A Decision Support Framework for Software Test Managers

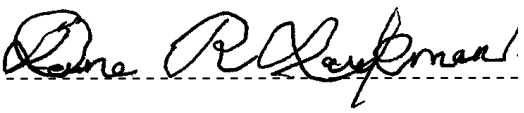
Deane Robert Larkman

Submitted in partial fulfilment of the requirements
for the degree of
Doctor of Information Technology
University of Canberra

February 2012

Copyright

This thesis may be freely copied and distributed for private use and study. However, no part of this thesis, or the information contained therein, may be included in or referred to in a publication without prior written permission of the author. Any references must be fully acknowledged.

Signature: 

Date: 10/2/2012

© 2012 Deane Robert Larkman

To my wife Vicki

To get through the hardest journey we need take only one step at a time, but we must keep on stepping – Chinese proverb (ThinkExist.com, 2011, para. 4)

It is good to have an end to journey toward; but it is the journey that matters, in the end – Ursula K. LeGuin (ThinkExist.com, 2011, para. 5)

When you have completed 95 percent of your journey, you are only halfway there – Japanese proverb (ThinkExist.com, 2011, para. 8)

ABSTRACT

Software is an indispensable part of our modern lifestyle. Users of software in business and society rely on software to support the tasks they need to, or want to achieve. Unfortunately, software fails, and the human and economic consequences can be devastating. High quality software can help minimise the consequences of software failures. Software testing is one way to improve software quality, and thereby provide high quality software. A comprehensive review of the literature has shown that, within the parameters of the boundary and focus of this research, support for test managers' decision making in software testing is scarce. In particular, research addressing decision support for the software test manager is especially lacking in planning and risk management of successful software testing.

This research investigates the need for a decision support framework for software test managers: with a focus on the planning and risk management done by software test managers for successful software testing. The research findings developed during this project are based on concepts, issues and ideas, grounded in evidence, reasons, and experience. The major contribution of this research is the development and application of a decision support framework. Software test managers can use the decision support framework in assisting them to plan and provide risk management for successful software testing.

The research contribution falls into two parts. The first part is the design and development of a decision support framework, which is described and discussed. The components of the framework are detailed, and the topology of those components is illustrated graphically. The second part of the research contribution is the application of the decision support framework by software test managers, establishing a basis for risk management. Application of the decision support framework is a two-step procedure. The software test manager first assigns relevant input to the framework, and this input allows the test manager to use the framework to model their particular software testing situation, set in the context of their organisation. Next, the software test manager using one or more analytical techniques, developed from two different evaluation perspectives, evaluates the model built in the first application step. The analytical techniques are presented and their potential use by software test managers is illustrated.

In summary, this research has produced a decision support framework, a practical and useful tool for the software test manager to plan for and manage the risk in successful software testing. It is envisaged that software test managers can apply the decision support framework to their specific organisational and software testing circumstances.

CERTIFICATE OF AUTHORSHIP OF THESIS

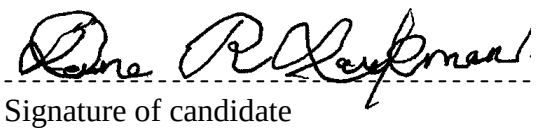
Except where clearly acknowledged in footnotes, quotations and the bibliography, I certify that I am the sole author of the thesis submitted today entitled –

A Decision Support Framework for Software Test Managers

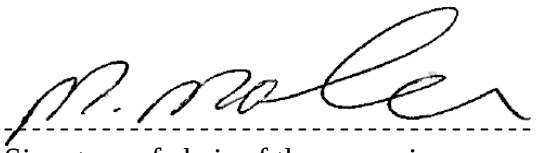
I further certify that to the best of my knowledge the thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

The material in the thesis has not been the basis of an award of any other degree or diploma except where due reference is made in the text of the thesis.

The thesis complies with University requirements for a thesis as set out in *Gold Book Part 7: Examination of Higher Degree by Research Theses Policy, Schedule Two (S2).* Refer to http://www.canberra.edu.au/research-students/goldbook



Signature of candidate



Signature of chair of the supervisory panel

Date: 10/2/2012

KEYWORDS

Software testing, risk management, test planning, frameworks, models, software test manager, software test management, decision making, and decision support.

PUBLICATIONS

Some of the material in this thesis has been peer reviewed and published.

Larkman, D., Jentzsch, R. & Mohammadian, M. (2011). "Software Testing - Factor Contribution Analysis in a Decision Support Framework". In J. Watada, G. Phillips-Wren, L. C. Jain & R. J. Howlett (Eds.), Smart Innovation, Systems and Technologies: Vol 10. Intelligent Decision Technologies: Proceedings of the 3rd International Conference on Intelligent Decision Technologies (IDT' 2011), Piraeus, Greece, July 20-22, 2011, (pp. 897-905). Berlin: Springer-Verlag. doi: 10.1007/978-3-642-22194-1_88

Larkman, D., Mohammadian, M., Balachandran, B. & Jentzsch, R. (2010). "Fuzzy Cognitive Map for Software Testing Using Artificial Intelligence Techniques". In H. Papadopoulos, A. S. Andreou & M. Bramer (Eds.), IFIP Advances in Information and Communication Technology: Vol 339. Artificial Intelligence Applications and Innovations: Proceedings of the 6th IFIP WG 12.5 International Conference, AIAI 2010, Larnaca, Cyprus, October 6-7, 2010, (pp. 328-335). Berlin: Springer. doi: 10.1007/978-3-642-16239-8_43

Larkman, D., Mohammadian, M., Balachandran, B. & Jentzsch, R. (2010). "General Application of a Decision Support Framework for Software Testing Using Artificial Intelligence Techniques". In G. Phillips-Wren, L. C. Jain, K. Nakamatsu & R. J. Howlett (Eds.), Smart Innovation, Systems and Technologies: Vol 4. Advances in Intelligent Decision Technologies: Proceedings of the Second KES International Symposium IDT 2010, Baltimore, Maryland, July 28-30, 2010, (pp. 53-63). Berlin: Springer-Verlag. doi: 10.1007/978-3-642-14616-9_5

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I want to thank the chair of my supervisory panel Dr Masoud Mohammadian for his continuous support and ready advice, interspersed with his spontaneous humour, willingness to interact with me and for his friendship. I also wish to thank my research adviser Dr Ric Jentzsch for his insight into the software testing industry, together with his advice and timely suggestions. Dr Jentzsch was able to provide guidance, support and motivation when Masoud was unfortunately unavailable for an extended period due to ill health. Ric was not only a mentor during Masoud's absence, but a friend. I am indebted to Beryl Pedvin and Garry Collins for their timely and expert advice on my numerous referencing questions.

