this mistake occurs when the “<MEASURE>” options in `cmdline` were grouped into two class partitions – the one with minimum of two characters and the one with minimum of three characters. This is done with the understanding that the program treated all the `measure` options in the same way, and that the only difference is the minimum number of characters that can represent each. Also, that none of the measures perform an operation different from the other. As it turned out, it became apparent that this was a wrong assumption as this allows for fault masking.

Focusing on specification as a driver to the testing exercise rather than be carried away by execution is important. For example, in one of the testing exercise, where the order of input does not matter, the general input form was adhered to throughout the testing exercise, forgetting to even change the order of the input and see what will happen. The order was not changed until reading the specification again and rerun a test case that had been run earlier without detecting a fault. By changing the input order, this test case behaved in a different manner revealing misbehavior by the program.

### 4.5.2.2 Comments on Decision Coverage

This technique is interesting but may become tedious if the program is large and contains several conditional points and inter-related structures – coupling and complexity. This is because the generation of any test case involved reading through the code to note all the decision points and also to note all the inter-dependency and inter-relationship between different decision points within the program as a whole. So, the higher the coupling and complexity the more weary a tester might become and the less effective he will be at understanding the program logic which will in turn negatively affect the development of useful test cases.

The knowledge of the programming language may be needed to do a good job with the technique though not necessarily. If a tester is versed in the programming language used, this is good as he will understand the codes easily and could make informed decisions as to what inputs can negate or validate each condition or even negate/validate multiple condition points with a single input. However, the minimum condition required for this technique is the ability to be able to put values that is capable of negating and validating each decision point which may be achieved by just interpreting a condition like: `if(measure == NULL)`, this is interpretable as when `measure` is empty or zero. This can be interpreted even without knowing much of the program implementation.
This technique seems exhaustive in nature, because one has to read through the code and find decision points and also determine the relationship between decision points in different parts of the program. A complex program may break the tester’s interest and concentration. Therefore, he might be less effective at developing the correct test cases capable of revealing faults.
Chapter 5

Common Structure

There has been a common view for the description of the experiments but not for the results. Therefore, a major motivation for undertaking this research is to have a joint and common view for the data and result files of the experiments in the series. This will afford an easy joint interpretation of the experiment results. The data from the series of experiments used in this study were extracted from several response forms filled by the subjects, into excel file. The excel file data are further transferred into SPSS data files based on the focus of the analysis required – effectiveness, efficiency etc. This thus results in several SPSS output files per experiment. Usually, each experiment consists of several different file types:

- **Experiment description**: These are document files describing the experiment – factors, hypothesis, design, etc. – in details.
- **Training Materials**: These are experiment material like the programs to be used for the experiments, solutions to the program or slides explaining some concepts of the techniques.
- **Experiment Material**: These are materials used during the experiments e.g, worksheets, instructions, source codes etc.
- **Excel File**: This is the data file that contains extracts of subject’s input into the experiment worksheets.
- **SPSS files**: Two types of SPSS files are kept – data files and output files. The data files contains relevant extracts from the excel file. The output files contain the result of analysis of the data file. For each experiment, there are usually many data and output files, each focused on specific analysis.

Until now, the organization of these files was chaotic, so, the first step taken in this work was to create a structured directory to organize and store the experiments’ files - excel file, SPSS data file, SPSS output files, general experiment description, training materials and experiment materials. The details of organization of experiment description files and training materials are not covered in this
work. Also, the structure and language of the excel files differ based on the prerogative or style of the researcher that handled each experiment. For the purpose of this and subsequent studies that will use the experiments’ data and analysis results, the excel files are re-organized to conform as much as possible to the same structure and language in this research.

5.1 Directory Structure

It is important to store the files of the experiments – data, analysis, training as well as other supporting materials- in a good well structured location for easy access and referencing in case of future use. Therefore, a computer directory was created for the files.

The root directory is called Experiment II. As mentioned, the series of experiments analyzed in this study belongs to Experiment II series. Experiment II is a series of experiments performed with the same goal – hypothesis and design. So, each of the subdirectories contain further directories of information on the series of experiments that was conducted with similar focus.

For example, in the concrete case of Experiment II’s directory, it contains subdirectories named Experiment files. This folder then contains several folders named after each of the experiments in the series with the nomenclature: place of experiment + year. Consequently, Experiment file’s directory contains eight other directories named: UPM 2001, UPM 2002 up to 2005, UPV 2005, ORT 2005 and UdS 2005.

The directory of each of the specific experiments, for example, UPM 2001, contains two other directories named: Experiment description and Collected Data. The Experiment description folder contains a file named experiment description and a folder named Materials. The Materials folder contained two folders – one named Training containing materials like the program, slides, solutions to programs etc. and the other named Experiment containing worksheets, instructions, source code etc.

The second top level directory named Data Collected contained an excel file a folder called Data analysis. The Data Analysis folder consists of two other folders named SPSS data files and SPSS result files. The Data Files folder houses the SPSS data files while the Results files folder houses the analysis output files from SPSS. This illustration is depicted in Figure 5.1.
Figure 5.1: Structure of the directory for the experiments

5.2 The Original Excel Files

There are eight experiments in the series being studied. Therefore, there are eight excel files to work with. As earlier mentioned, the internal structure of each file as well as the structure of the worksheets is different (Table 5.1). The column naming convention is different for each file - similar data are stored under different headings. The most challenging in this task – creating uniformity - is maintaining a correspondence between the intended meanings of each data across files to preserve data integrity. The experiments are named by combining the abbreviation of the place where the experiment took place e.g. UPM for Universidad Politécnica de Madrid and the last two digits of the year the experiment was conducted e.g. 01 for 2001.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Number of worksheets</th>
<th>Language used</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPM 01</td>
<td>3</td>
<td>English</td>
</tr>
<tr>
<td>UPM 02</td>
<td>3</td>
<td>English</td>
</tr>
<tr>
<td>UPM 03</td>
<td>3</td>
<td>English</td>
</tr>
<tr>
<td>UPM 04</td>
<td>3</td>
<td>English</td>
</tr>
<tr>
<td>UPM 05</td>
<td>4</td>
<td>Spanish</td>
</tr>
<tr>
<td>ORT 05</td>
<td>1</td>
<td>Spanish</td>
</tr>
<tr>
<td>UdS 05</td>
<td>2</td>
<td>Spanish</td>
</tr>
<tr>
<td>UPV 05</td>
<td>4</td>
<td>Spanish</td>
</tr>
</tbody>
</table>

Table 5.1: File structure for each experiment
Basically, each of the files contains performance information of each subject during the experiment. The information includes: subject name, time spent to generate test cases, time spent running the test cases, number of test case generated, number of faults/failure found amongst others.

5.3 Procedure used to Create the Structure of the New Excel Files

In order to give the files a common structure, in terms of language, worksheet naming, data field naming and the intended meaning of each of these, the following actions were taken on the existing (old) excel files:

- Create a glossary for all the terms
  - Collate separately, all English and Spanish terms used for field naming in the excel files
  - Translate all Spanish words to English equivalent
- Determine the sort of data are relevant to defining the dependent variables of the experiment
- Conduct a mapping of data collected, terms and meaning for each field to establish a basic set of fields to include in the new excel file. This step is more important for the files that were in Spanish.
- Create a master excel file from which the actual fields for the new file of each experiment is extracted

With these steps, we were able to determine the common structure for the excel files and corresponding worksheets. The new files are all in English with four worksheets – subject data, technique application, failure visibility and predicted failure visibility. The data field in each worksheet varies according to what the sheet defines.

5.3.1 Glossary of Terms

The first step taken towards creating a uniform file structure for the experiment files is to get the meaning of all the terms used in order to create meaningful equivalence. The details given in this section about the file location of each worksheet and the sheet location of each data field (original
excel files structure) are respectively summarized in Appendix A. The name abbreviations in sections below and in the appendix are as used in the files.

5.3.1.1 Terms used in the UPM 01 – UPM 04 Files

The experiment files for the UPM 01, 02, 03 and 04 files exist in a similar format – sheet structure and terms used. The files contain 16 unique terms used for column naming and three sheets namely – subject data, technique influence and failure visibility. The terms are used to define the sheets.

The following are the data field names (terms and their meanings) as used in the files:

- **Subject**: The name of the participants in the experiment.
- **Program**: The name of the program used by each subject.
- **Technique**: The technique applied by each subject on a particular program.
- **Version**: The version of the program in on which the technique is applied.
- **F$_i$ (1 ≤ i ≤ 7)**: The list of fault numbered F1 to F7, each column corresponding to faults 1 to fault 7. The columns are either marked ‘0’ or ‘1’. This term have a slightly different meaning based on the sheet where it is used. The different meanings will be expatiated in later discussions.
- **Rel. Ex.**: This is the relative experience of the subject in respect of application of the evaluation techniques. The grade is on a scale of 1 to 10 chosen by the subject.
- **Abs. Ex.**: This is the absolute measure of the subject’s experience regarding the application of the evaluation techniques. The subjects grade themselves from 1 to 10.
- **Motivation**: A subjective measure of the subject’s level of motivation.
- **Time 1**: The time spent in applying each technique measured in minutes. This means the time spent to develop the test cases.
- **Time 2**: The time spent to run the test cases or abstractions.
- **Time 3**: Time spent to detect either the expected failures and identify the faults.
- **N Abs/classes**: Number of abstraction or classes. This applies to the review and the functional techniques. Number of abstraction refers to the number of abstractions generated while applying the review technique. Classes refer to the number of equivalent classes generated for a particular program.
• **N cases**: This is an abbreviation for “number of cases”. It stands for the number of test cases developed in the dynamic techniques.

• **% defects**: This is the percentage ratio of the expected defects detected to total faults expected. Calculated as \( \left( \frac{\text{faults detected}}{\text{total faults}} \right) \times 100 \).

• **Total**: The sum of the faults (F1 – F7) columns.

• **Confidence**: A subjective grading of how confident the subject is on the solutions provided to the tasks given. The subjects grade themselves on a scale of 1 to 5.

As mentioned earlier, the files have three sheets each. The following list highlights each sheet and explained the contents.

• **Subject data**: This sheet contains comprehensive general performance information of all the subjects during the experiment. The data recorded in the subject data sheet are: subject, program, technique, version, rel. ex, abs. ex, time 1, time 2, time 3, motivation, N Abs/classes, N cases, % defects and confidence. These terms are as defined above.

• **Technique influence**: The experiment data indicating whether a subject is able to generate test cases for the faults seeded into the program are recorded in this subject. The technique application sheet contains information on how well the subjects performed regarding test case generation. It is clear from this sheet how many test cases targeted towards each of the faults were generated by each subject. The sheet consists of: subject, program, technique version, F1 – F7 and total data fields. Regarding the F1 to f7 fields, a column marked ‘1’ in this sheet indicates that a test case was generated for that particular fault while a ‘0’ indicates otherwise.

• **Failure visibility**: This sheet is a record of data that indicates whether the test cases recorded in the technique application sheet actually reveal the corresponding fault or not. The failure visibility sheet measures the performance of the test cases developed as recorded in the technique application sheet. The sheet contains and means the same columns as the technique application except for the F1 – F7 fields. The fields signify whether a failure was actually detected during the execution of the test cases or not. Therefore, a column marked ‘1’ means that the fault was detected while ‘0’ indicates otherwise.
5.3.1.2 Terms used in UPM 05 File

This file was prepared in Spanish (Table 5.1). It contains 4 sheets and 21 distinct column names. The following are the data fields as found in the UPM 05 data file:

- **Alumno**: The name of the subject.
- **Clase**: An indication whether the subject attended the training session for the Experiments. There were two training sessions.
- **Vez**: A field indicating how many of the two sessions the subject attended.
- **Programa**: The name of the program used during the experiment by the subject.
- **Tecnica**: The technique out of the three techniques applied by the subject.
- **Version**: The version of the program used by each subject during the experiment.
- **Ex. Rel.**: A short form for “experiencia relativa”. This field indicates the relative experience of the subject regarding evaluation techniques. It is subjectively graded on a scale of 1 – 10 by the subject.
- **Ex. Abs.**: Short form for “experiencia absoluta”. The field records the absolute experience of the subject in relation to evaluation techniques. It is also subjectively graded on a 1 – 10 scale.
- **Tiempo 1**: The time it takes to apply the techniques. That is, develop the test cases in dynamic techniques and create abstraction in the code review.
- **Tiempo 2**: The time it takes to run the test cases in the case of dynamic techniques or use the abstraction to identify faults in code review.
- **Tiempo 3**: The time it takes to detect the failure and identify fault applicable only to the dynamic techniques.
- **No. Abs/classes**: Number of abstraction generated in the code review or equivalent classes in the case of the functional technique.
- **No cases**: Number of test cases generated for either of the dynamic techniques.
- **% fallos**: Percentage ratio of failure/fault observed to that of expected.
- **Confianza**: Subjective confidence level of the subject in regards to the solutions provided during the experiment.
- **Fi (1 ≤ i ≤ 7)**: These seven fields indicate the subject’s performance in the fault discovery process.
• **Total**: The sum of the corresponding columns of the F1 – F7 columns.

• **Porcentaje**: Percentage ratio of the ‘total’ field.

• **Maximo**: The maximum number of faults detectable barring the effect of fault masking.

• **Tiempo técnica**: Same as time for applying the technique (tiempo 1).

• **Tiempo fallos**: This is addition of tiempo 2 and tiempo 3.

• **% faltas técnica**: This column refers to the percentage of faults detected by cases generated with the technique.

• **Tiempo total**: Total time spent by each subject conducting the experiment.

As earlier mentioned, the UPM 05 file has four worksheets: datos sujetos, datos tecnicas todos and datos fallos. The sheets are highlighted below:

• **Datos sujetos**: This sheet contains the overall general performance record and time information of each subject. The fields in the sheets are: alumno, programa, tecnica, version, clase, vez, ex. Rel., ex. Abs., tiempo 1, tiempo 2, tiempo 3, No. abs/clases, No. cases, %fallos and confianza.

• **Datos tecnicas**: The datos tecnicas sheet contain information on the number of faults found by each subject using the code review technique and the number of test cases capable of exposing the seeded faults in the program developed by each subject. The fields in the sheet are: alumno, programa, tecnica, version, F 1 – F7, total, percentaje and maximo. The F1 – F7 in this sheet reports the number of test cases developed by each subject relevant to the faults in the programs.

• **Datos fallos**: This sheet records the performance of each subject regarding the dynamic techniques only. It shows information on how many failures were actually detected by the subjects using the test cases each developed. The sheet contains the same set of fields as the datos tecnicas sheet. However, F1 – F7 columns in this sheet reports the number of failures that were detected by the subjects.