My thanks to Melanie Kovacs of UC AccessAbility, and her predecessors Dee Jackson and John Galvin for their long term funding of a support worker, to accompany me to university so that I could pursue my studies. I emphasise long term because UC AccessAbility has provided financial support for me, every year, since I began studying at the University of Canberra in 1995.

My penultimate acknowledgement is to my family, starting with my wife. I want to thank my long suffering wife Vicki, without whose unconditional support I would never have completed my thesis. Vicki's suffering has been long because of my very protracted studies. Producing my thesis has been a very long journey, with many unfortunate twists and turns, but Vicki has been a constant in my life. My sons, Callum and Keenan were always *very* quick to stir their "old man" for not having finished his thesis (their readiness to stir me was fully justified, because I was stuck on Chapter 1 of my thesis for a very long time). My mother in law, BJ (short for Betty Jane) always asked about my university studies.

Finally, but certainly not least, I want to thank my friends (family friends, university friends, work colleagues and other friends), for their enduring interest in my progress, and for their generous words of encouragement. Julie Dalco well deserves my thanks for proofreading my entire thesis.

CHAPTER 1. OVERVIEW OF THE RESEARCH

The following diagram provides an outline and overview of the conceptual structure of this chapter.

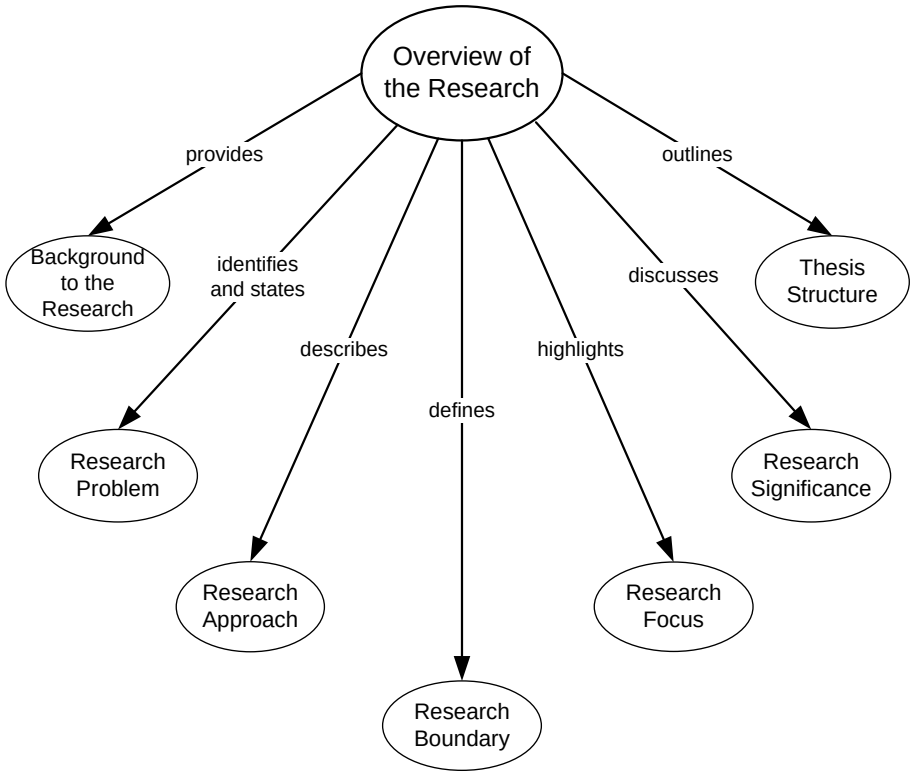


Figure 1.1: Conceptual structure of Chapter 1

1.1 Introduction to the Research

Testing is emerging into a healthy industry and an established discipline. Over recent years, testing has been examined from various perspectives – such as market size, potential growth, the cost of software errors, and the number of testing staff. The global software testing market is said to be worth US$13 billion (Amitysoft Technologies, 2006)[1]. The market opportunity for offshore testing companies in India alone was estimated to rise from US$2 billion in 2006 to US$8 billion by 2008 (Amitysoft Technologies, 2006). The Ovum report (2009) projected that the worldwide market for computer software and systems testing services will reach US$56 billion by 2013, despite taking a hit from the global economic crisis (PhysOrg.com, 2009). The Ovum report also predicted that testing services will grow at a compound annual growth rate of 9.5% from 2008 to 2013, faster than most other information technology services (PhysOrg.com, 2009).

[1] This reference is for a press release on a Web page, and there is a limit on its availability. The reference is no longer available, and therefore an interested reader cannot recover the Amitysoft Technologies reference. Unfortunately, a copy of the relevant Web page was not made while the reference was available.

Inadequate software testing infrastructure is estimated to cost the US economy an estimated US$59.5 billion annually (National Institute of Standards and Technology [NIST], 2002). This cost represented about 0.6% of the US`s US$10 trillion dollar GDP (NIST, 2002). Software developers accounted for about 40% of the total impacts on the testing infrastructure, whereas software users accounted for the about 60% (NIST, 2002).