• **Todos**: This sheet todos serves as a repository of information to other sheets. The records in other three sheets were extracted from this sheet. Therefore, the sheet contains all the data fields listed above except the fault/failure (Fi) fields.
5.3.1.3 Terms used in UPV 05 File

The UPV file was prepared in Spanish; it contains 4 sheets and contains 24 distinct data fields. The following data fields are used in the UPV file:

- **Grupo**: At UPV, subjects worked in groups therefore, the term grupo refers to the group identification number for each subject.
- **Proced.:** Subject’s background information
- **Tiempo**: This is the time it takes each subject to develop test cases in dynamic techniques or generate abstractions in the code review technique.
- **Tiempo ej.:** The time it takes each subject to run the test cases.
- **Tiempo id.:** The time taken to identify failure or fault by each subject.
- **Visible**: The sum of the F1 – F7 fields.
- **Proced.:** Subject’s background information
- **Visible**: During the experiment, it was discovered that some of the faults masks one another. This field indicated the maximum amount of the faults visible/detectable.
- **T_tecnica**: This is the time it takes to apply the technique. The field is equal to Tiempo.
- **T_fallos**: Time used to detect the failures
- **T_total**: This field is defined as: \((\text{tiempo 1} + \text{T_fallos} ÷ 23 \times \text{No. casos})\)
- **CaF_1 (1≤7)**: There are seven CaF fields. These fields indicate the test case number from each subject that detect the fault ‘i’. For example, if test case 5 detects fault F1, then the column CaF1 will be assigned number 5.
- **Porc_Tec**: The percentage of faults detected by test cases generated by the technique. It is defined by: \((\text{vistos/visible} \times 100)\).
- **Porc_Vistos**: Percentage of total failures observed.
- Other fields are: alumno, programa, grupo, tecnica, version, ex. Rel., ex. Abs., tiempo 1, tiempo 2, tiempo 3, No. abs/clases, No. cases, %fallos, F1 – F7 and confianza, which has the same meaning as defined in section 5.3.1.2.

The file contains four sheets: datos sujetos, datos tecnica, datos fallos and todos, as highlighted below:

- **Datos sujetos**: The datos sujetos worksheet contains general information about subjects, time and performance during the experiment. It contains data like: alumno, grupo, programa, tecnica, version, ex. Rel., ex. Abs., tiempo, tiempo ej., tiempo id., No.
abs/classes, No. casos, %fallos and confianza. It contains general and time information about the subject’s performance in the experiment.

- **Datos técnica**: This sheet contains the record of the subject’s performance concerning the application of the technique. The data fields in it are: alumno, grupo, programa, tecnia, version, F1 – F7, vistos, visible and porcentaje. The fault fields (F1 – F7) indicates the number of case studies developed by each subject targeted at each of the faults.

- **Datos fallos**: This sheet contains similar field as the datos técnica except that the F1 – F7 fields indicates the failures detected by the dynamic techniques.

- **Sheet Todos**: This sheet is a master sheet where the content of other sheets were extracted. It contains the following extra fields beside the ones mentioned above for other sheets: T_tecnica, T_fallos, T_total, CaF1 – CaF7, Porc_Tec and Porc_vistos.

### 5.3.1.4 Terms used in ORT 05 File

The ORT file contains only one sheet named “Todos” meaning “All”. This implies that unlike the UPV sheet, the data were not sorted into other focused worksheets after the initial data entry. It contains 21 distinct data fields which will be defined here.

The fields are named in Spanish like the UPV and UPM 05. Only data fields that are new words/phrase will be defined in this section. The ones with similar terms and meaning to other files (previously defined) will only be referenced. Below are the fields’ definitions:

- **Carrera**: The career or specialization of the subjects e.g., computer engineer or information system analyst

- **Tur2o**: The turn when the experiment was performed (Morning or evening).

- **Asignacion**: Internal experiment design subgroup. That is, groups within a group.

- **No. estudiante**: This is the student’s registration number, used as a form of unique identification.

- **Nombre**: The name of the subject.

- **Orden**: The order of applying the technique.

- **No clases/caminos**: This refers to the number of equivalent classes generated by the functional technique or the number of abstraction in the code review (caminos).

- **Aplicacion**: This term is used to define the subject’s confidence level concerning the solution he provided to the experimental tasks. It is a subjective grading on a scale of 1 – 5.
• **Calificacion**: This is the teacher’s grading of the subject’s performance, assigned at the end of the experiment. The maximum grade obtainable is 5.

• **Detectables**: This is the number of fault detectable during the experiment without the masking effect. This field was required because one of the faults masks the other in one of the programs. So, there was no way to exercise it.

• **Tiempo**: This is the sum of time spent to execute the test cases and identify the failures. No static technique was applied in this experiment.

• **Experiencia relativa**: This is the relative experience of the subject in the evaluation techniques.

• **Experiencia absoluta**: the absolute experience of the subject in the evaluation techniques.

Other fields are: grupo, tecnica, programa, version, No. casos, F1 – F7, total and percentaje. These meant practically the same thing as similar terms in previous sections.

**5.3.1.5 Terms used in the UdS 05 File**

The UdS 05 file also contained two sheets named “casos” and “ejecucion”. The file has 18 distinctly named fields.

This sheet has 18 fields some of which are similar in terms and meaning to those previously defined. These will not be redefined in this section, only the newly introduced term will be defined.

• **Pareja**: Subjects worked in pairs to form groups in this experiment. This term is used to identify the pair to which a subject belongs to.

• **Asignacion**: This term was used as the group identifier. It denoted the group to which each pair of subject belongs. The groups are made up of several subject pairs.

• **Proc.**: This term is used to reference the subject’s background. For example, participants in the experiment came from different faculty and departments.

• **ExpRel**: The relative experience of each subject regarding the evaluation techniques. This column is subjectively graded by the subject.

• **ExpAbs**: The absolute experience of the subject regarding the evaluation techniques. Grading is same as in “ExpRel”.

• **Clases**: The number of equivalent classes generated by each pairs of working pair. Subjects worked in two pairs.
• **Casos**: The number of test cases generated by each subject pairs (two pairs of four people working together).

• **Tgen**: Time spent to generate test cases.

• **AutoEval**: The confidence grading of each subject pairs regarding the solutions they provided during the experiment.

• **Tteclear**: The time it takes to input test parameters.

• **Tfallos**: The time spent in identifying fault/failure.

• **PorEst**: The percentage of faults, estimated to be detected by each subject.

Other field that has similar terms and meaning to the ones previously explained in previous sections are: programa, tecnica, version, alumno, F1 – F7 and nombre.

As mentioned earlier, the file contains two worksheets: “casos” and “Ejecucion”. The two are highlighted below:

• **Casos**: This sheet contains information about test case generation and time spent on this activity by each subject group (no static technique was applied in this experiment). The fields in the sheet are: pareja, asignacion, nombre1, nombre2, exprel1, expabs1, exprel2, expabs2, proc1, proc2, programa, tecnica, version, clases, Tgen, casos, autoeval and F1 to F7. The F1 – F7 columns in this sheet indicates whether a test case is developed for a particular fault or otherwise.

• **Ejecucion**: This sheet contains information (time and performance) on the execution of the test cases. The fields it contains are: programa, version, alumno1, alumno2, Tteclear, Tfallos, PorEst and the F1 – F7 fields. The F1 – F7 fields in this sheet indicates the performance of the test cases or the subject at revealing or detecting the failures.

### 5.3.2 Abstraction of Relevant Data

After collating and explicitly defining the terms used for naming fields in all the experiment files; a wide variety was noticed in how the researchers used different terms to mean the same thing, for example, ‘alumno’ and ‘nombre’ both refers to the name of the subject.
5.3.2.1 Basic Data Fields

Consequently, we took a critical step of defining the set of data fields are particularly relevant or are likely to have future relevancy to the analysis of the experiments. Independent of any of the files, we defined these basic data fields as a first step. The fields concluded on are:

- **Serial Number (No.):** This is an identification given to each subject that participated in the experiment. The number is maintained consistently across the sheets of the file. This number is controlled by the subject’s name, the program he worked with, the program’s version and the technique. Therefore, a subject that worked with `cmdline` version 1 using code review and also `nametbl` version 2 using functional technique will have different numbers for these two situations. This makes cross referencing between worksheet easy and consistent.

- **Subject Number:** This is another type of unique identification given to each subject. This is controlled by the subject’s name irrespective of whatever artifact he worked with.

- **Subject:** This field defines the name of the subject.

- **Program:** The name of the program the subject worked with during the experiment.

- **Technique:** The technique applied on a particular program.

- **Version:** The version of the program used with a particular technique.

- **Rel. Exp.:** The relative experience of the subject.

- **Abs. Exp:** The absolute experience of the subject.

- **Technique application time:** The time it takes each subject to apply the technique. By apply the technique we mean, develop the test cases or abstraction as the case may be.

- **Test case execution time:** The time it takes each subject to execute their test cases in the case of dynamic techniques.

- **Failure/fault detection time:** The time taken by each subject to detect the failures while executing the test cases (dynamic techniques) or identify the faults after the abstraction (static technique).

- **No. of abstraction:** The number of abstractions generated while applying the code review technique.

- **No. of classes:** Number of equivalent classes generated in the functional technique.
• **No. of test cases**: The number of test cases developed for either of the dynamic techniques.

• **% estimated defects**: The percentage estimation of defects detectable by the test cases.

• **Confidence**: The subjective confidence level grading by the subjects on the solutions provided during the experiments.

• **F1- F7**: The fields relating to the seven faults seeded in each program.

• **Total**: The sum of the F1 – F7 fields.

• **Visible**: The maximum number of fault detectable during testing considering the effect of fault masking.

• **Percentage**: The percentage ratio of the “total” field to the “visible” field.

• **Execution and Id time**: Addition of test execution time and failure/fault identification time. This field is used in any case where it is impossible to separate the two times.

### 5.3.2.2 Other Accommodated Data Fields

In order to accommodate all the files without losing vital experiment information, the above list of data was expanded to include:

• **Origin**: The origin of each subject noting their background.

• **Motivation**: The level of motivation of each subject, subjectively graded by the subjects.

• **Group**: In the case of experiments conducted on group of subjects, this field indicates the group identification for each subject or subject pairs.

• **Pair**: Pairing identification for the cases where subjects worked in pairs.

• **Attendance**: A field indicating whether the subject attended the training session for the experiments or not.

• **Qualification**: A researcher’s grading of the subject’s performance in the experiment.

• **Order**: The order of applying the technique.

• **Case F1 – Case F7**: These fields indicate the test case number from each subject that detect the fault ‘f’. For example, if test case 5 detects fault F1, then the column Case F1 will be assigned number 5.
Also, due to pairing, in some experiments, fields like subject, Rel. Exp., Abs. Exp. and origin were duplicated to allow for the pairs. For example, the subject field appeared as subject1 and subject2. In cases where subjects work individually, only one of these fields will be used.

The terms defined in this section are used to create the master excel file which will be discussed in section 5.2.4 below.

### 5.3.2.3 Worksheets Definition

In this new work, it was deemed necessary that the excel file for each of the experiments should consist of four worksheets. The sheets are:

- **Subject data**: A sheet containing the basic general information of each subject. It also contains time spent by each subject on each of the important tasks of the experiment.
- **Observable faults**: This sheet contains information on how each subject faired in the test case development or abstraction phase of the technique application as the case may be during the experiment.
- **Failure visibility**: This sheet contains the record of the actual failure detected by the corresponding test cases developed by each subject as recorded in the observable fault sheet.
- **Observed faults**: This sheet presents the F1 – F7 column of the code review row for each subject in the “observable faults” sheet and also the logical AND of the corresponding F1 – F7 fields in the “observable faults” and the “failure visibility” sheets for the dynamic techniques.

### 5.3.3 Mapping of Data

Now that the required data fields have been defined, the next step is to find the correlation between the terms previously used by the researchers either in English or Spanish. This relationship will be used to create the new files for each of the experiments without losing the original information. The key to this step is to compare definition of terms in the existing files to the new English terms and create the new files. Any field mentioned in Subsection 5.2.1 but not in this for each file means it is excluded from the newly created file.
5.3.3.1 Mapping the New Data Definition with the UPM 01 – UPM 04 Files

These four UPM files are structure identical and also existed in English language. Therefore, the same criteria apply to the four of them. The field names and definitions fit well with the new ones. The few ones that will be changed are presented in Table 5.2.

<table>
<thead>
<tr>
<th>Old field name</th>
<th>Corresponding new name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time1</td>
<td>Technique application time</td>
</tr>
<tr>
<td>Time2</td>
<td>Test case execution time</td>
</tr>
<tr>
<td>Time3</td>
<td>Failure/fault detection time</td>
</tr>
<tr>
<td>N abs/classes²</td>
<td>No. of Abstraction</td>
</tr>
<tr>
<td></td>
<td>No. of classes</td>
</tr>
<tr>
<td>N cases</td>
<td>No. of test cases</td>
</tr>
<tr>
<td>% defects</td>
<td>% estimated defects</td>
</tr>
</tbody>
</table>

Table 5.2: Old and new field names correspondence in the UPM 01 – UPM 04 files

The “visible” field is also included in the appropriate sheets.

5.3.3.2 Mapping the New Definition with the UPM 05 File

The new data field names that correspond to the terms used in the UPM 05 files is presented below (Table 5.3). Generally, the “F1 – F7” and “total” fields remained unaffected by the change in term. This also applies to other files.

<table>
<thead>
<tr>
<th>Old field name</th>
<th>Corresponding new name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alumno</td>
<td>Subject</td>
</tr>
<tr>
<td>Programa</td>
<td>Program</td>
</tr>
<tr>
<td>Tecnica</td>
<td>Technique</td>
</tr>
<tr>
<td>Tiempo 1</td>
<td>Technique application time</td>
</tr>
<tr>
<td>Ex. Rel.</td>
<td>Rel. Exp.</td>
</tr>
<tr>
<td>No. Abs/clases</td>
<td>No. of classes</td>
</tr>
<tr>
<td></td>
<td>No. of Abstraction</td>
</tr>
<tr>
<td></td>
<td>% fallos</td>
</tr>
<tr>
<td></td>
<td>% estimated defects</td>
</tr>
<tr>
<td></td>
<td>% fallos</td>
</tr>
<tr>
<td></td>
<td>% estimated defects</td>
</tr>
<tr>
<td></td>
<td>% fallos</td>
</tr>
<tr>
<td></td>
<td>% estimated defects</td>
</tr>
<tr>
<td></td>
<td>% fallos</td>
</tr>
<tr>
<td></td>
<td>% estimated defects</td>
</tr>
</tbody>
</table>

Table 5.3: Old and new field names correspondence in the UPM 05 file

5.3.3.3 Mapping the New Definition with the ORT 05 File

The field names: tecnicas, programa, version, porcentaje and No. casos have the same translation as in Table 5.4. Others are:

²This field was separated into two: N Abs – No. of Abstraction and clases – No. of classes
Table 5.4: Old and new field names correspondence in the ORT 05 file

5.3.3.4 Mapping the New Definition with the UPV 05 File

The original data field names correspond to the new names as shown in Table 5.5. The fields: alumno, grupo, programa, tecnica, version, No. Abs/clases, % fallos, confianza, porcentaje, No. casos, Tiempo, Ex. Rel. and Ex. Abs. have corresponding names as indicated in previous sections.

Table 5.5: Old and new field names correspondence in the UPV 05 file

5.3.3.5 Mapping the New Definition with the US 05 File

In the US 05 file, the fields: programa, tecnica, version, nombre (1 and 2) and alumno (1 and 2) correspond to the names previously shown. Other correspondence between the original terms and the new terms are shown below (Table 5.6).