Examination of the software testing industry in Australia has not been done for well over a decade. The most current information is through the Ross Report. The 2010 Ross Report estimated that in Australia alone there would be an additional need of between 1500 and 2000 software testers for the next two years if the release of software was not going to be significantly delayed (Swan, 2011). The Australian Bureau of Statistics (ABS) publishes information about business expenditure and investment on computer software (ABS, 2010, 2011), but does not collect statistics specifically for software testing.

Business and society are increasingly reliant on information technology. Software is an essential part of information technology, and is a crucial element of the systems and devices that underlie the infrastructure and activities of today's modern lifestyle. The pervasiveness of software can be seen in the plethora of electronic and mechanical devices (both industrial and domestic) containing embedded software; and in the multitude of software applications in use by organisations and individuals. Users depend on the correct behaviour of software to perform the task(s) that they need to or want to do. Users optimistically expect that the software will function as intended, and fulfil the user's requirements and expectations.

Unfortunately, software does fail, and the failures are accompanied by a range of business, financial, legal, human and environmental consequences – and some of the consequences can be devastating. The consequences of software failure range from the trivial (such as the need to restart a program), through the very inconvenient (such as the malfunction of traffic signals), to the catastrophic – where life and property may be at risk (Dick & Kandel, 2005).

Despite its importance to business and society, software is the most labour intensive and most error prone product of the industrial world (Dick & Kandel, 2005). Software has become increasingly complex and highly integrated in meeting the relentless demands of users for more functionality and reliability. Users want the software delivered faster and cheaper, but with the quality of the software meeting, or exceeding, their expectations (Dustin, Garrett &

Gauf, 2009). Users are not only intolerant of failures, but they also demand that the software is reliable, secure, safe, robust, and easy to use. A necessary response to these great demands and expectations is that software development organisations must build high quality software to minimise software failure, and its subsequent impacts, but in less time and with smaller budgets.

Software testing is an important empirical quality assurance technique used to assess and improve the quality of software during its development (Dick & Kandel, 2005; Hailpern & Santhanam, 2002). It has been suggested that software quality assessment may be the hardest problem in the software life cycle (Friedman & Voas, 1995). Testing has been a part of software development since the days of Alan Turing, who wrote an article about checking a large software routine in 1949 (Turing, 1949). Research on automatic programming aims to remove fallible humans from the job of coding large systems, but achieving this aim is still elusive (Dick & Kandel, 2005). In the absence of effective alternatives, software testing remains important today, and represents a significant proportion of a software development project's overall resources of cost and time (Dustin et al., 2009).

The task of the software test team is enormous: not only must they help produce high quality software, a goal that is in tension with the need to test more software, more often, at less cost and with fewer people (Dustin et al., 2009), but the test team has to contend with other more technical challenges. These challenges include being faced with complex software that is often composed of millions of lines of code; ensuring the software works in various environments, with different operating systems and on diverse platforms; and meeting the need to test a range of specific properties for different types of software. Examples of different types of software include embedded software, real-time software, safety-critical software, mission critical software, Web applications, and object oriented software (Ammann & Offutt, 2008). Software constructed concurrently, or with graphical user interfaces, introduces more properties that have to be tested (Pezzè & Young, 2008).

The test manager has to manage the overall software test project. This management includes estimating, planning and organising all the software testing activities. Decision making is an essential part of the test manager's diverse activities, and the success of software testing can hinge on the availability and suitability of support for the test manager to make informed decisions.

This research is guided by a specific target, which is largely defined by the research boundary (see Section 1.4), and the research focus (see Section 1.5). The research target, identified as a well-defined research gap by the literature review (see Chapter 2), builds on the foundation of this research (established in Chapter 3, and enhanced by Chapter 4). A key issue in this research is the distinction between research on decision support frameworks, and research on other forms of decision support in software testing. The apparent previous research on decision support in software testing has been excluded from this research. The reason for this decision was not to dismiss, out of hand, the apparent previous research, but because that research is outside the boundary and the focus of this research. The apparent previous research can only be viewed loosely as containing material relevant to decision support frameworks (the subject of this research). Nonetheless, the apparent previous research must be acknowledged, and serves an important role. The apparent previous research helps establish a decision support context for this research. The context is established because the apparent previous research, on decision support in software testing, forms a body of knowledge which builds a context for this research. The contribution of the research in this thesis adds to that body of knowledge and augments it.

There is inadequate research on supporting decisions in software testing. This claim is governed and constrained by the boundary and focus of this research. Some apparently relevant previous literature includes work related to various decisions: When should testing inspections be terminated? (Miller, Macdonald & Ferguson, 2002); Is there any need for re-inspections of test artefacts? (Briand, El Emam, Freimut & Laitenberger, 2000); What is the suggested order of integrating modules for system testing? (Briand, Feng & Labiche, 2002); How can Orthogonal Defect Classification (ODC) provide feedback to the decision making of the development team, and of management? (Butcher, Munro & Kratschmer, 2002). Nevertheless, analysis of these research examples demonstrates that they lie outside the boundary, and/or research focus of this research. According to Ruhe (2003) there are two kinds of research contributions to decision support in software engineering: direct and indirect (Ruhe, 2003a, 2003b). In the terminology of Ruhe, the apparent previous research contributes indirectly to decision support. See the literature review (Chapter 2) for further examples of previous literature that, in terms of the research in this thesis, can only be classified as loose forms of decision support.

Despite an intensive literature search, the only previous work found on decision support frameworks for software testing is published work arising from this research – see the Publications section of this thesis (Larkman, Jentzsch & Mohammadian, 2011; Larkman, Mohammadian, Balachandran & Jentzsch, 2010a, 2010b). Ramler (2004) proposed a decision support framework for software testing (Ramler, 2004, n.d.), see Chapter 2, Section 2.5.3 and Section 2.6. However, Ramler does not appear to have published any papers that address the core of his proposal. Moreover, the literature search did not reveal any published material that identified the lack of research on decision support for software testing, and that suggested the need for further research in the area. Decision support for the test manager has received very little research attention (see Chapter 2). Research is lacking about decision support for the test manager for their planning and risk management in software testing (see Chapter 2). Therefore, an investigation is warranted to develop a tool to assist the test manager in their risk management and planning associated with successful software testing. The research will adopt a direct approach to decision support. This direct approach will include an explicit effort to develop the decision support tool, a clear focus on the software test manager and their planning and risk management activities, where the decision support tool fits in the overall software life cycle will be described, and use of the decision support tool will be fully explained and illustrated. Such an investigation will help to understand the decision making process of the test manager, clarify some aspects of the role of the test manager, and help to identify where decision support tools can usefully be applied by the test manager in the software testing process.

1.2 Research Problem and the Strategy to Tackle that Problem

This section identifies the research problem, and describes the development of the three layered research strategy designed to address the research problem.

1.2.1 Research Problem

The introduction observed the increasing importance of software to business and society, and emphasised the crucial role of software testing to help produce high quality software, and thereby minimise the potentially serious consequences of software failure. The cost of inadequate software testing comes at a very high price, in terms of the human, environment and economic impacts that can result from software failure (Dick & Kandel, 2005; NIST, 2002). The introduction claimed that research for decision support in software testing is very limited. It was also claimed in the introduction that there is a significant lack of research about support for the planning decisions the test manager is often required to make, which

include risk management planning before and during the testing of the software. These claims lie within the boundary and the focus of this research (see respectively, Section 1.4 and Section 1.5), and are supported by the results of the literature review (see Chapter 2).

The introduction to the research further highlighted several issues that are the basis for constructing the statement of the research problem. The research problem that drives this research is:

> ***What areas of neglect in decision support for software test managers would benefit from the use of a decision support framework, to help ensure more effective implementation of decisions in software testing?***

The use of the word *neglect* in the context of the research problem needs to be clarified. Neglect is used in the sense of inadequate or too little attention to decision support, not the disregard of decision support. Neglected areas of decision support will include decision support that may be occurring in practice, but has not necessarily been reported on, or has not been the subject of research.

The motivation for this research was to address this research problem (which has been posed as a question), and explore how to develop a useful decision support tool for software test managers. The research strategy built to seek an answer to this research problem consists of the research goal, research objectives and research questions. Each part of the research strategy is outlined below. Other important elements that support the research strategy are the research approach, research boundary and research focus (discussed in Section 1.3, Section 1.4 and Section 1.5 respectively).

1.2.2 Research Goal

The research goal was generated from the research problem. The research goal is used to clarify the knowledge to be generated by this research, and to direct the development of the research. The goal of this research is:

> ***To provide a practical, integrated and conceptually sound decision support framework, which can be used to assist test managers achieve successful software testing.***

To achieve this research goal the following research objectives and research questions have to be addressed.

1.2.3 Research Objectives

The research goal was used to construct the research objectives. The research objectives are:

1. identify, describe, and discuss if and how decision support has been used for software testing;
2. explore the reasons for any areas of neglect in decision support for software testing;
3. develop a decision support framework to address the areas of neglect in decision support for software testing; and
4. investigate the framework that emerges to understand how its components interact, and how the framework can be applied by software test managers to their particular organisational and software testing settings.

1.2.4 Research Questions

Given the research problem, the research goal, and the research objectives, the following research questions were developed.

1. How has decision support been used for software testing?
2. What are the reasons for the areas of neglect in decision support for software testing?
3. Can a decision support framework be designed and developed to address the areas of neglect in decision support for software testing?
4. How can the decision support framework be applied, evaluated and interpreted by software test managers?
5. From what perspectives can the decision support framework be evaluated and interpreted by software test managers, and how can this be illustrated?

Obtaining answers to these research questions will satisfy the research objectives, achieve the research goal, clarify the research problem, and help to understand the research problem.

1.3 Approach to the Research

The overall approach to this research can best be described as grounded research. Grounded research draws from the principles of grounded theory, which is a qualitative research methodology (Bryant & Charmaz, 2010; Charmaz, 2006; Glaser & Strauss, 1967; Martin & Turner, 1986). As defined here, grounded research allows the research findings to be developed by exploring the research problem – the areas of neglect in decision support for

software test managers that would benefit from the use of a decision support framework. The concepts, ideas and issues used to develop the research findings are grounded in evidence, reasons, and experience. The evidence, reasons, and experience were based on literature review, information sourced from industry, informal discussions and testing experience. The research findings were generated from the emerging concepts, ideas and issues, and the relationships amongst them. Then the research findings were deductively evaluated.

1.4 Boundary of the Research

The research focuses on the software test manager, and the use of frameworks to support the decision making of test managers in their software testing endeavours. However, this focus needs to have a clear, well defined boundary, which sets unambiguous limits for the research. The elements included in the research boundary, and those elements outside the research boundary are listed below.

Elements included in the research boundary:

- a practitioner, and real world perspective of test management, where test management is the responsibility of the software test manager, or a person with equivalent software test managerial skills and test experience (such as the test head, test lead, or senior tester);
- application software and system software;
- the software product viewed from a software development perspective;
- the software program viewed from the perspective of a flexible basic entity, which can equally be a standalone software program, or a component in a software programming system;
- frameworks for decision support and the application of those frameworks to specific software testing situations (where the frameworks are used to create models that usefully represent specific testing situations); and
- software testing for business activities in organisational settings.

Elements outside the research boundary:

- the software product viewed from the perspective of what is delivered to the user;
- the treatment of software quality and its related issues – the research acknowledges the important role of software quality for an understanding of software testing, and notes

that software quality and its related issues are one of the responsibilities of the software test manager;

- decision support systems – which involve complexities that are inconsistent with the uncluttered, keep it simple approach adopted for the development of the decision support framework produced in this research. The research recognises that decision support systems have a long research history, and that decision support systems continue to attract considerable research interest;
- the use of independent or standalone models for decision support (models that are not constructed from a framework, and that do not fit into a framework); and
- software release (including software release planning) – the organisational process to release fully tested software to the user (where the user may be a customer or a population of users in the wider community).