3Caminos is No. of abstraction
Summarizing the Results of a Series of Experiments: Application to Effectiveness of Software Evaluation Techniques

<table>
<thead>
<tr>
<th>Old field name</th>
<th>Corresponding new name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pareja</td>
<td>Pairs</td>
</tr>
<tr>
<td>Asignacion</td>
<td>Group</td>
</tr>
<tr>
<td>ExpRel*</td>
<td>Rel. Exp.</td>
</tr>
<tr>
<td>ExpAbs</td>
<td>Abs. Exp.</td>
</tr>
<tr>
<td>Proc</td>
<td>Origin</td>
</tr>
<tr>
<td>Clases</td>
<td>No. of classes</td>
</tr>
<tr>
<td>Casos</td>
<td>No. of test cases</td>
</tr>
<tr>
<td>Tgen</td>
<td>Technique application time</td>
</tr>
<tr>
<td>AutoEval</td>
<td>Confidence</td>
</tr>
</tbody>
</table>

Table 5.6: Old and new field names correspondence in the UPV 05 file

5.4 The Master Excel File

The works described in this chapter lead to the creation of a universal excel file named “master sheet.xls”. It is universal in the sense that all other data files of the experiments derived their data fields as appropriate from it. Therefore, none of the data files for all the experiments will henceforth contain a data field not defined in this file. This sheet may grow in the future depending on research direction. However, this is done to establish a data reporting standard for other experiments in the series. This will make comparison of experiment result not only possible but also meaningful.

As mentioned earlier, the file is made up of four worksheets: subject data, observable fault, failure visibility and observed fault.

5.4.1 Subject Data Sheet

The subject data sheet contains general experimental subject information. Also contained in the sheet are time records of each subject’s performance on the different tasks of the experiment.

The fields contained in this sheet are: Subject Number, Subject1, Subject2, Pair, Origin1, Origin2, Program, Technique, Version, Group, Rel. Exp1., Rel. Exp.2, Abs. Exp.1, Abs. Exp.2, Motivation, Technique application time, Test case execution time, Failure/fault detection time, No. of abstraction, No. of test cases, No. of classes, Attendance, % estimated defects and Confidence.

\*This property appears in duplicate, each for a member of the pair. Same goes for ExpAbs, Proc, nombre and alumno
5.4.2 Observable Fault Sheet

The record about the number of test cases relevant to the faults in the programs, developed by each subject is contained in this sheet. The fields contained in this sheet are: Subject number, Pair, Subject1, Subject2, Origin1, Origin2, Program, Technique, Version, Group, F1 – F7, Total, Percentage, Visible, Attendance and CaF1 – CaF7.

Each of the ‘F’ fields corresponds to each of the faults in the program. In this sheet, it connotes whether test case was developed for a particular fault or not.

5.4.3 Failure Visibility Sheet

This sheet records the number of failures detected while executing the test cases recorded in the observable fault sheet. The sheet consists of: Subject number, Pair, Subject1, Subject2, Origin1, Origin2, Program, Technique, Version, Group, F1 – F7, Total, Visible and Percentage data fields.

The F1 – F7 fields here represents the failures/faults that are detected or not.

5.4.4 Observed Faults Sheet

The observed fault sheet presents the logical ‘AND’ of the corresponding F1- F7 fields from the observable fault and the failure visibility sheets. It shows the faults that were actually detected through the test cases developed for them. The fields in the sheet are: Subject number, Pair, Subject1, Subject2, Origin1, Origin2, Program, Technique, Version, Group, F1 – F7, Total, Visible and Percentage.

5.5 Creation of the Specific Sheets

In order to re-create the individual experiment sheets from the master sheet described in the last section, a set of basic data fields are selected from the sheet as demanded by how the experiment was conducted or originally documented. For example, to create the new UPM 01 file, the following data fields are necessary to define the subject data sheet: Subject1, Subject number, Program, Technique, Version, Rel. Exp1., Abs. Exp1., Motivation, Technique application time, Test case execution time, Failure/fault identification time, confidence, % estimated defects, No. of test cases and No. of abstraction. Other fields in the subject data sheet of the master file are not
required for this file. Whereas, all the fields in the subject data sheet of the master file except motivation and No. of abstraction are used in creating the sheet for US 05 file.

Consequently, the sheets in each of the files are regular subset of the corresponding sheets of the master file.

The relationships between the old files and the new files are summarized with tables in Appendix B. For each of the fields in the new sheets, the tables present for each of the experiment, the previous location (old sheet name) and the old field name. The fields that are not applicable to any of the years are printed in grey.
Chapter 6

Global View of the Experiments

Apart from restructuring the data files and creating a directory for the experiments, another aim of this study is to pool the result of the analysis of all the experiments together. This facilitates the deduction of any possible pieces of knowledge from the collection of the analysis. As far as the joint analysis is concerned, this study will focus primarily on the effectiveness of the evaluation techniques. By effectiveness, we mean the capability of each of the techniques to expose the faults or in other words to produce test cases that are capable of exposing the failures in the programs.

6.1 A Process to Summarize the Results of Several Experiments

We were faced with the task of synthesizing the analysis results of the different experiments in Experiment II and deducting a common knowledge thereof. At this stage of the research, we decided not to use the formal aggregation technique yet for this purpose. Unfortunately, we found no other existing systematic method to perform this task. Therefore, we created a qualitative deduction approach (informal aggregation) to systematically synthesize the results.

This approach is a four-step approach. The steps which will be elaborated further in subsequent sections are:

- **Extraction**: Extract the significance value of all the treatments from the various ANOVA tables and summarize them in one table.
- **Classification**: Classify the patterns using some code (e.g., alphabets).
- **Classification Ranking**: Study the distribution pattern in the code table and rank the pattern.
- **Deduction**: Study each category and deduce evidence from each.

The steps will be explained further in the following subsections while the rest of the chapter will be dedicated to showing how it was applied in our own particular case.
6.1.1 The Extraction Step

The main purpose of this step is to present the ANOVA result (significance) of all the experiments in a single table. At this level, it is important that all the experiments have the same number of treatments (main and interaction). If all the experiments were not analyzed using the same value for the confidence level, then the researcher needs to make a choice out of two options:

I. Flexible combination: Accommodate the different confidence levels as used. For example, if one experiment used 90% and the other 95%. The researcher will apply these two levels to all the experiment and extract significance values that fall to both levels. However, it is advisable to make a distinction between which values were accommodated for which confidence level. This approach can be viewed as downgrading.

II. Strict combination: The researcher decides to maintain the higher confidence level; therefore he will only extract treatment values that satisfy this condition or better re-analyze the affected experiment. This is more or less an upgrading approach.

So, the main idea of this step is to extract the significance values that satisfy the researcher’s criteria from the different experiment analysis result and present them in a table (see Table 6.2 for a sample).

6.1.2 The Classification Step

The classification step basically looks at the available significance options and assigns unique code to each distinct possible combination of values. The distinct possible combination (say ‘y’) is usually “y=2x – 1”, where x is (the number of accommodated confidence level + 1). For example, in the hypothetic situation cited above, if 90% and 95% are the acceptable confidence level, as decided by the researcher. Then, x = 3 and y = 2³ – 1 = 7. The deducted 1 is usually an impossible situation.

In this step, it is helpful to create a table of ‘x’ columns and (y + 1) rows to accommodate header. The headings will be the different confidence levels and the Not significant option. The cells will then be filled logically (coded) with 0s and 1s (Yes/No or True/False). Each column will afterwards be coded, say alphabetically, to distinguish them from each other. This step results in the creation of a classification table (see Table 6.3).
6.1.3 The Classification Ranking

After the classification, the next step is to rank (numerically) the classification table based on the perceived strength or clarity of the knowledge presented by the combination in each column (the row entry). This step becomes tricky, most especially when using the flexible combination. It is usually of three sub steps:

- **Assignation:** Assigning numerical ranks to the different codes. So, some decisions will have to be made upfront concerning the interpretation of the significance values. For example, we need to decide whether a treatment significant in different experiments at the 95% is same in knowledge contribution as that is not significant at all or not. Also, such a decision will have to be taken between all the confidence interval used and a relationship must be established between them i.e., if one is stronger than the other. A more concrete example follows in subsequent sections.

- **Interpolation:** The summarized ANOVA table produced in step 1 is then interpolated with corresponding codes and ranks. The usefulness of the codes becomes more pronounced in situation where certain treatments have the same rank qualification but different codes. The code will tell us what combination of values lead to the rank.

- **Streamlining:** After this step, it will be possible to make an intermediate decision table, which will show for each treatment, how many experiments fell under each confidence level. This will enable us to decide whether such treatment is still of interest to further analysis or not. For example, in our case, a treatment with all or more than 75% of the experiments experiment not significant to the synthesis process. A treatment with 50% significant and 50% not is tagged as ambiguous, while something stronger is either significant with strong knowledge evidence or tagged as having tendency to be significant. At the end we decided to work with the last three classes and discarded only the non significant ones.

6.1.4 The Deduction Step

In this step, the set of treatments decided to be of further relevance to analysis are thus selected and studied in-depth. At this stage, the researcher will have to decide what statistical yard sticks to use in establishing/extracting relationship between the different experiments for each treatment selected from step 3. In our own concrete case, we used the mean values, the confidence interval,
the profile plot and the stock plot to establish behavior of treatments in the different experiments. Also, we decided that since this is not the analysis of a single experiment, it is better to also study the main effects even if there were interaction effects involving the main effect. The approach to studying the main effect is different to that of interaction effects:

- For any significant main effect selected from step 3, all the experiments shall be studied to have a general insight into the factor’s characteristics. Even those that are not specifically significant.
- However, for the interaction effects, those experiments that are significant at the specified confidence level(s) shall be selected for study.

6.2 Summary of the Factors’ Significance

In order to achieve a result that is representative of all the experiments, we chose to use the flexible combination option - the 0.01 and 0.05 were considered as the acceptable significance level for the treatment effects – main or interaction. We did this to see if we can have a trend across all results because it was difficult to establish a common tendency among the results of the experiments initially. In Table 6.1, a quick look of the significant main and interaction effects is presented without recourse to actual values. Black ink filled column indicates treatment in respective experiments, significant at 99% confidence level, while grey ink fill indicates the ones significant at 95% confidence level and white indicates not significant.

The actual value of each treatment and their interaction is presented in Table 6.3. The bold face fonts signify 0.01 significance level, normal fonts signifies 0.05 significance level and the grayed fonts indicates not significant at both levels.

6.3 Classification

In this section, we will attempt to abstract the pattern of distribution from Table 6.3. As earlier mentioned and discernible form Table 6.1, three situations exist as far as factors’ significance is concerned. The first category, printed in bold, are those factors that are significant at 0.01 level, the second, printed in normal font, are those that are significant only if the significance level is reduced to 0.05 and of course the last category printed in grey are those that are not significant at both levels. To analyze how the factors in Table 6.3 interplay concerning these three categories, each
possible distinct combination of these categories is assigned an alphabetical tag (Table 6.2) to distinguish the pattern of combination for each factor across the experiments. Though, there are eight possible combinations as opposed to seven shown in Table 6.2, this is when a factor does not have any value that fall to any of the three categories as explained above. This is an impossible situation, thus, we are left with seven choices.

Table 6.1: Overview of the significant main and interaction effects

<table>
<thead>
<tr>
<th>Code</th>
<th>Significant at 0.01</th>
<th>Significant at 0.05</th>
<th>Not significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>B</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>C</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>D</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>E</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>F</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>G</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 6.2: Coding of the possible combination of treatment based on level significance
### Table 6.3: Significance value from the ANOVA table for all experiments

<table>
<thead>
<tr>
<th>Notes</th>
<th>Treatments</th>
<th>UPM 01</th>
<th>UPM 02</th>
<th>UPM 03</th>
<th>UPM 04</th>
<th>UPM 05</th>
<th>UPV 05</th>
<th>ORT 05</th>
<th>UdS 05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corrected model (sig/power)</td>
<td></td>
<td>0.000/0.99</td>
<td>0.005/0.92</td>
<td>0.003/0.95</td>
<td>0.014/0.86</td>
<td>0.002/0.97</td>
<td>0.036/0.54</td>
<td>0.004/0.54</td>
<td>0.005/0.89</td>
</tr>
<tr>
<td>Model used: Type III</td>
<td>Program</td>
<td>0.025</td>
<td>0.390</td>
<td>0.032</td>
<td>0.907</td>
<td>0.002</td>
<td>0.026</td>
<td>0.003</td>
<td>0.006</td>
</tr>
<tr>
<td>Sum of Squares</td>
<td>Technique</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.764</td>
<td>0.003</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td>0.256</td>
<td>0.095</td>
<td>0.114</td>
<td>0.323</td>
<td>0.361</td>
<td>0.388</td>
<td>0.635</td>
<td>0.057</td>
</tr>
<tr>
<td></td>
<td>Fault</td>
<td>0.062</td>
<td>0.316</td>
<td>0.853</td>
<td>0.009</td>
<td>0.215</td>
<td>0.013</td>
<td>0.000</td>
<td>0.050</td>
</tr>
<tr>
<td></td>
<td>Program</td>
<td>* 0.006</td>
<td>0.017</td>
<td>0.001</td>
<td>0.048</td>
<td>0.040</td>
<td>0.022</td>
<td>0.014</td>
<td>0.076</td>
</tr>
<tr>
<td></td>
<td>Technique</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td>* 0.145</td>
<td>0.590</td>
<td>0.797</td>
<td>0.156</td>
<td>0.061</td>
<td>0.054</td>
<td>0.710</td>
<td>0.832</td>
</tr>
<tr>
<td></td>
<td>Program * Fault</td>
<td>0.000</td>
<td>0.041</td>
<td>0.056</td>
<td>0.002</td>
<td>0.000</td>
<td>0.021</td>
<td>0.001</td>
<td>0.018</td>
</tr>
<tr>
<td></td>
<td>Technique * Version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version * Fault</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Technique * Fault</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version * Fault</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.4 Ranking of the Classification Table

The coding shown in Table 6.3 distinguishes each distinct pattern, however, this has little relevancy to the strength of knowledge deductible form the combination. Put simply, different code does not necessarily mean different level of evidence. Therefore, the table (Table 6.3) is further analyzed and thus ranked based on the strength of the knowledge piece deductible from each row (Table 6.4).

<table>
<thead>
<tr>
<th>Code</th>
<th>Significant at 0.01</th>
<th>Significant at 0.05</th>
<th>Not significant</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 6.4: Ranking and classification of effect behavior

As seen from Table 6.4, two different codes, for example, C and E ended up having the same rank. This implies that though the pattern of information in the two rows is different, but the information presented by the two has the same knowledge significance according to our analysis pattern explained in the next section.

6.4.1 Explanation of the Ranks

The ranks are introduced based on the strength of knowledge deductible by the combination of values presented by each row. For example, any treatment that has values significant only at 0.01 for all the experiments is considered to present information of similar weight to that of a treatment which is not significant for all the experiments. The ranks are explained in details below:

- **Rank 1**: The rows ranked 1 are those that provide strongest homogenous values either significant at 0.01 or not significant at both 0.01 and 0.05. This provides a clear stand at either end. Evidence generated from this rank will be considered very strong.