1.5 Research Focus

This research targets a particular area in the software testing research space. The research boundary consists of a carefully chosen set of parameters, which identify and constrain the research area. Within the research boundary, this research has an explicit focus. The research focus consists of the following four elements:

- *the software test manager* – the purpose of the decision support framework is to help the software test manager achieve the goal of successful software testing;
- *the planning and risk management activities of the software test manager* – the software test manager must have a good grasp of, and control over these activities in their efforts to successfully tackle the software testing process;
- *dynamic software testing* – testing a running program (that is, operating the software as a user would), and using the test results to help improve the quality of the software; and
- *the planning phase of the software testing life cycle (STLC)* – the planning phase is the first phase of the STLC.

1.6 Research Significance

The research is significant because it highlights that decision support frameworks are an under-represented and under-utilised analytical tool for software testing. This understanding helps the reader appreciate that decision support frameworks can potentially be applied to

manage and manipulate the complexity found in a diverse range of uncertain problem domains in software engineering.

1.7 Outline of the Thesis

An outline of the thesis chapters is provided below. The structure and content of each chapter is described and the appendices are briefly summarised. The thesis outline provides a road map for the thesis and thereby a guide to navigate the thesis.

The thesis has been organised to provide a set of related, but logically sequenced building blocks to achieve its goal. Chapter 2 focuses on only the essentials, and includes supporting background material at a relevant level. This ensures that the literature review (Chapter 2) remains uncluttered, maintains its focus and feeds into the next Chapters. Chapter 3 consists of the "base" background material and terminology that builds on the information presented in Chapter 2. Therefore, a solid foundation is provided for the rest of the thesis. Chapter 4, though short, begins to put together the essential concepts and parameters that support Chapters 5, 6, and 7.

- **Chapter 2 – Literature Review**

  Chapter 2 analyses the literature and discusses decision support for test managers when they test and manage software. The groundwork for Chapter 2 is established and discussed in Chapter 3 and Chapter 4. The central role of absence of evidence in the literature review is explored. The criteria developed to facilitate analysis of the literature are stated. Next, decision support is examined as an aid to an organisation's decision makers, in structured, semi-structured, or unstructured situations. The current and past research on decision support for software testing is captured and analysed. The research gap exposed by this analysis demonstrates the need for decision support frameworks, to assist the software test manager when they face the strongly uncertain and complex environment of software testing. Following this analysis, the reasons for the areas of neglect in software testing are explored and discussed. The research gap is the basis for identifying the research problem. Several other issues raised by Chapter 2 help to provide the focus and specificity necessary to examine the research problem, and these issues set the direction and agenda for the remainder of this research.

- **Chapter 3 – Research Foundation**

  Chapter 3 describes the research foundation, which is built on relevant background material underpinned by the terminology related to this research. The research foundation, along with the literature review, helps establish the context for this research, lays the groundwork for a better understanding of the research boundary (first discussed in Section 1.4), and therefore fosters a better understanding of the research. The terminology related to the research foundation is highlighted and defined to avoid ambiguities, and any concepts or issues underlying the terms are discussed. Next, Chapter 3 discusses various aspects of software testing, ranging from the fundamentals of software testing, through the software testing life cycle, to the considerations to be kept in mind for software testing. The chapter defines decision support and focuses on decision making, decision makers and related issues. Risk management in a software testing context is explored, and considered from a decision making perspective. The chapter concludes by examining the needs of software test managers, and their decision making role, in the context of their responsibility for managing the test effort.

- **Chapter 4 – Frameworks for Decision Makers**

  The chapter starts by investigating the concepts and formation of frameworks and models. This investigation demonstrates three things, how frameworks and models assist decision makers, the differences between frameworks and models, and how these differences affect the use of frameworks and models by decision makers. Chapter 4 explores how frameworks relate to decision makers and, from a general perspective, how frameworks support decision making. The focus of the chapter then turns to the software test manager. Two issues are examined, why software test managers need frameworks, and how frameworks can assist the software test manager generally, and in their more specific software testing role. This latter issue prompted an investigation of the activities of the software test manager, to show how decision making is an inherent part of these activities, and how decision support frameworks can assist software test managers. This demonstrates further why decision support frameworks are an important tool, as they support the decision making of software test managers.

- **Chapter 5 – Decision Support Framework for Software Test Managers**

  Chapter 5 presents the development of the decision support framework, a tool for software test managers to use when they test software. A generic decision support structure is established and described. This structure forms the basis for the design of a generic decision support framework, which is illustrated and explained. The generic decision support framework provides the foundation for the design of the decision support framework. This chapter then describes the sequence of steps involved in the development of the decision support framework. Next, the elements, factors, and directional signed relationships for the framework are explored and described in detail. Simultaneously, an application of the decision support framework is illustrated for planning and analysing the risk of not achieving the goal of successful software testing. Where the framework fits into the overall system development and software testing life cycles is identified. Details are provided of the completed model that results when the decision support framework is applied by software test managers to their particular organisational and software testing settings. The decision support framework model created for this application is ready for evaluation, which is dealt with in Chapter 6.

- **Chapter 6 – Using the Decision Support Framework**

  Chapter 6 describes how the test manager would use the model that results from the application of the decision support framework to a specific software testing situation. Several analytical techniques are introduced for the software test manager to evaluate and interpret the model. The static techniques of cross review, risk path analysis, and event path analysis are discussed. The techniques and their use for test managers are explored. The chapter continues by describing how the dynamic techniques of fuzzy cognitive maps and factor contribution analysis can be used with this decision support framework. Chapter 6 emphasises that all the analytical techniques provide test managers with the basis for test planning, and assessing and interpreting risk management in successful software testing. Evaluation of the framework is then explored by illustrating how both dynamic analytical techniques can be used to assist software test managers in their planning of risk management.

- **Chapter 7 – Further Perspectives on the Decision Support Framework**

  Chapter 7 offers additional perspectives on the software testing decision support framework (DSF). The different perspectives consolidate the information established

in Chapter 5 and Chapter 6, provide more insight into the DSF, while underscoring the usefulness of the DSF as a tool that the software test manager has at their disposal in testing any type of software. The perspectives cover a range of areas. Successful software testing is investigated and the multiple dimensions embraced by this framework goal are unravelled, clarified and discussed. The inherent advantages of the DSF, which are the foundation for its capacity to support the software test manager, are described. The relationship of the framework to test planning is examined. The chapter takes a unified look at how the software test manager applies the DSF to their particular software testing situation. Chapter 7 concludes with a detailed discussion of additional insights into the DSF.

- **Chapter 8 – Conclusions and Future Research**

  Chapter 8 reflects critically on this research, focuses on and discusses several areas, and looks to the future. The chapter first identifies and highlights a collection of issues (based on a background review and research considerations), which are a foundation for a better understanding of the chapter. Then the implications of the research findings are explored: answers to the research questions are described, and the decision support framework (the major research contribution) is presented, together with an outline about how the software test manager can use the framework. The research findings are analysed and used to draw several conclusions. Issues that accompany real world use of the decision support framework by software test managers are investigated. Next, the limitations of the research are discussed at length, and proposals for future research are described. The final remarks of the chapter offer an overall perspective to complete this research.

- **Appendices**

  Appendix A includes examples of approaches to software testing described in the literature, and thus provides additional supportive information that is lateral to the essential part of the research. Appendix B lists the terminology defined for this research, and other terminology used in the thesis, but sourced, unchanged, from various authors. Finally, Appendix C enumerates the different abbreviations used in this thesis.

- ||| -

CHAPTER 2. LITERATURE REVIEW

The following diagram provides an outline and overview of the conceptual structure of this chapter.

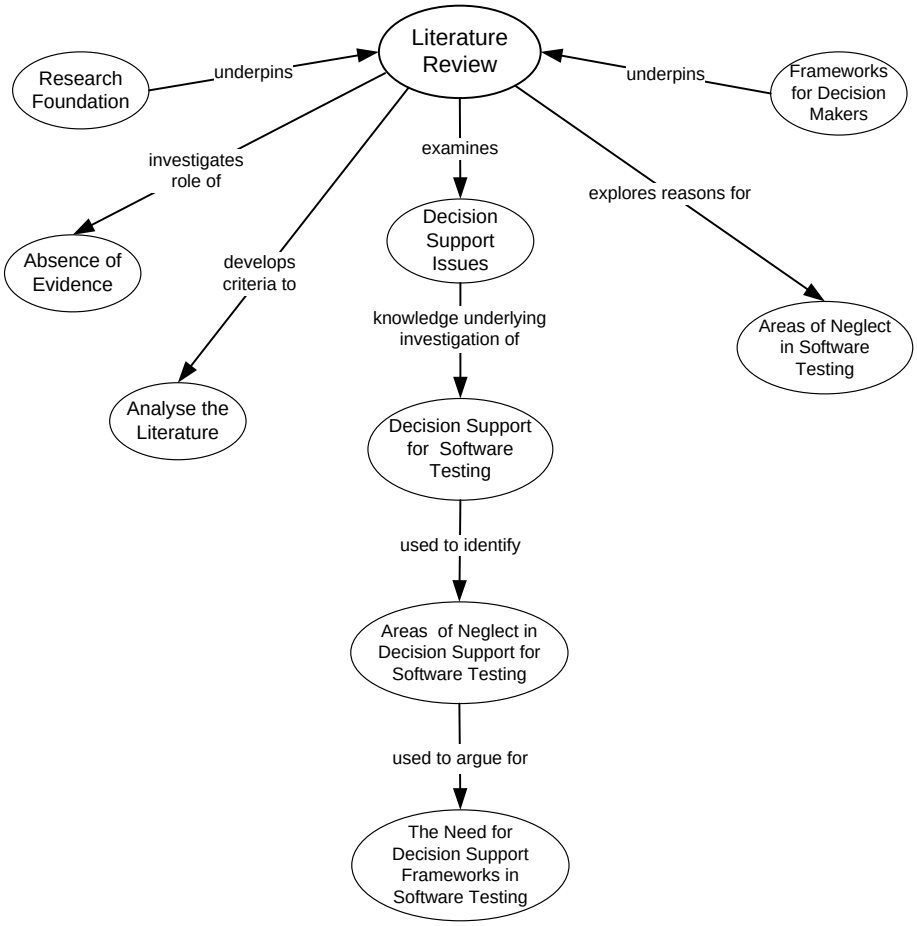


Figure 2.1: Conceptual structure of Chapter 2

2.1 Introduction

The literature review is underpinned by and builds on the background material and terminology discussed in Chapter 3, and the material on frameworks explored in Chapter 4. Absence of evidence is central to establishing the findings of the literature review, and the role of absence of evidence is investigated. The criteria that were developed and applied to the literature review are based on addressing the research problem. Decision support is introduced and examined as a tool to promote the decision making process, especially in an organisational context. The meaning of decision support and the issues that underlie that meaning are given close scrutiny. The basic elements that make something decision support are described: and these elements form the kernel of every decision support mechanism, regardless of its complexity. This strategy aims for a better understanding of decision support within the context of this research.

Building on an understanding of decision support, the focus of the chapter shifts to investigating how decision support has been used for software testing. Using this investigation, the neglected areas that exist in decision support for software testing are highlighted and discussed. The reasons for the neglected areas of decision support are explored next. The neglected areas of decision support are analysed and used to construct and present the argument that decision support frameworks are needed to assist the software test manager when they test software. A decision support framework is developed and explored in the thesis as the way to address the research problem.