- **Rank 2**: This rank is used for the code that has the tendency of being significant. Though, how strong the evidence is, will depend solely on which of the significance levels have the most value. Evidence deducted from this rank will be tagged quite strong.
• **Rank 3**: The code with values that are absolutely significant only at 0.05. We will interpret this as having the tendency to be significant. This rank will be considered strong in evidence generation.

• **Rank 4**: This rank is used for a row with all the possibilities of significance or non significant values, the information deductible from this row will depend on which value dominates or at worst it will be difficult to make any meaningful interpretation without further evidence. This has the tendency of swinging any way or at times it is possible to provide no clue as to what can be made of the values. For example, for a significant or non significant situation that lies between 0% - 29% presents no tendency, 30% - 40% we do not know what to say, 40% - 60% there seems to be a tendency and 60% upward there is a tendency.

• **Rank 5**: As mentioned above, this is an impossible situation. So it is not displayed in the table.

### 6.4.2 Interpolation of the Ranking into the ANOVA Table

Based on the ranks and codes in Table 6.4, the ANOVA (Table 6.2) was interpolated (Table 6.5). The codes and ranks are assigned following the standard as explained from Table 6.3 and Table 6.4.

#### 6.4.2.1 General Comments on the Interpolated ANOVA Table

The following comment can be deducted preemptively from Table 6.5 as a tendency of what evidence is likely to result from the results of analysis of the series of experiments being studied:

- **Version**: There seem to be a strong indication; based on the eight experiments in the series under analysis that version does not in any way have influence on the effectiveness of the evaluation techniques.

- **Technique**: There is also a strong evidence to say that technique has a strong effect on the outcome of the experiment.

- **Fault**: There is no clear evidence to say whether or not the fault type has any influence on the ability of an evaluation technique to detect it.

- **Program**: There is a fairly strong evidence to say that the type of program being tested has an influence on the effectiveness of the evaluation technique.
• **Program * version**: The evidence in Table 6.5 supports the fact that the interaction between program and fault has no effect on the effectiveness of software evaluation technique.

• **Technique * version**: The table shows fairly strong evidence that the technique-version interaction has no significant effect on the effectiveness of the evaluation technique in our experiment.

• **Fault * version**: This is the same case as the technique * version interaction.

• **Program * fault**: The distribution in Table 6.5 points to the fact that the interaction between program and fault has a tendency of being significant.

This information is summarized in Table 6.6.

<table>
<thead>
<tr>
<th>Code</th>
<th>Rank</th>
<th>Treatment</th>
<th>Numbers of experiments significant at:</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>Version</td>
<td>0/8</td>
<td>0/8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Program * version</td>
<td>0/8</td>
<td>0/8</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>Technique * version</td>
<td>0/8</td>
<td>2/8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Fault * version</td>
<td>0/8</td>
<td>2/8</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>Technique</td>
<td>7/8</td>
<td>0/8</td>
</tr>
<tr>
<td>G</td>
<td>4</td>
<td>Program</td>
<td>3/8</td>
<td>3/8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Fault</td>
<td>3/8</td>
<td>1/8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Program * fault</td>
<td>3/8</td>
<td>4/8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Technique * fault</td>
<td>1/8</td>
<td>3/8</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>Program * technique</td>
<td>2/8</td>
<td>5/8</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>19/80</td>
<td>20/80</td>
</tr>
</tbody>
</table>

Table 6.5: Interpretation of ranks in the ANOVA table

**6.4.3 ANOVA Analysis Summary**

The information in the status column of Table 6.6 can be summarily displayed as shown in Table 6.7. Technique is the only treatment that clearly stands as significant. Program dominates the category that has tendency to be significant while version and all its interaction fell to the ‘not significant’ category.
<table>
<thead>
<tr>
<th>Notes</th>
<th>Treatments</th>
<th>Significance</th>
<th>Code</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corrected model (sig/power)</td>
<td></td>
<td>UPM 01</td>
<td>0.000/0.994</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UPM 02</td>
<td>0.005/0.928</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UPM 03</td>
<td>0.003/0.956</td>
<td></td>
</tr>
<tr>
<td>Model used: Type I11 Sum of</td>
<td>Program</td>
<td>UPM 04</td>
<td>0.014/0.863</td>
<td>0.006</td>
</tr>
<tr>
<td>Squares</td>
<td>Technique</td>
<td>UPM 05</td>
<td>0.002/0.973</td>
<td>0.006</td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td>ORT 05</td>
<td>0.036/0.541</td>
<td>0.002</td>
</tr>
<tr>
<td></td>
<td>Fault</td>
<td>UPM 05</td>
<td>0.004/0.941</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Program *</td>
<td>UPM 05</td>
<td>0.007/0.950</td>
<td></td>
</tr>
<tr>
<td>Technique</td>
<td>Version</td>
<td>UPM 05</td>
<td>0.008/0.951</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fault</td>
<td>UPM 05</td>
<td>0.048/0.863</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Program *</td>
<td>UPM 05</td>
<td>0.009/0.973</td>
<td></td>
</tr>
<tr>
<td>Technique * Version</td>
<td>Fault</td>
<td>UPM 05</td>
<td>0.040/0.863</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Program *</td>
<td>UPM 05</td>
<td>0.002/0.941</td>
<td></td>
</tr>
<tr>
<td>Technique * Version</td>
<td>Fault</td>
<td>UPM 05</td>
<td>0.022/0.941</td>
<td></td>
</tr>
</tbody>
</table>

Significance levels: 0.01 and 0.05

Table 6.6: ANOVA summary with the codes and ranks
Henceforth, based on the outcome of extractions presented in Table 6.6 and 6.7, further to deduct available pieces of knowledge will be focused only on the factors that are in the ‘significant’ and ‘significant tendency’ category. Afterwards, we will try to get some insight from the factors or interactions that have the ‘not clear’ status. The results will be combined. The ‘not significant’ category will not be considered any further.

<table>
<thead>
<tr>
<th>Significant</th>
<th>Significant tendency</th>
<th>Not clear</th>
<th>Not significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technique (E4)</td>
<td>Program (G4)</td>
<td>Fault (G4)</td>
<td>Version (A1)</td>
</tr>
<tr>
<td>Program * Fault (G4)</td>
<td>Technique * fault (G4)</td>
<td>Program * version (A1)</td>
<td></td>
</tr>
<tr>
<td>Program * technique (G4)</td>
<td>Technique * version (C4)</td>
<td>Fault * version (4)</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.7: Summary of factors' status

6.5 Deduction

Now that we have been able to decide which main or interaction effect will be studied; we will further explore the SPSS analysis results by examining the mean values, confidence intervals, profile plots of each of the selected treatments, as well as their excel stock plots. This will afford the opportunity of gaining more insight into the pattern of behavior exhibited by each of the treatments. The categories will be discussed in turn, starting from the category that is most clear to the unclear.

6.5.1 Status: Significant

As mentioned, the treatment(s) that has a clear significant effect will be discussed first. Technique is the only treatment in this category (Table 6.7). Since seven out of eight of the experiments are significant at 0.01 significance level and because ‘technique’ is a main effect, the ‘techniques’ profile plot (Table 6.8) for all the experiments will be studied.

In Table 6.8, the SPSS profile plot of technique for all the experiments in the series is presented with comments on the observed behavior of each evaluation technique in relation to the other.

Each column of Table 6.8 is divided into four rows. The first row contains the profile chart to be interpreted, the second row contains the mean value as plotted on the chart, displayed from highest to lowest and the third row is a brief statement on the observed behavior of the techniques on the chart while the last row is a brief statement on the observed behavior.
Summarizing the Results of a Series of Experiments: Application to Effectiveness of Software Evaluation Techniques

<table>
<thead>
<tr>
<th>UPM 01</th>
<th>UPM 02</th>
<th>UPM 03</th>
<th>UPM 04</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Graph" /></td>
<td><img src="image2" alt="Graph" /></td>
<td><img src="image3" alt="Graph" /></td>
<td><img src="image4" alt="Graph" /></td>
</tr>
</tbody>
</table>

Significance level: 0.01

- F: 82.95, S: 76.70, R: 45.53
- F: 81.66, S: 79.62, R: 35.31
- S: 87.87, F: 82.72, R: 47.62
- F: 82.57, S: 79.80, R: 55.56

- (F = S) > R
- Functional behaved same as structural
- Review behaved worst

<table>
<thead>
<tr>
<th>UPM 05</th>
<th>UPV 05</th>
<th>ORT 05</th>
<th>UdS 05</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image5" alt="Graph" /></td>
<td><img src="image6" alt="Graph" /></td>
<td><img src="image7" alt="Graph" /></td>
<td><img src="image8" alt="Graph" /></td>
</tr>
</tbody>
</table>

Significance level: 0.01

- Not Significant
- S: 80.29, F: 70.33
- S: 76.48, F: 67.99, R: 41.80

- (F = S) > R
- Functional behaved same as structural
- Review behaved worst

<table>
<thead>
<tr>
<th>Key</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F – Functional technique, R – Code review technique, S – Structural technique, &gt; - better behavior, = - same behavior</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.8: SPSS profile plot for technique
The functional technique and the structural technique behave very much alike, better than the code review except for the ORT 05 where the structural technique behaved better than the functional technique. In this case, code review was not used. There is probably a factor influencing the results of the experiments that needed to be examined further.

Nevertheless, in five situations (UPM 01, 02, 04, 05 and UPV 05) the functional technique has mean value greater than the structural technique. The story is otherwise for UPM 04, ORT 05 and UdS 05. The difference in their mean values is usually negligible in most cases, so is the difference in the confidence interval.

6.5.2 Status: Significant Tendency

The program, program-fault interaction and the program-technique interaction are the factors and interactions that were classified as having tendency of significant effect in the experiments. For program which is a main effect, all the experiments will be discussed, while for the interaction effects, only the experiments that are significant (at 0.01 or 0.05) will be analyzed. The profile plot for program is shown in Table 6.9; program-fault interaction is presented in Table 6.10 and program-technique interaction in Table 6.11.

6.5.2.1 Program

The SPSS profile plot for ‘program’ for each of the experiments is presented in Table 6.9. Each of the experiment columns is divided into four rows:

- The chart of the factor’s behavior
- Mean values of the different levels in decreasing order
- Representation of the behavior as evident from the chart
- Concise statements illustrating the behavior

No definite or regular pattern is discernible across all the experiments in Table 6.9. Four different patterns can be deduced from the table.

1. All programs behaved the same: This is the situation with the UPM 02 and UPM 04 experiments. All the programs in the experiments exhibit similar behavior. However in all the cases ‘nametbl’ ended up having highest mean value. This happens for the experiments where program was not significant.
### Summarizing the Results of a Series of Experiments: Application to Effectiveness of Software Evaluation Techniques

#### UPM 01
- **Na**: 75.33, **Nt**: 65.45, **C**: 65.39
- **Significant level**: 0.05
- **Na > (C = Nt)**
- Nametbl behaved better than cmdline and ntree
- Cmdline behaved same as ntree

#### UPM 02
- **Na**: 69.32, **Nt**: 64.51, **C**: 62.75
- **Not Significant**
- C = Na = Nt
- The three programs behaved the same

#### UPM 03
- **Na**: 79.72, **C**: 70.79, **Nt**: 67.70
- **Significance level**: 0.05
- Na > (C = Nt)
- Nametbl behaved better than cmdline and ntree
- Cmdline behaved same as ntree

#### UPM 04
- **Na**: 73.81, **Nt**: 72.22, **C**: 71.89
- **Not Significant**
- C = Na = Nt
- All the programs behaved equally

#### UPM 05
- **Nt**: 82.56, **C**: 71.15, **Na**: 66.27
- **Significance level**: 0.01
- Nt > (C = Na)
- Ntree behaved better than nametbl and cmdline
- Nametbl and cmdline behaved the same

#### UPV 05
- **Na**: 69.32, **Nt**: 72.14, **C**: 62.44
- **Significance level**: 0.05
- (Na = Nt) > C
- Nametbl and ntree behaved equal better than cmdline

#### ORT 05
- **Na**: 79.94, **C**: 70.32
- **Significance level**: 0.01
- Na > C
- The nametbl behaved better than the cmdline program

#### UdS 05
- **Na**: 62.86, **C**: 53.08
- **Significance level**: 0.01
- (Na = Nt) > C
- Nametbl and ntree behaved equally better than cmdline

**Key**: C – cmdline, Nt – ntree, Na – nametbl, > - better behavior, = same behavior

Table 6.9: SPSS profile plots for program
2. ‘Nametbl’ behaved better than ‘cmdline’ and ‘ntree’: In this case, ‘ntree’ and ‘cmdline’ exhibit similar behavior inferior to ‘nametble’. This was the case in UPM 01, ORT 05 and UPM 03 experiments. Both of the UPM experiments are significant at 0.05 while the ORT experiment was significant at 0.01.

3. ‘Nametbl’ and ‘ntree’ behaved similarly: In this situation both behaved better than the ‘cmdline’ program. Examples are in UPV 05 and UdS 05 experiments. UPV 05 is significant at 0.05 while UdS 05 is significant at 0.01.

4. ‘Ntree’ behaved better than ‘nametbl’ and ‘cmdline’: The last situation is when ‘ntree’ behaved better than the two other programs which behaved similarly. This happens in UPM 05 which is significant at 0.01.

In all, the results presented in the table shows that ‘nametbl’ exhibited the highest tendency of faults being detected in it. Though, this result interpreted in isolation, may not be so much reliable as there are interaction effects involving ‘program’ and other factors in all the experiments.

It is interesting to note the fact that the result of UPM 05 exhibit a behavior which is somewhat different to the pattern noticed in other results. ‘Nametbl’ has the lowest value and ‘ntree’ the highest, which does not happen in any other experiment. This may probably be due to the influence of the order of program execution either in the experiment or the other experiments.

Closer examination of the mean value of the three programs in all the experiments revealed that UPM 01, UPM 02, UPM 03 and UPM 04 experiments exhibit likely effect of execution order in the results. The program executed first has the least mean and the one executed last has the largest mean value. Whereas, this was not the case with UPM 05 where the program executed first ‘ntree’ ended up having the highest value and ‘nametbl’ which was executed last had the least mean value.

There was no order of execution in the remaining experiments. In the three, ‘nametbl’ behaved equal to ‘ntree’ twice – UPV 05 and UdS 05; and equal as ‘cmdline’ in ORT 05 experiment.

6.5.2.2 Program – fault interaction

Seven of the eight experiments have a significant interaction effect between ‘program’ and ‘fault’. Four of the programs (UPM 01, UPM 04, UPM 05 and ORT 05) are significant at 99% confidence level while UPM 02, UPV 05 and UdS 05 are significant at 95% confidence level.
The series of tables (Table 6.10 – 6.16) present the result of analysis regarding this interaction. The tables have seven rows each:

- **Experiment**: The title of the experiment.
- **Significance level**: The level at which the experiment was significant for this interaction.
- **Interaction type**: The type of interaction in the profile plot. Usually, there are two types of interaction, the ordinal and disordinal interaction types [22].
  - **Ordinal**: This happens when the effect of a treatment not always equal across all levels of another treatment but the group difference remains constant irrespective of how they are combined. The difference is always in the same direction greater or lower than another level of the same treatment. In this case a certain pattern is usually deductible from the relationship. The plot lines ten to converge without intersecting.
  - **Disordinal**: When the differences between level switch depending on the combination with level from another treatment. The difference varies in magnitude and direction. The plot lines cross over each other in this case.
- **Relationship**: A semi-formal expression of the behavioral pattern as deducted from the analysis result
- **Profile plot**: The plot of treatment profile from the SPSS
- **Stock plot**: A graph that displays the confidence interval of the mean response variable for each fault grouped by program.
- **Comments**: Some comments on observations made from the two charts – stock and profile plots on how the factors relate.