2.2 The Literature Review and Absence of Evidence

The claims made in the introduction to the research (see Chapter 1, Section 1.1), for the lack of research on decision support framework development and use in software testing, and other forms of decision support in software testing, are based on an absence of evidence in the literature. Put another way, the absence of evidence suggests the non-presence, or non-existence, of research that investigates the aforementioned areas in software testing. It is acknowledged that "absence of evidence is *not* evidence of absence". Nevertheless, it is asserted that the literature review, driven by the boundary and the focus of this research, provides the diversity and thoroughness of analysis necessary to establish the absence of evidence, which underlies the claims made in the introduction to the research. The preceding sentence clarifies an important role of the literature review in this thesis – to establish that there is an absence of evidence. However, this perspective on the role of the literature review has a counterpart; and it is instructive to examine the inherent drawbacks in support based on absence of evidence.

Support for absence of evidence will always attract close critical scrutiny. But to successfully meet that scrutiny is not easy. The support can only be inferred or deduced from the absence of evidence, and is therefore difficult to prove. Additionally, the support will always have an element of uncertainty, for at least two reasons. Published research in the area may have escaped the literature search, and research in the area may exist, but not have been published. Furthermore, the reading audience for the absence of evidence can potentially be drawn from a range of disciplines. Each reader will bring different knowledge, experiences and values, which will affect the way they perceive the support for absence of evidence, and partially determine if they accept or reject that support.

The introduction to the research, which is brief, is not a suitable place to establish support for absence of evidence. Only the literature review, grounded in the boundary developed for this research, has the depth and diversity of analysis required to show that a well-defined, but hitherto unidentified gap, does exist in the decision support literature on software testing.

2.3 Criteria Applied to the Literature Review

Several criteria were developed to determine the suitability of prior research found during the literature review. The research problem and research goal (as outlined in Chapter 1) were analysed to determine the selection criteria. This analytical exercise was refined by applying relevant aspects of the research boundary described in Chapter 1. The selection criteria that resulted from this process include:

- a clear but simple understanding of decision support needs to be presented, with a focus on its essential characteristics;
- decision support in software testing has to address how decision support has been used in support of achieving successful software testing; and
- the requirement to clearly examine what software test managers need to do their job, as opposed to their role, responsibilities or activities in their job.

2.4 Decision Support Issues

This section explores decision support for decision makers as it relates to structured, semi-structured, and unstructured situations/issues/problems/requirements/opportunities. The issues and concepts are investigated, with the aim of understanding the issues and investigating two different perspectives of decision support. Attention then turns to the minimum set of elements that characterise, or make something decision support.

2.4.1 Understanding Decision Support

A decision support mechanism of some type can be developed for a decision maker. Generally, these mechanisms are focused on managers within specific applications (accounting, finance, etc.). The role of a decision support mechanism is to support the decision maker in their decision making role, not to replace their judgement (Hayen, 2006; Turban, Sharda & Delen, 2011). Thus, the effectiveness to which a decision maker can make a decision is increased using decision support mechanisms, rather than an increase in the efficiency of the decision maker (Hayen, 2006).

The structured, semi-structured or unstructured situations could be situations, issues, problems, requirements or opportunities. A structured situation has a framework comprising elements and relations between them that are well known and well understood. In contrast to structured situations, decision makers may have considerable familiarity with the elements and relationships of semi-structured or unstructured situations, but the elements and relationships are less well known and less well understood. Elements and relationships of semi-structured or unstructured situations will vary within and between situations, issues, problems, requirements and opportunities. In these situations the decision maker will need additional information to enhance and support the various outcomes in support of the decisions.

Decision making can be improved because some decision support mechanisms allow the decision maker to apply their knowledge to not only structured but also semi-structured or unstructured situations. Knowledge has been discussed from various perspectives (Khosrowpour, 2000; Wang, Hjlmervik & Bremdal, 2001; White, 2002). Knowledge is defined, for this research, as the decision makers' practical understanding of or experience within a particular domain.

The majority of decision support mechanisms are used to collect and analyse business data, and present it for business decisions, such as the decisions required for solving a problem or for evaluating opportunities (Turban et al., 2011).

The definition of decision support is based on Alter's perspective of decision support (Alter, 2004). Alter wanted to understand decision support in real world settings and suggested a broad interpretation of decision support, and proposed the context of a work system for decision support. Alter's (2004) perspective recognised that typical organisations use work systems to achieve their many business functions, and that business professionals care about improving decisions, and have no special preference about alternative approaches to how decisions are improved (Alter, 2004).

This research adopts the broader perspective presented by Alter, and sees decision support in the context of work systems. Thus, it is instructive to understand the nature of a work system. A work system is a system where humans and/or machines perform a business process, using information, technology, and other resources, to create products and/or services for internal or

external customers (Alter, 2004). Alter (2001) developed a work system life cycle model, and he considered that the life cycle of a software product is a type of work system (Alter, 2001).

Alter (2004) proposed that a work system has nine elements: business process, participants, information, technology, products and services, customers, infrastructure, environment and strategy. Alter (2004) suggested that decision support may come from any of the nine different elements of the work system, not only technology. Alter (2004) argues that decision support systems (DSSs) research examines technological artefacts to support decision making. DSSs are highlighted here because they have been an academic research subject of extensive research for more than forty years (Alter, 1980; Keen & Scott Morton, 1978; Sprague & Carlson, 1982; Turban et al., 2011).

DSSs can be interactive computer or non-computer based systems that help decision makers use data, models and data manipulation tools to make decisions in structured, semi-structured or unstructured situations (Alter, 2002; Power, 2001; Turban et al., 2011).