The profile plot for most of the experiments displays disordinal relationship. Ordinarily it is advisable to redesign and rerun the experiments to obtain an ordinal relationship rather than deduct interpretation from the disordinal graphs [22]. Nevertheless, for the purpose of this study we will go ahead and use the graphs as they are.

We use cluster analysis to analyze the overall effect of the fault and program interaction in order to establish a general behavior pattern across all the experiments. With the charts presented in Tables 6.10 to 6.16, four characteristics (general average behavior pattern) were established regarding how the faults behaved in the programs: very good, good, average and poor fault behavior. To establish these characteristics, three steps were followed:
<table>
<thead>
<tr>
<th>Experiment</th>
<th>UPM 01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance level</td>
<td>0.01</td>
</tr>
<tr>
<td>Interaction type</td>
<td>Disordinal</td>
</tr>
<tr>
<td>Relationship</td>
<td>No definite pattern can be established</td>
</tr>
</tbody>
</table>

**Profile plot**

![Profile plot of UPM 01](image)

**Stock plot**

![Stock plot of UPM 01](image)

**Comment**
- There is a wide variability in all the programs depending on fault.
- f2 and f6 behave closely identical for all the programs.
- f3 and f7 behave identically in ‘cmdline’ and ‘nametbl’ better than ‘ntree’.
- f4 behave identically in ‘cmdline’ and ‘nametbl’ worse than ‘ntree’.
- f5 behave identically in ‘ntree and ‘nametbl’ better than ‘cmdline’.
- f1, f2, f6 and f7 behave identically in ‘cmdline’.
- f2, f3, f6 and f7 behave identically in ‘nametbl’.
- f5 records low value in ‘cmdline’, f4 in ‘nametbl’ and f7 in ‘ntree’.
- f3 records high value in ‘cmdline’ and f1, f3, f5 and f7 in ‘nametbl’ and f4 and f5 in ‘ntree’.

Table 6.10: Program-fault interaction for UPM 01
### Table 6.11: Program-fault interaction for UPM 02

<table>
<thead>
<tr>
<th>Experiment</th>
<th>UPM 02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance level</td>
<td>0.05</td>
</tr>
<tr>
<td>Interaction type</td>
<td>Disordinal</td>
</tr>
<tr>
<td>Relationship</td>
<td>No definite pattern can be established</td>
</tr>
</tbody>
</table>

#### Experiment

- **UPM 02**

- **Significance level**: 0.05
- **Interaction type**: Disordinal
- **Relationship**: No definite pattern can be established

#### Comments

- All faults display wide variability in behavior depending on the program
- f3 and f5 behave identical in ‘cmdline’ and ‘ntree’, worse than ‘nametbl’
- f2 behave identical for ‘cmdline’ and ‘ntree’, better than ‘nametbl’
- f7 behave identical in ‘cmdline’ and ‘nametbl’ better than in ‘ntree’; same goes for f6 but worse than in ‘ntree’
- f1 behave identical in ‘nametbl’ and ‘ntree’ better than in ‘cmdline’
- f5, f6, f7 record high values in ‘cmdline’ and ‘nametbl’; so is f2, f3 and f4
- f1, f2 and f3 behave the same in ‘ntree’; so is f4 and f5
- f3, f4 and f7 record high values in ‘cmdline’, f1, f3 and f5 in ‘nametbl’ and f1, f2 and f6 in ‘ntree’
- f1 record low value in ‘cmdline’, f2 in ‘nametbl’ and f7 in ‘ntree’
Chapter 6: Global View of the Experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>UPM 04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance level</td>
<td>0.01</td>
</tr>
<tr>
<td>Interaction type</td>
<td>Disordinal</td>
</tr>
<tr>
<td>Relationship</td>
<td>No definite pattern can be established</td>
</tr>
</tbody>
</table>

**Profile plot**

**Stock plot**

**Comments**

- f1, f2, f6 and f7 behave identically for all programs
- f3 behave identical in ‘cmdline’ and ‘nametbl’ better than ‘ntree’
- f4 behaved similar in ‘cmdline’ and ‘ntree’ better than ‘nametbl’
- f5 behavior is identical in ‘nametbl’ and ‘ntree’ better than ‘cmdline’
- f1 and f3 have identical behavior in ‘cmdline’
- f1, f2, f3 and f5 have identical behavior in ‘nametbl’
- f2 and f5 have identical behavior in ‘ntree’
- f1, f3 and f6 have high value in ‘cmdline’, f1, f2, f3, f5 and f6 in ‘nametbl’ and f2 and f5 in ‘ntree’
- f5 have low values in ‘cmdline’, f4 in ‘nametbl’ and f7 in ‘ntree’

Table 6.12: Program-fault interaction for UPM 04
### Replication

<table>
<thead>
<tr>
<th>Replication</th>
<th>UPM 05</th>
</tr>
</thead>
</table>

### Significance level

<table>
<thead>
<tr>
<th>Significance level</th>
<th>0.01</th>
</tr>
</thead>
</table>

### Interaction type

<table>
<thead>
<tr>
<th>Interaction type</th>
<th>Disordinal</th>
</tr>
</thead>
</table>

### Relationship

<table>
<thead>
<tr>
<th>Relationship</th>
<th>No definite pattern</th>
</tr>
</thead>
</table>

### Profile plot

![Profile plot]

### Stock plot

![Stock plot]

### Comments

- There is wide variability in the faults behavior depending on the program.
- F1, f6 and f7 display similar behavior for all programs.
- F2 behave identical in nametbl and ntree worse than cmdline.
- F3 behave identical in cmdline and ntree better than nametbl.
- F4 behave identical in cmdline and nametbl worse than ntree.
- F5 behave similar in nametbl and ntree better than cmdline.
- F3, f4 and f5 have highest values in ntree.
- F3 has lowest value in nametbl.

Table 6.13: Program-fault interaction for UPM 05
Replication: UPV 05
Significance level: 0.05
Interaction type: Disordinal
Relationship: No definite pattern can be established

<table>
<thead>
<tr>
<th>Profile plot</th>
<th>Stock plot</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Profile plot" /></td>
<td><img src="image" alt="Stock plot" /></td>
</tr>
</tbody>
</table>

**Comment**
- f2 and f3 perform equally in all programs
- f1 behave differently in the three programs
- f7 behave identically in ‘cmdline’ and ‘nametbl’ better than in ‘ntree’
- f4 and f6 behave identically for ‘nametbl’ and ‘ntree’ better than ‘cmdline’
- f1 and f2 behave identically in ‘nametbl’, so is f6 and f7
- f4, f5 and f6 have identical behavior in ‘cmdline’
- f2, f3, f4, f5 and f6 behave identically in ‘ntree’
- f4, f6 and f5 have low value in ‘cmdline’ and f7 in ‘ntree’
- f1 and f3 have high value in ‘cmdline’, f3 and f4 in ‘nametbl’ and f2, f3, f4, f5 and f6 in ‘ntree’

Table 6.14: Program-fault interaction for UPV 05
<table>
<thead>
<tr>
<th>Replication</th>
<th>ORT 05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance level</td>
<td>0.01</td>
</tr>
<tr>
<td>Interaction type</td>
<td>Disordinal</td>
</tr>
<tr>
<td>Relationship</td>
<td>C = Na: f3, f6 and f7; Na &gt; C: f4 and f5; C &gt; Na: f1</td>
</tr>
</tbody>
</table>

### Profile plot

![Profile plot](image)

### Stock plot

![Stock plot](image)

### Comment
- This interaction display a bit of disordinal behavior but it is minute enough to be overlooked
- f3, f6 and f7 behave identically for both programs
- f1 behave better in ‘cmdline’ than ‘nametbl’
- f4 and f5 behave identical and better in ‘nametbl’ than ‘cmdline’
- f4 has low value in ‘cmdline’
- f1, f3, f6 and f7 have high value in ‘cmdline’ and f3, f4, f5, f6 and f7 in ‘nametbl’
- Averagely, behaved better or equal to ‘cmdline’ with all faults except f1

Table 6.15: Program-fault interaction for ORT 05
Chapter 6: Global View of the Experiments

<table>
<thead>
<tr>
<th>Replication</th>
<th>UdS 05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance level</td>
<td>0.05</td>
</tr>
<tr>
<td>Interaction type</td>
<td>Disordinal</td>
</tr>
<tr>
<td>Relationship</td>
<td>No definite pattern can be established</td>
</tr>
</tbody>
</table>

**Replication**

**Significance level**

**Interaction type**

**Relationship**

**Profile plot**

![Profile Plot](image)

**Stock plot**

![Stock Plot](image)

**Comment**

- All programs exhibit same behavior with $f_1$ and $f_2$
- $f_3$ behave identically in ‘cmdline’ and ‘ntree’ better than ‘nametbl’
- $f_6$ behave the same in ‘nametbl’ and ‘ntree’ better than ‘cmdline’
- $f_4$ and $f_5$ behave best in ‘ntree’ better than ‘nametbl’ which is better than ‘cmdline’
- $f_1$ and $f_2$ behave identical in ‘cmdline’; so is $f_4$ and $f_5$ as well as $f_6$ and $f_7$
- $f_1$ and $f_2$ behave identical in ‘nametbl’ so is $f_6$ and $f_7$
- $f_3$, $f_4$ and $f_5$ behave identical in ntree; so is $f_1$ an $f_2$
- $f_1$, $f_2$, $f_4$ and $f_5$ behave poorly in ‘cmdline’, $f_1$, $f_4$ and $f_2$ in ‘nametbl’ and $f_1$, $f_2$ and $f_7$ in ‘ntree’
- $f_3$ have high value in ‘cmdline’, $f_6$ and $f_7$ in ‘nametbl’ and $f_3$, $f_4$, $f_5$ and $f_6$ in ‘ntree’

Table 6.16: Program-fault interaction for UdS 05
• Create a performance scale based on the mean values from the profile plots. The scale is something like: Very good $\geq 75$ and above, Good $= 74 – 65$, Average $= 64 – 55$ and Poor $\leq 54$; in this concrete case.

• Create a table and fit in all the experiments. In cases where obvious, that is, all experiments fall within a category, the decision is made here. Else,

• When not obvious, that is, there is disparity in the columns where the mean values fell, the average of the mean values is found and the factor is fixed in the corresponding column of the average. Even, when obvious, the mean is still calculated to double check placement.

Using this approach, the classification for program-fault interaction is shown in Table 6.17. This table does not represent any particular experiment. It is an overview of the average behavioral pattern from all the experiments. This fact also goes for all subsequent tables of this form.

<table>
<thead>
<tr>
<th>Fault behavior (average of marginal means)</th>
<th>Cmdline</th>
<th>Nametbl</th>
<th>Ntree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very good ($\geq 75$)</td>
<td>F3</td>
<td>F3, F5, F7</td>
<td>F3, F4, F5, F6</td>
</tr>
<tr>
<td>Good ($65 \leq 74$)</td>
<td>F1, F2, F6, F7</td>
<td>F1, F2, F6, F1, F2, F5, F6</td>
<td></td>
</tr>
<tr>
<td>Average ($55 \leq 64$)</td>
<td>F4</td>
<td>F4</td>
<td>-</td>
</tr>
<tr>
<td>Poor ($\leq 54$)</td>
<td>F5</td>
<td>-</td>
<td>F7</td>
</tr>
</tbody>
</table>

Table 6.17: Output of cumulated cluster analysis for program-fault interaction

From the table, it can be inferred that:

• F3 (omission, initialization) behave very well in all the programs.

• F1 (omission, cosmetic), F2 (commission, cosmetic) and F6 (omission, control) behaved the same in all the programs. The three have good behavior in all the programs.

• F5 (commission, control) behave poorly in ‘cmdline’ but exhibits good and very good behavior in ‘nametbl’ and ‘ntree’ respectively.

• F7 (commission, computation) behaved poorly in the ‘ntree’ program but shows good behavior in both ‘cmdline’ and ‘nametbl’.

• F4 (commission, initialization) was averagely detected in ‘cmdline’ and ‘nametbl’ but it was better detected in ‘ntree’.

In a little bit more details the following can further be said about the faults with respect to the programs. There are:
• Faults that behave the same irrespective of the program; example is F3 as well as F1, F2 and F6.
• Faults that behave the same in ‘cmdline’ and ‘nametbl’ e.g., F4 and F7.
• Faults that behave well in ‘nametbl’ and ‘ntree’ but not in ‘cmdline’. This is the case of F5.

6.5.2.3 Program – technique interaction

Like the program – fault interaction, seven of the eight experiments (Table 6.18 to 6.24) show significant effect for program – technique interaction. However, only two – UPM 01 and UPM 03, were significant at 99% confidence level, others – UPM 02, UPM 04, UPM 05, UPV 05 and ORT 05, were only significant at 95%. The table columns and rows are organized in the same way as in the explained in Section 6.4.2.2.

Cluster analysis of the pooled mean values across all the experiments resulted in four categories of average performance characterization of the techniques as affected by the programs. The approach described in the last section for Table 6.17 was used here as well. The grouping is shown in Table 6.25.

The behavior exhibited by the techniques as influenced by the programs can be summarized as follows:

• The functional and structural technique behaved the same, better than code review irrespective of the program for all experiments.
• There is variability in the techniques’ performance depending on the program.
• Code review behaved best with ‘ntree’ with a tendency of matching the dynamic techniques’ performance but worst with ‘cmdline’ in all the experiments.
• Functional and structural techniques behaved best with ‘nametbl’ and worst with ‘ntree’ except in UPM 05. This deviation can be considered as negligible.

6.5.3 Status: Not clear

Fault and fault – technique interaction are the effects that fell under this classification. Due to the fact that the analysis in this study involved eight experiments, overlooking the effects in this category may result into loss of vital knowledge concerning the factors. Therefore, the pieces of knowledge deductible from the effects in this section will be used together with those of the effects in the significant and significant tendency categories to have a full grasp of knowledge pieces passed along by all the experiments.
### Replication

<table>
<thead>
<tr>
<th>Replication</th>
<th>UPM 01</th>
</tr>
</thead>
</table>

### Significance level

<table>
<thead>
<tr>
<th>Significance level</th>
<th>0.01</th>
</tr>
</thead>
</table>

### Interaction type

<table>
<thead>
<tr>
<th>Interaction type</th>
<th>Ordinal</th>
</tr>
</thead>
</table>

### Relationship

<table>
<thead>
<tr>
<th>Relationship</th>
<th>(F = S) &gt; R</th>
</tr>
</thead>
</table>

### Profile plot

![Profile Plot](image)

### Stock plot

![Stock Plot](image)

### Comment

- The technique performance is dependent on the program
- The order of program execution have no visible impact on technique performance
- Functional technique behave same as the structural better than code review with all the programs
- Review has poor performance with all the program
- Review has almost the same performance as functional and structural with ‘ntree’
- Functional and structural techniques displayed lower performance with ‘ntree’
- Functional and structural behave best with ‘nametbl’
- Code review behave best with ‘ntree’
- Functional and structural performs better than code review

Table 6.18: Program-technique interaction for UPM 01
Chapter 6: Global View of the Experiments

<table>
<thead>
<tr>
<th>Replication</th>
<th>UPM 02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance level</td>
<td>0.05</td>
</tr>
<tr>
<td>Interaction type</td>
<td>Disordinal</td>
</tr>
<tr>
<td>Relationship</td>
<td>(F = S) &gt; R</td>
</tr>
</tbody>
</table>

**Profile plot**

- The technique performance is dependent on the program
- The impact of order of program execution cannot be established
- Functional and structural techniques behave the same better than code review in all the programs
- Code review exhibits poor performance in all the programs
- The performance of code review improved in ‘ntree’
- Functional and structural techniques display lower performance in ‘ntree’
- Functional and structural techniques behave best with ‘nametbl’
- Code review behaved best in ‘ntree’
- Functional and structural performs better than code review
- Functional technique has higher mean values than the structural technique except in ‘ntree’

**Stock plot**

**Comment**

Table 6.19: Program-technique interaction for UPM 02
### Replication
UPM 03

### Significance level
0.01

### Interaction type
Ordinal

### Relationship
\( (F = S) > R \)

### Profile plot

![Profile plot for UPM 03](image)

### Stock plot

![Stock plot for UPM 03](image)

### Comment
- Technique performance varies according to program
- Order of program execution had no visible impact on the technique performance
- All the techniques perform the same with ‘ntree’
- Functional and Structural exhibits equal performance in all the programs
- Code review exhibits the worst performance
- Code review shows constant performance with ‘cmdline’ and ‘nametbl’
- Functional and Structural show lower performance in ‘ntree’
- Functional and Structural techniques behaved best with ‘nametbl’
- Code review behave best with ‘ntree’
- Functional and Structural performs better than code review only in ‘cmdline’ and ‘nametbl’

Table 6.20: Program-technique interaction for UPM 03
### Chapter 6: Global View of the Experiments

#### Replication
- UPM 04

#### Significance level
- 0.05

#### Interaction type
- Ordinal

#### Relationship
- \((F = S) > CR\)

#### Profile plot

![Profile plot](image)

#### Stock plot

![Stock plot](image)

#### Comment
- All techniques perform the same with ntree
- Functional and structural perform equally with cmdline and nametbl better than code review
- Code review exhibits the lowest performance
- Code review behaved best in code review
- Functional technique behaved best in cmdline
- Structural technique behaved best in nametbl

---

Table 6.21: Program-technique interaction for UPM 04
Replication | UPM 05
--- | ---
Significance level | 0.05
Interaction type | Disordinal
Relationship | \((F = S) > R\)

Profile plot

![Profile plot diagram](image)

Stock plot

![Stock plot diagram](image)

Comment
- Technique performance is dependent on the program
- All three techniques behaved differently for ‘cmdline’
- Functional and structural behave the same for ‘nametbl’ and ‘ntree’ better than code review
- Functional is better than structural which is better than code review with ‘cmdline’
- Code review exhibit the worst behavior
- Code review shows good tendency with ‘ntree’.
- Code review behaved better in ‘cmdline’ than ‘nametbl’
- Functional and structural technique behaved best with ‘ntree’
- Functional technique behaved worst with ‘nametbl’
- Structural technique behaved worst with ‘cmdline’

Table 6.22: Program-technique interaction for UPM 05
<table>
<thead>
<tr>
<th><strong>Replication</strong></th>
<th>UPV 05</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Significance level</strong></td>
<td>0.05</td>
</tr>
<tr>
<td><strong>Interaction type</strong></td>
<td>Disordinal</td>
</tr>
<tr>
<td><strong>Relationship</strong></td>
<td>No definite pattern</td>
</tr>
</tbody>
</table>

**Profile plot**

![Profile plot diagram](image)

**Stock plot**

![Stock plot diagram](image)

**Comment**

- Structural technique exhibits a constant performance in all programs
- Functional and structural techniques behaved the same in the ntree program
- Functional technique behaved better than structural in nametbl
- Functional technique showed a low performance in cmdline

Table 6.23: Program-technique interaction for UPV 05
**Replication** | ORT 05  
---|---  
**Significance level** | 0.05  
**Interaction type** | Ordinal  
**Relationship** | $F = S$  

**Profile plot**

---

**Stock plot**

---

**Comment**
- Functional behave like the structural in ‘cmdline’
- Structural technique behave better in ‘nametbl’ than functional
- Both have a slightly better performance with ‘nametbl’ than ‘cmdline’

Table 6.24: Program-technique interaction for ORT 05
### 6.5.3.1 Fault

Fault is a main effect therefore the profile plot for all the experiments will be presented irrespective of whether it is significant at 99%, 95% confidence level in the experiment or not significant at all. The profile plot, mean values, relationship of faults and brief comments for each experiment as deducted from the plots is shown in Table 6.26 and Table 6.27.

Two of the experiments are significant for fault at 0.01 level of significance, these are UPM 04 and ORT 05, also two are significant at 0.05, UPV 05 and UdS 05 and the rest are not significant.

In Table 6.26 and Table 6.27, the experiments where fault was significant appeared in normal print while the non-significant ones appeared in grey.

No definite and consistent pattern was visible from the fault’s profile plots for all the experiments in Table 6.26 and Table 6.27. However, the facts below were deducted from the plots and cluster distribution of the mean values as representative of all the experiments.

- F3 maintains a high value in all the experiments. Actually, it was the fault with highest value in 6 out of the eight experiments. It shows a very good visibility tendency.
- Faults F1 and F6 maintained the same level of performance across all the experiments. These faults exhibit good tendency for visibility.
- F2, F5 and F4 showed fairly good tendency for visibility.
- F7 displayed the worst visibility tendency.

#### Table 6.25: Cluster analysis of overall program-technique interaction

<table>
<thead>
<tr>
<th>Technique performance (average of marginal means)</th>
<th>Cmdline</th>
<th>Nametbl</th>
<th>Ntree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very good (≥ 80)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Good (65 ≤ 79)</td>
<td>Functional</td>
<td>Structural</td>
<td>-</td>
</tr>
<tr>
<td>Poor (55 ≤ 64)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Very poor (≤ 54)</td>
<td>Review</td>
<td>Review</td>
<td>-</td>
</tr>
</tbody>
</table>

Fault is a main effect therefore the profile plot for all the experiments will be presented irrespective of whether it is significant at 99%, 95% confidence level in the experiment or not significant at all. The profile plot, mean values, relationship of faults and brief comments for each experiment as deducted from the plots is shown in Table 6.26 and Table 6.27.

Two of the experiments are significant for fault at 0.01 level of significance, these are UPM 04 and ORT 05, also two are significant at 0.05, UPV 05 and UdS 05 and the rest are not significant.

In Table 6.26 and Table 6.27, the experiments where fault was significant appeared in normal print while the non-significant ones appeared in grey.

No definite and consistent pattern was visible from the fault’s profile plots for all the experiments in Table 6.26 and Table 6.27. However, the facts below were deducted from the plots and cluster distribution of the mean values as representative of all the experiments.

- F3 maintains a high value in all the experiments. Actually, it was the fault with highest value in 6 out of the eight experiments. It shows a very good visibility tendency.
- Faults F1 and F6 maintained the same level of performance across all the experiments. These faults exhibit good tendency for visibility.
- F2, F5 and F4 showed fairly good tendency for visibility.
- F7 displayed the worst visibility tendency.
## Table 6.26: Profile plot for fault

<table>
<thead>
<tr>
<th>UPM 01</th>
<th>UPM 02</th>
<th>UPM 03</th>
<th>UPM 04</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Profile plot for UPM 01" /></td>
<td><img src="image2" alt="Profile plot for UPM 02" /></td>
<td><img src="image3" alt="Profile plot for UPM 03" /></td>
<td><img src="image4" alt="Profile plot for UPM 04" /></td>
</tr>
</tbody>
</table>

- **UPM 01**: Not Significant
  - F3 : 78.01
  - F1 : 76.49
  - F2 : 67.89
  - F5 : 67.63
  - F7 : 66.44
  - F6 : 66.40
  - F4 : 58.20
  - (F1=F2=F3=F5=F6=F7) > F4

- **UPM 02**: Not Significant
  - F3 : 75.72
  - F5 : 66.71
  - F1 : 65.77
  - F6 : 64.93
  - F4 : 64.85
  - F2 : 63.70
  - F7 : 57.02
  - (F1=F2=F3=F5=F6=F7) > F4

- **UPM 03**: Not Significant
  - F1 : 78.70
  - F6 : 75.00
  - F3 : 74.72
  - F7 : 71.48
  - F4 : 71.20
  - F2 : 69.82
  - F5 : 68.24
  - (F1=F2=F3=F5=F6) > (F4=F7)

- **UPM 04**: Significance level: 0.01
  - F2 : 82.41
  - F5 : 77.78
  - F1 : 77.01
  - F6 : 76.85
  - F3 : 75.93
  - F4 : 60.20
  - F7 : 58.33
  - (F1=F2=F3=F5=F6) > (F4=F7)

- The performance of F4 is slightly lower than other faults.
- **According to Scheffe analysis all faults behave the same; However, cluster analysis show f4 and f7 behave a little worse than others**

- **All faults behave the same except for F7 that exhibits a lower performance**
Chapter 6: Global View of the Experiments

Babatunde Kazeem

Table 6.27: Profile plot for fault
6.5.3.2 Fault - technique interaction

UPM 01, UPM 03, UPM 05 and UdS 05 are the four experiments that showed significant effect for fault-technique interaction. All the experiments are significant at 95% confidence level except for the UPM 01 experiment that is significant at 99%. The tables (Table 6.29 to 6.31) are structured and mean the same like similar tables in previous sections.

The plots display a bit of ordinal interaction so, some pattern of behavior as highlighted below can be discerned from it:

- In general, functional technique behaved same as the structural technique and both are better than the code review for all faults.
- There is variability in the performance of all the techniques based on fault.
- The code review technique display a tendency of matching the performance of the dynamic techniques with faults F1, F3 and F7 which are: omission, cosmetic; omission, initialization and commission, computation respectively.
- The code review exhibited a consistent worse performance behavior with F2, error of commission, cosmetic.
- The functional and structural techniques show a consistent high performance with F6, omission, control; and occasionally with F2 and F4 – commission, cosmetic and commission initialization respectively.
- The dynamic techniques exhibited good performance behavior for all fault types.
- The code review appears not good for the errors of commission, cosmetic - F2 and commission initialization - F4, but show good tendency with F5 – commission, control.

The average behavior characterization is presented in Table 6.32, using the approach described in Section 6.5.2.2.
Replication  | UPM 01  
Significance level | 0.01  
Interaction type | Ordinal  
Relationship | (F = S = R): F1; F > (S = R): F3 and F7 (F = S) > R: F2, F4, F5 and F6  

**Profile plot**

![Profile plot](image)

**Stock plot**

![Stock plot](image)

**Comment**

- The techniques, most especially code review displayed varied behavior for different faults
- All techniques behave the same for F1
- Structural technique and code review behave identical for F3 and F7
- Functional and structural behave the same and better than code review with F2, F4, F5 and F6
- Structural and functional techniques behaved best with F2
- Code review recorded peak performance with F1, F3 and F7
- Code review behaved worst with F2
- Wide variability is noticed in code review

Table 6.28: UPM 01 fault-technique interaction
### Replication
<table>
<thead>
<tr>
<th>Technique</th>
<th>Functional</th>
<th>Structural</th>
<th>Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>88.7</td>
<td>88.6</td>
<td>122.0</td>
</tr>
<tr>
<td>f2</td>
<td>88.3</td>
<td>88.9</td>
<td>122.0</td>
</tr>
<tr>
<td>f3</td>
<td>88.6</td>
<td>88.0</td>
<td>122.0</td>
</tr>
<tr>
<td>f4</td>
<td>88.0</td>
<td>88.3</td>
<td>122.0</td>
</tr>
<tr>
<td>f5</td>
<td>88.3</td>
<td>88.6</td>
<td>122.0</td>
</tr>
<tr>
<td>f6</td>
<td>88.6</td>
<td>88.0</td>
<td>122.0</td>
</tr>
<tr>
<td>f7</td>
<td>88.0</td>
<td>88.3</td>
<td>122.0</td>
</tr>
</tbody>
</table>

### Comment
- Performance of the techniques is influenced by fault type, most especially code review.
- Functional and structural behave the same for all faults and is always better than code review.
- Structural and functional techniques have highest values with F2.
- Code review has highest value with F1 and recorded good values with F3 and F7.
- Code review behave worst with F2.
- Code review shows a wide variability based on faults.

Table 6.29: UPM 03 fault-technique interaction
### Chapter 6: Global View of the Experiments

<table>
<thead>
<tr>
<th>Replication</th>
<th>UPM 05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Significance level</td>
<td>0.05</td>
</tr>
<tr>
<td>Interaction type</td>
<td>Disordinal</td>
</tr>
<tr>
<td>Relationship</td>
<td>(F = S) &gt; R</td>
</tr>
</tbody>
</table>

#### Profile plot

![Profile plot](image)

#### Stock plot

![Stock plot](image)

#### Comment

- All the techniques showed wide variability in performance
- All techniques behaved the same for F3 and F5
- Functional and structural behave the same for F1, F2 and F6
- Functional technique and code review behaved identical for F7
- Functional technique have highest performance with F4
- Structural technique have highest performance with F2
- Code review have highest performance with F5 and good value with F1 and F7
- Functional and structural behave worst with F3
- Code review behaved worst with F2 and F4

Table 6.30: UPM 05 fault-technique interaction
Replication | US 05  
Significance level | 0.05  
Interaction type | Disordinal  
Relationship | \((F = S) > R\)  

### Profile plot

![Profile plot](image)

### Stock plot

![Stock plot](image)

### Comment
- The performance of the techniques is influenced by fault type
- All techniques behaved identically with F7
- Functional and structural techniques behaved identical for F2, F3, F4, F5 and F6
- Functional technique and code review behaved identical for F1
- Functional technique had best performance with F3
- Structural technique had best performance with F6
- Code review behave best with F7
- Functional technique recorded low performance with F1
- Code review behave worst with F2

Table 6.31: UdS 05 fault-technique interaction
### Chapter 6: Global View of the Experiments

#### 6.5.4 Overview of the joint analysis

As mentioned earlier, the focus of this research is mainly to analyze the effectiveness of the evaluation techniques, based on the result of the eight experiments under study. Effectiveness, in this case is measured as the ability of the dynamic techniques to generate test cases or the code review to generate abstraction that expose the faults.

So far, we have been able to see that the effectiveness of the three techniques is influenced by the program and the fault type. The techniques behaved differently in each of the programs, the same thing happened to the faults as well. For example, all faults of omission displayed the tendency of being easily detectable in all the programs unlike the faults of commission that displayed varied behavior depending on the program except for F2: commission, cosmetic that is good for all programs. The fault of commission control – F5 behaved very well in the data programs and poorly in the functional program. F4 behaved good in ntree and average in both cmdline and nametbl while F7 behaved poorly in ntree but good in cmdline and nametbl; both being errors of commission. The initialization error of omission is easier to detect in all programs while that of commission only behave well in the ntree. Control errors are probably easily exposed in data programs as F5 and F6 behaved well in nametbl and ntree while only control omission behaved well in cmdline. **There is a fairly strong indication that the program type in some way affects the detection of faults.**

The functional technique exhibited very good (according to the average performance classification terms used in the research) performance behavior in ‘cmdline’ and ‘nametbl’ but good in ‘ntree’; while the structural technique is only very good with ‘nametbl’ and good with ‘cmdline and ‘ntree’. The code review is very poor with ‘cmdline’ and ‘nametbl’ and poor with ‘ntree’. Here it is apparent that the dynamic techniques will behave effectively enough in any program. though, we cannot still explain why the structural technique appears to behave similar in cmdline and ntree and the functional technique seems to do the same between cmdline and nametbl. Such behavior would

---

<table>
<thead>
<tr>
<th>Fault detection (average of marginal means)</th>
<th>Functional</th>
<th>Structural</th>
<th>Code review</th>
</tr>
</thead>
<tbody>
<tr>
<td>High (≥ 75)</td>
<td>F2, F3, F4, F5, F6</td>
<td>F1, F2, F3, F4, F5, F6</td>
<td></td>
</tr>
<tr>
<td>Fairly high (60 ≤ 74)</td>
<td>F1, F7</td>
<td>F7</td>
<td>F1, F7</td>
</tr>
<tr>
<td>Low (50 ≤ 59)</td>
<td></td>
<td></td>
<td>F3, F5</td>
</tr>
<tr>
<td>Very low (≤49)</td>
<td></td>
<td></td>
<td>F2, F4, F6</td>
</tr>
</tbody>
</table>

Table 6.32: Cluster behavior of the fault-technique interaction

Babatunde Kazeem

123
have been expected between nametbl and ntree. Though, the margin in performance is negligible and the problem might also be due to our classification style.

Also, it is interesting to note the pattern in fault behaviors with the techniques – some behave better than others – which is the case with F7 error of commission computation that behaved worse than other faults with the dynamic techniques and usually better than others with the code review. Likewise, the code review behaved poorly with F2 – commission, cosmetic. Nevertheless, the code review seems to work well with errors of omission as it displayed a tendency of matching the performance of the dynamic techniques with F1 and F3 except F6, all errors of omission. However, converse was the case with errors of commission F2, F4 and F5 except F7. Averagely, the dynamic techniques seem capable of exposing all faults well better than code review.

It is unclear whether the classification of the programs played any significant role in these experiments. One would have expected either a fault or a technique to at least behave identically for programs of the same type. This was not always the case. For example, F4 and F7 behaved identical with ‘cmdline’ and ‘nametbl’ (rather than ‘nametbl’ and ‘ntree’) noting that the two are functional program and data program respectively. It might be interesting to ask why it does not behave in the same way in ‘ntree’ which a data program like ‘nametbl’.

The same thing applies to the techniques in the programs. In some cases, close similarity in the behavior of two techniques is found between a data program and a functional program. The expectation would have been that the techniques behave the in the two functional programs (nametbl and ntree), either similar or different from the data program (cmdline).

Summarily, either with the programs or with the faults, it can be concluded from the synthesis that the functional and structural techniques behaved better than the code review. They both displayed stronger tendency or capability of being able to expose all faults in all programs better than the code review.

The two techniques put up identical effectiveness performance for all the faults, except for F1 where the structural technique behaved slightly better than the functional technique. So, both techniques are equally effective and better than the code review. This conclusion contradicts the findings of earlier experiments but it is consistent throughout the eight experiments used for this study.

Of course, further research is required to further establish the facts found in these experiments. It will be necessary to refine the design of the experiments. It may be desirable to re-classify the faults and/or use new set of programs. Furthermore, to raise the expertise of the subjects and
eliminate any imbalance in the homogeneity of the subjects, three or more preparatory exercises may be introduced prior to the experiment. This might help stabilize performance and eliminate learning during the experiment. More importantly, it will ensure homogeneous subject. Doing this, will probably raise the confidence level of any significant effects and more especially create interaction profiles that are ordinal in nature. This will boost the reliability of the results.
Summarizing the Results of a Series of Experiments: Application to Effectiveness of Software Evaluation Techniques

Babatunde Kazeem
Chapter 7

Conclusion and Recommendation

7.1 Conclusions

The main purpose of this research, as presented in this thesis work, was to create a joint analysis summary for a series of experiments conducted to investigate similar hypothesis – weather fault detection technique, fault type and program have any impact on the techniques’ effectiveness or efficiency or not; and whether the failures generated by faults have any impact on its visibility. The experiments tagged Experiment II, used two versions of three different programs with seven faults seeded into each to investigate these facts.

The eight experiments were conducted between 2001 and 2005, five at UPM in 2001 through 2005 and one each at ORD, UdS and UPV in 2005. Due to the difference in time and location of the conduct of the experiments, they were conducted by different researchers. This created a significant difference in the format and style of the data and analysis files of the experiment which apparently was an initial obstacle to the synthesis process.

In order to achieve aims of this research, the first step of the thesis was the simulation of the experiments for improved familiarization and increased understanding of the experiment situation and data collected. Afterwards, we have decided to create a structured directory for the experiments’ files similar for all experiments and also store all files on a single location. Furthermore, we re-structured all data files for the experiments to conform to a similar language, structure and data fields naming as much as possible, in order to facilitate comparability and help future joint interpretation effort. All the excel files now consists of four sheets named: Subject data, Observable faults, Failure visibility and Observed faults, except for the ORT 05 which does not have the failure visibility and Observed faults sheet because test case execution did not take place in the experiment and UdS 05 which does not have failure visibility sheet because of the test case generation and execution was done by different set of subject pairs. A correspondence table
was also created to maintain traceability between the original files and the newly created ones in case there is any future need for such.

Then, using the approach we created, to systematically summarize the results, a joint ANOVA table was created from the individual analysis of the different experiments. For fear of loss of knowledge, the confidence interval level for significance was maintained at both 99% and 95% with 99% held as presenting more reliable evidence than 95%. This idea was used to create a significance classification for the effects, significant, significant tendency, unclear and not significant. The technique, adjudged as being significant, the program, the program – technique and the program – fault interactions adjudged as having significance tendency and fault and fault – technique interaction categorized as having ambiguous or unclear significance were selected as the treatments relevant to the summary result. Other treatments were categorized as not significant.

Eventually, it was overall discovered that, the structural and functional technique behaved identically; which means, in the context of the programs and fault types used for the experiments none of the two can be said to be more effective than the other in terms of producing test cases that could expose the faults. Both behaved better than the code review. The code review behaved poorly for all faults and appears to be worst for faults of commission and worse for omission faults.

7.2 Further research

This research work opens the way for several other research questions that need to be answered and consequently, more research areas in need of further investigation. In our opinion, the effectiveness of the three techniques is a subject that is not supposed to be closed based on the evidence of this joint summary. Further investigation is desired to further verify the finding from this research. More importantly, to strengthen the weaknesses in the strength of the analysis used for this summary. For example, most of the effects used for our deduction purpose fell into the significance tendency or unclear category.

The following is a list of suggested future improvements:

- **Improved experiment design:** Looking back at the analysis on which the conclusions of this study were based, all the interaction effects fell under either significant tendency or unclear status. The profile plots were also mostly disordinal. A refinement of the design of the experiments may eliminate some of the unstable factors and improve the reliability of
the evidence produced by the experiments. Some design improvement suggestions have been given in Section 6.5.4

- **Improved deduction process**: The deduction process we created and used for this research could not still be said to be at its best. The steps of the process could still be refined or redefined to be more consistent and stable irrespective/independent of the analysis. This affects:
  - Ranking and classification: These two steps are important to the eventual result of the deduction process. Therefore it is important to work more on the process of assigning class and ranks and what will be the meaning/implication of the ranks.
  - Consistency: The deduction process we introduced still needs to be investigated for consistency in output to ascertain its credibility.

- **Extensive analysis**: In this work, we have only presented the summary analysis of the experiments. Further investigation, using a ‘white box’ approach to the summary will be interesting. To search what factors are responsible or causes of some results or anomalies. Such study will present a more informed summary.

- **Formal aggregation**: Another line of research is to use formal aggregation method in place of this type of qualitative method and compare results. The result of this process will go a long way to either reinforce or refute the findings of this research and the reliability of the method used.
References


## Appendix A

### Structure of the Original Excel Files

<table>
<thead>
<tr>
<th>SHEET</th>
<th>EXPLANATION</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject Data</td>
<td>The sheet containing comprehensive information about the subjects and their performance activities during the experiment.</td>
<td>UPM 01  UPM 02 UPM 03 UPM 04 UPM 05</td>
</tr>
<tr>
<td>Datos Sujeto</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technique Application</td>
<td>This sheet contains experiment data indicating whether a subjects is able to generate test cases for the faults seeded into the program</td>
<td></td>
</tr>
<tr>
<td>Datos tecnica</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure visibility</td>
<td>This sheet is a record of data that indicates whether the test cases recorded in the technique application sheet actually reveal the corresponding fault</td>
<td></td>
</tr>
<tr>
<td>Datos Fallos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ejecucion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Casos</td>
<td>A combination of data in the subject data and technique application sheets as defined above</td>
<td></td>
</tr>
<tr>
<td>Datos</td>
<td>A combination of data in the subject data, failure visibility and technique application sheets as defined above</td>
<td></td>
</tr>
<tr>
<td>Todos</td>
<td>The sheet containing all the possible data recorded during the experiment</td>
<td></td>
</tr>
</tbody>
</table>

Table A. 1 List, location and explanation of old excel worksheets
<table>
<thead>
<tr>
<th>Field name</th>
<th>Explanation</th>
<th>Sheet location</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1 – F7</td>
<td>These fields are used to indicate whether a subject was able to generate a test case that is capable of revealing a corresponding fault $F_i$</td>
<td>X</td>
</tr>
<tr>
<td>F1 – F7</td>
<td>These fields are used to report whether the test case generated for a particular fault did actually expose the fault</td>
<td>X</td>
</tr>
<tr>
<td>Subject/ Nombre/ Alumno</td>
<td>The name of the experimental subject</td>
<td>X X X X X X X</td>
</tr>
<tr>
<td>Program/programa</td>
<td>Program used for experiment</td>
<td>X X X X X X X</td>
</tr>
<tr>
<td>Technique/tecnica</td>
<td>The technique applied by the subject</td>
<td>X X X X X X X</td>
</tr>
<tr>
<td>Version/version</td>
<td>The version of the program used by the subject</td>
<td>X X X X X X X</td>
</tr>
<tr>
<td>Time 1/tgen/tiempo</td>
<td>Time spent to apply the technique</td>
<td>X</td>
</tr>
<tr>
<td>Time 2/tiempo ej./Tteclear</td>
<td>Time spent to execute the test cases</td>
<td>X</td>
</tr>
<tr>
<td>Time 3/Tfallos /tiempo id.</td>
<td>Time spent to detect either the expected faults or failures</td>
<td>X</td>
</tr>
<tr>
<td>Rel. ex./Ex. Rel./ Experiencia relative/ ExpRel</td>
<td>Relative experience of the subject</td>
<td>X</td>
</tr>
<tr>
<td>Rel. abs./ Ex. Abs./ Experiencia absoluta/ExpAbs</td>
<td>Absolute experience of the subject</td>
<td>X</td>
</tr>
<tr>
<td>Motivation</td>
<td>The subjective grading of the subject’s motivation</td>
<td>X</td>
</tr>
<tr>
<td>“N Abs/classes”/” N° clases / camino”/ Clases</td>
<td>Number of abstractions or equivalent classes generated for the code reading or the functional technique respectively.</td>
<td>X X X</td>
</tr>
</tbody>
</table>

Table A. 2: List of data fields and corresponding sheets where they were located
### Field name | Explanation | Sheet location
--- | --- | ---
N cases/ Nº casos/ Casos | Number of test cases generated for the structural technique | X | X | X
% defect | The percentage ratio of | X
Confidence/confianza/auto eval./aplicacion | A subjective grade of how confident the subject is on the solutions provided to the tasks given | X | X | X
Total/visto | Accumulation of corresponding F1 to F7 entries | X | X | X
Percentaje | The percentage ratio of test cases generated or failures detected with what was expected | X | X
Grupo/ asignacion | The group in which subjects work | X | X | X | X | X
Carrera/ pareja | Group pair numbers | X | X
Tur2o |  | 
No. estudiante | Subject’s matriculation number |  
Orden |  | 
Tiempo | Sum of time2 and time3 as defined above | X
Calificacion |  | 
Detectables | Maximum number of failures detectable by the subject | X
Proc/ Proced. |  | X
PorEst |  | X
visible | The maximum number of failures detectable by the subject in a particular program | X | X
Clase | An indication whether the subject attends the class where training on the experiment was given | X
Maximo | The maximum number of faults detectable | X

Table A. 3: Continued list of data fields and corresponding sheets where they were located
Appendix B

Mapping between New and Original Excel Files

B.1 Subject data sheet

<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPM 01</td>
<td>UPM 02</td>
</tr>
<tr>
<td>UPM 03</td>
<td>UPM 04</td>
</tr>
<tr>
<td>UPM 05</td>
<td>UPV 05</td>
</tr>
<tr>
<td>ORT 05</td>
<td>UdS 05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>Old sheet name: subject data Old field name: Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Old sheet name: subject data Old field name: Program</td>
</tr>
<tr>
<td>Technique</td>
<td>Old sheet name: subject data Old field name: technique</td>
</tr>
<tr>
<td>Version</td>
<td>Old sheet name: subject data Old field name: Version</td>
</tr>
</tbody>
</table>

*Table B.1: Data fields of 'subject data' sheet*
<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative experience</td>
<td>Old sheet name: subject data Old field name: Rel. Ex</td>
</tr>
<tr>
<td>Absolute experience</td>
<td>Old sheet name: subject data Old field name: Rel. Abs</td>
</tr>
<tr>
<td>Motivation</td>
<td>Old sheet name: subject data Old field name: motivation</td>
</tr>
<tr>
<td>Technique application time</td>
<td>Old sheet name: subject data Old field name: time 1</td>
</tr>
<tr>
<td>Test case execution time</td>
<td>Old sheet name: subject data Old field name: time 2</td>
</tr>
</tbody>
</table>

Table B. 2: Data fields of 'subject data' sheet (contd.)

<table>
<thead>
<tr>
<th>UPM 01</th>
<th>UPM 02</th>
<th>UPM 03</th>
<th>UPM 04</th>
<th>UPM 05</th>
<th>UPV 05</th>
<th>ORT 05</th>
<th>UdS 05</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative experience</td>
<td>Old sheet name: subject data Old field name: Rel. Ex</td>
<td>Old sheet name: subject data Old field name: Rel. Ex</td>
<td>Old sheet name: subject data Old field name: Rel. Ex</td>
<td>Old sheet name: Datos sujetos Old field name: Ex. Rel.</td>
<td>Old sheet name: Datos sujetos Old field name: Ex. Rel.</td>
<td>Old sheet name: Datos</td>
<td>Old field name: ExRel</td>
</tr>
<tr>
<td>Absolute experience</td>
<td>Old sheet name: subject data Old field name: Rel. Abs</td>
<td>Old sheet name: subject data Old field name: Rel. Abs</td>
<td>Old sheet name: subject data Old field name: Rel. Abs</td>
<td>Old sheet name: Datos sujetos Old field name: Ex. Abs</td>
<td>Old sheet name: Datos sujetos Old field name: Ex. Abs</td>
<td>Old sheet name: Datos</td>
<td>Old field name: ExAbs</td>
</tr>
<tr>
<td>Motivation</td>
<td>Old sheet name: subject data Old field name: motivation</td>
<td>Old sheet name: subject data Old field name: motivation</td>
<td>Old sheet name: subject data Old field name: motivation</td>
<td>Old sheet name: Datos sujeto Old field name: tiempo 1</td>
<td>Old sheet name: Datos sujeto Old field name: tiempo 1</td>
<td>Old sheet name: N/A</td>
<td>Old field name: N/A</td>
</tr>
<tr>
<td>Technique application time</td>
<td>Old sheet name: subject data Old field name: time 1</td>
<td>Old sheet name: subject data Old field name: time 1</td>
<td>Old sheet name: subject data Old field name: time 1</td>
<td>Old sheet name: Datos sujetos Old field name: tiempo 2</td>
<td>Old sheet name: Datos sujetos Old field name: tiempo 2</td>
<td>Old sheet name: N/A</td>
<td>Old field name: N/A</td>
</tr>
<tr>
<td>Test case execution time</td>
<td>Old sheet name: subject data Old field name: time 2</td>
<td>Old sheet name: subject data Old field name: time 2</td>
<td>Old sheet name: subject data Old field name: time 2</td>
<td>Old sheet name: Datos sujetos Old field name: Tiempo 2</td>
<td>Old sheet name: Datos sujetos Old field name: Tiempo ej</td>
<td>Old sheet name: N/A</td>
<td>Old field name: N/A</td>
</tr>
</tbody>
</table>

Babatunde Kazeem
### Table B. 3: Data fields of 'subject data' sheet (contd.)

<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fault/failure detection time</strong></td>
<td>UPM 01</td>
</tr>
<tr>
<td></td>
<td>Old sheet name: subject data</td>
</tr>
<tr>
<td></td>
<td>Old field name: time 3</td>
</tr>
<tr>
<td><strong>Number of abstraction</strong></td>
<td>Old sheet name: subject data</td>
</tr>
<tr>
<td></td>
<td>Old field name: N Abstraction /classes</td>
</tr>
<tr>
<td><strong>Number of classes</strong></td>
<td>Old sheet name: subject data</td>
</tr>
<tr>
<td></td>
<td>Old field name: N Abstraction /classes</td>
</tr>
<tr>
<td><strong>Number of test cases</strong></td>
<td>Old sheet name: subject data</td>
</tr>
<tr>
<td></td>
<td>Old field name: N cases</td>
</tr>
<tr>
<td><strong>Percentage estimated defects</strong></td>
<td>Old sheet name: subject data</td>
</tr>
<tr>
<td></td>
<td>Old field name: % defect</td>
</tr>
<tr>
<td>New field name</td>
<td>Previous Location</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td><strong>Confidence</strong></td>
<td>Old sheet name: subject data Old field name: confidence</td>
</tr>
<tr>
<td><strong>Origin</strong></td>
<td>Old sheet name: N/A Old field name: N/A</td>
</tr>
<tr>
<td><strong>Group</strong></td>
<td>Old sheet name: N/A Old field name: N/A</td>
</tr>
<tr>
<td><strong>Pair</strong></td>
<td>Old sheet name: N/A Old field name: N/A</td>
</tr>
<tr>
<td><strong>Execution + Id time</strong></td>
<td>Old sheet name: N/A Old field name: N/A</td>
</tr>
<tr>
<td><strong>Qualification</strong></td>
<td>Old sheet name: N/A Old field name: N/A</td>
</tr>
</tbody>
</table>

Table B. 4: Data fields of ‘subject data’ sheet (contd.)
## B.2 Observable faults sheet

<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
<th>UPM 01</th>
<th>UPM 02</th>
<th>UPM 03</th>
<th>UPM 04</th>
<th>UPM 05</th>
<th>UPV 05</th>
<th>ORT 05</th>
<th>UdS 05</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subject</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old sheet name:</td>
<td>Technique influence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old field name:</td>
<td>Subject</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Program</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old sheet name:</td>
<td>Technique influence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old field name:</td>
<td>Program</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Technique</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old sheet name:</td>
<td>Technique influence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old field name:</td>
<td>Technique</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old sheet name:</td>
<td>Technique influence</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Old field name:</td>
<td>Version</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table B. 5: Data fields in ‘observable faults’ sheet
<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UPM 01</td>
</tr>
<tr>
<td>F1 … F7</td>
<td>Old sheet name: Technique influence Old field name: F1… F7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>Old sheet name: Technique influence Old field name: Total</td>
</tr>
<tr>
<td><strong>Percentage</strong></td>
<td>Old sheet name: N/A Old field name: N/A</td>
</tr>
<tr>
<td><strong>Attendance</strong></td>
<td>Old sheet name: N/A Old field name: N/A</td>
</tr>
<tr>
<td><strong>Origin</strong></td>
<td>Old sheet name: N/A Old field name: N/A</td>
</tr>
</tbody>
</table>

Table B. 6: Data fields in ‘observable faults’ sheet (contd)
<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UPM 01</strong></td>
<td><strong>UPM 02</strong></td>
</tr>
<tr>
<td>Old sheet name: N/A</td>
<td>Old sheet name: N/A</td>
</tr>
<tr>
<td>Old field name: N/A</td>
<td>Old field name: N/A</td>
</tr>
<tr>
<td><strong>UPM 03</strong></td>
<td><strong>UPM 04</strong></td>
</tr>
<tr>
<td>Old sheet name: N/A</td>
<td>Old field name: N/A</td>
</tr>
<tr>
<td>Old field name: N/A</td>
<td>Old field name: N/A</td>
</tr>
<tr>
<td><strong>UPM 05</strong></td>
<td><strong>UPV 05</strong></td>
</tr>
<tr>
<td>Old sheet name: Datos Tecnica</td>
<td>Old field name: Grupo</td>
</tr>
<tr>
<td>Old field name: Grupo</td>
<td></td>
</tr>
<tr>
<td><strong>ORT 05</strong></td>
<td><strong>UdS 05</strong></td>
</tr>
<tr>
<td>Old sheet name: Datos</td>
<td>Old field name: Asignacion</td>
</tr>
<tr>
<td>Old field name: Asignacion</td>
<td></td>
</tr>
</tbody>
</table>

Table B. 7: Data fields in ‘observable faults’ sheet (contd)
## B.3 Failure Visibility Sheet

<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subject</strong></td>
<td></td>
</tr>
<tr>
<td>Old sheet name: Failure Visibility</td>
<td>Old sheet name: Subject</td>
</tr>
<tr>
<td>Old field name: Subject</td>
<td>Old field name: Subject</td>
</tr>
<tr>
<td><strong>Program</strong></td>
<td></td>
</tr>
<tr>
<td>Old sheet name: Failure Visibility</td>
<td>Old sheet name: Program</td>
</tr>
<tr>
<td>Old field name: Program</td>
<td>Old field name: Program</td>
</tr>
<tr>
<td><strong>Technique</strong></td>
<td></td>
</tr>
<tr>
<td>Old sheet name: Failure Visibility</td>
<td>Old sheet name: Technique</td>
</tr>
<tr>
<td>Old field name: Technique</td>
<td>Old field name: Technique</td>
</tr>
<tr>
<td><strong>Version</strong></td>
<td></td>
</tr>
<tr>
<td>Old sheet name: Failure Visibility</td>
<td>Old sheet name: Version</td>
</tr>
<tr>
<td>Old field name: Version</td>
<td>Old field name: Version</td>
</tr>
</tbody>
</table>

Table B. 8: Data fields in ‘failure visibility’ sheet
<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1, ..., F7</td>
<td>Old sheet name: Failure Visibility Old field name: F1, ..., F7</td>
</tr>
<tr>
<td>Total</td>
<td>Old sheet name: Failure Visibility Old field name: Total</td>
</tr>
<tr>
<td>Percentage</td>
<td>Old sheet name: Datos Fallos Old field name: Porcentaje</td>
</tr>
<tr>
<td>Group</td>
<td>Old sheet name: Datos Fallos Old field name: Grupo</td>
</tr>
<tr>
<td>Visible</td>
<td>Old sheet name: Datos Fallos Old field name: Visible</td>
</tr>
</tbody>
</table>

Table B. 9: Data fields in ‘failure visibility’ sheet (contd)
### B.4 Observed Failure Sheet

<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>UPM 01</td>
</tr>
<tr>
<td></td>
<td>Old sheet name:</td>
</tr>
<tr>
<td></td>
<td>Failure Visibility,</td>
</tr>
<tr>
<td></td>
<td>Technique</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Old field name:</td>
</tr>
<tr>
<td></td>
<td>Subject</td>
</tr>
<tr>
<td></td>
<td>UPM 02</td>
</tr>
<tr>
<td></td>
<td>Old sheet name:</td>
</tr>
<tr>
<td></td>
<td>Failure Visibility,</td>
</tr>
<tr>
<td></td>
<td>Technique</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Old field name:</td>
</tr>
<tr>
<td></td>
<td>Subject</td>
</tr>
<tr>
<td></td>
<td>UPM 03</td>
</tr>
<tr>
<td></td>
<td>Old sheet name:</td>
</tr>
<tr>
<td></td>
<td>Failure Visibility,</td>
</tr>
<tr>
<td></td>
<td>Technique</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Old field name:</td>
</tr>
<tr>
<td></td>
<td>Subject</td>
</tr>
<tr>
<td></td>
<td>UPM 04</td>
</tr>
<tr>
<td></td>
<td>Old sheet name:</td>
</tr>
<tr>
<td></td>
<td>Failure Visibility,</td>
</tr>
<tr>
<td></td>
<td>Technique</td>
</tr>
<tr>
<td></td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>Old field name:</td>
</tr>
<tr>
<td></td>
<td>Subject</td>
</tr>
<tr>
<td></td>
<td>UPM 05</td>
</tr>
<tr>
<td></td>
<td>Old sheet name:</td>
</tr>
<tr>
<td></td>
<td>Datos Fallos,</td>
</tr>
<tr>
<td></td>
<td>Datos Tecnicas</td>
</tr>
<tr>
<td></td>
<td>Old field name:</td>
</tr>
<tr>
<td></td>
<td>Alumni</td>
</tr>
<tr>
<td></td>
<td>UPV 05</td>
</tr>
<tr>
<td></td>
<td>Old sheet name:</td>
</tr>
<tr>
<td></td>
<td>Datos Fallos,</td>
</tr>
<tr>
<td></td>
<td>Datos Tecnicas</td>
</tr>
<tr>
<td></td>
<td>Old field name:</td>
</tr>
<tr>
<td></td>
<td>Alumni</td>
</tr>
<tr>
<td></td>
<td>ORT 05</td>
</tr>
<tr>
<td></td>
<td>Old sheet name:</td>
</tr>
<tr>
<td></td>
<td>Datos</td>
</tr>
<tr>
<td></td>
<td>field name:</td>
</tr>
<tr>
<td></td>
<td>Nombre</td>
</tr>
<tr>
<td></td>
<td>UdS 05</td>
</tr>
<tr>
<td></td>
<td>Old sheet name:</td>
</tr>
<tr>
<td></td>
<td>Ejecucion</td>
</tr>
<tr>
<td></td>
<td>Old field name:</td>
</tr>
<tr>
<td></td>
<td>Nombre</td>
</tr>
</tbody>
</table>

| Program        | UPM 01            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Program           |
|                | UPM 02            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Program           |
|                | UPM 03            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Program           |
|                | UPM 04            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Program           |
|                | UPM 05            |
|                | Old sheet name:   |
|                | Datos Fallos,     |
|                | Datos Tecnicas    |
|                | Old field name:   |
|                | Program           |
|                | UPV 05            |
|                | Old sheet name:   |
|                | Datos Fallos,     |
|                | Datos Tecnicas    |
|                | Old field name:   |
|                | Program           |
|                | ORT 05            |
|                | Old sheet name:   |
|                | Datos             |
|                | field name:       |
|                | Nombre            |
|                | UdS 05            |
|                | Old sheet name:   |
|                | Ejecucion         |
|                | Old field name:   |
|                | Nombre            |

| Technique      | UPM 01            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Technique         |
|                | UPM 02            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Technique         |
|                | UPM 03            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Technique         |
|                | UPM 04            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Technique         |
|                | UPM 05            |
|                | Old sheet name:   |
|                | Datos Fallos,     |
|                | Datos Tecnicas    |
|                | Old field name:   |
|                | Tecnica           |
|                | UPV 05            |
|                | Old sheet name:   |
|                | Datos Fallos,     |
|                | Datos Tecnicas    |
|                | Old field name:   |
|                | Tecnica           |
|                | ORT 05            |
|                | Old sheet name:   |
|                | Datos             |
|                | field name:       |
|                | N/A               |
|                | UdS 05            |
|                | Old sheet name:   |
|                | Ejecucion         |
|                | Old field name:   |
|                | N/A               |

| Version        | UPM 01            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Version           |
|                | UPM 02            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Version           |
|                | UPM 03            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Version           |
|                | UPM 04            |
|                | Old sheet name:   |
|                | Failure Visibility,|
|                | Technique         |
|                | Application       |
|                | Old field name:   |
|                | Version           |
|                | UPM 05            |
|                | Old sheet name:   |
|                | Datos Fallos,     |
|                | Datos Tecnicas    |
|                | Old field name:   |
|                | Version           |
|                | UPV 05            |
|                | Old sheet name:   |
|                | Datos Fallos,     |
|                | Datos Tecnicas    |
|                | Old field name:   |
|                | Version           |
|                | ORT 05            |
|                | Old sheet name:   |
|                | Datos             |
|                | field name:       |
|                | Version           |
|                | UdS 05            |
|                | Old sheet name:   |
|                | Ejecucion         |
|                | Old field name:   |
|                | Version           |

Table B.10: Data fields in the ‘observed faults’ sheet
<table>
<thead>
<tr>
<th>New field name</th>
<th>Previous Location</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UPM 01</td>
</tr>
<tr>
<td><strong>F1 .... F7</strong></td>
<td>Old sheet name: N/A</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>Old sheet name: N/A</td>
</tr>
<tr>
<td><strong>Percentage</strong></td>
<td>Old sheet name: N/A</td>
</tr>
<tr>
<td><strong>Visible</strong></td>
<td>Old sheet name: N/A</td>
</tr>
<tr>
<td><strong>Group</strong></td>
<td>Old sheet name: N/A</td>
</tr>
</tbody>
</table>

Table B. 11: Data fields in the ‘observed faults’ sheet (contd)