As stated previously, decision support systems (DSSs) involve complexities that are outside the boundary of this research (see Chapter 1, Section 1.4). To understand why decision support systems fall outside the boundary of this research, the complexities of DSSs are discussed next. It is important to note that the DSS complexities contrast with the uncluttered, keep it simple approach, adopted for the development of the decision support framework constructed in this research. Chapter 1 identified this development approach for the framework (see Chapter 1, Section 1.4). Decision support systems are systems, which rightly or wrongly, suggests a level of complexity because a system is a set of interacting components (Alter, 2002). Decision support frameworks, like any framework, are composed of a collection of ideas and assumptions (see Chapter 4, Section 4.3.1). These ideas and assumptions can be translated to a collection of interacting components, which can be viewed as a simple system. However, even simple DSSs involve a level of complexity (in terms of their components and functionality) not apparent in decision support frameworks.

Decision support systems involve different levels of functionality and focus on a major technological component (Power, 2001; Turban et al., 2011). By definition, DSSs include three major components: data management, model management and a user interface (Turban et al., 2011). Even assuming an elementary level of functionality for DSSs invokes some

complexity. A DSS with elementary functionality could involve a simple spreadsheet user interface (a common form of user interface for many forms of decision support), and complexity would be added depending on the focus of the DSS. For example, simple query and retrieval tools to access and manipulate data (a simple data source would be a file system), or simple statistical and analytical tools to access and manipulate the particular model in use (Alter, 1977; Power, 2001). The degree of functionality of DSSs, and hence their complexity, escalates from this elementary level of functionality.

2.4.2 What Makes Something Decision Support

The elements that make something decision support can be discovered by analysing the issues common to definitions for decision support and decision support systems.

Something is decision support when it is developed by any computerised or non-computerised means to:

- support and assist decision makers in their decision role, but not meant to replace their judgement;
- provide a degree of value to the decision at the point of time when it is needed;
- improve the effectiveness of decision making (effectiveness of the decision considers accuracy, timeliness and quality) (Turban et al., 2011);
- assist in formulating decisions based on issues in structured, semi-structured or unstructured situations; and
- accommodate situations where information can, needs to, and does change over time.

If something is to support decision making, irrespective of its complexity, it must address the above characteristics in some way.

2.5 Decision Support for Testing Software

The research boundary for this thesis is explicit (see Chapter 1, Section 1.4). The research targets decision support for the software test manager; and the research also has a focus on dynamic software testing (see Chapter 1, Section 1.5). The preceding information sets a clear, well-defined guide on the search for relevant literature about decision support in software testing.

Within the parameters established by this thesis for the literature review, there has apparently been little research that deals with providing decision support for software testing.

Furthermore, there is scant research with an explicit and direct focus on risk management, on the use of test metrics or on the use of artificial intelligence techniques for decision support in software testing (see respectively, Section 2.5.2, Section 2.5.3 and Section 2.5.5 for discussions and related references).

2.5.1 Decision Support for Test Planning

Software test planning figures prominently in the test process, and is an important responsibility of the software test manager (Ammann & Offutt, 2008). The software test manager has to consider all the areas that are relevant to testing a particular type of software. Such areas include the resources needed, staff responsibilities, risk involved, and the test requirements (Mosley & Posey, 2002). Harnessing and integrating these areas allows the test manager to develop the test plan, a major output of test planning, which guides the test process (Kaner, Bach & Pettichord, 2002).

Test planning consonant with the terminology of this research (see Chapter 3, Section 3.5 and Chapter 7, Section 7.4.1), is the high level planning necessary to obtain, and organise, the information relevant to testing a particular type of software. Decision support for software test planning defined here has received scant attention in the literature. An intensive search of the literature exposed only one report apparently addressing decision support for software test planning. However, this report has an "in press" status and only the abstract is available. A further search did not find any archives where the contents of that report were available. The ensuing discussion is therefore indicative. Only the contents of the report can allow a proper assessment to be made of the research, which will either support or refute the analysis below.

The research, amongst other things, addresses planning for system tests in software organisations with a product line approach (Engström & Runeson, 2011). Software product line development refers to software engineering techniques for creating a collection of similar software systems, from a shared set of software assets, using common means of production ("Software product line", 2011). The need for tool support for this test planning is noted, and it is implied that the tool support provides decision support (Engström & Runeson, 2011). Furthermore, the research appears to focus on the possibilities of implementing context specific decision support, at various levels of automation related to regression testing (Engström & Runeson, 2011). It therefore appears that the planning for system tests (in a product line context) is a specific planning activity focused on a particular type of test, which

is *not* equivalent to the test planning that is the subject of this research. Test planning in the context of this research is the planning necessary to elicit and organise the information pertinent to testing a particular type of software.

2.5.2 Risk Management and Software Testing

Software testing is a risk management activity, intended to give confidence commensurate with the risks involved in the software project (Redmill, 1999). Charette (2005) has observed that one of the common factors for software project failure is unmanaged risks (Charette, 2005), and unmanaged risks cost money (Dedolph, 2003). The consequences of software development risks are far reaching "Almost every software development risk event affects schedule, quality, or both." (Dedolph, 2003, p. 92).

Journal articles, conference papers, books, and blogs (Bannerman, 2008; Craig & Jaskiel, 2002; Dustin, 2003; Dustin, Rashka & Paul, 1999; Silva, 2008; Tamres & Mills, 2002), when it comes to software testing, discuss risk management within the software testing environment, from the view of:

- analysis of the type of software to be tested and its characteristics;
- software testing impact on the business;
- testing component selection;
- prioritising testing component selection; and
- assessment on assigning of testing resources.

Risk management, particularly being able to assess risk in great detail, is a difficult task (Dustin, 2003). The task of risk management needs to be done as early as possible in the software testing life cycle (Craig & Jaskiel, 2002). Both Dustin (2003) and Craig and Jaskiel (2002) give a high level view on risk, as it relates to addressing the resources needed during the planning process. As Silva (2008) and Dustin et al. (1999) have pointed out, the planning phase is a critical phase for risk management in successful software testing (Dustin et al., 1999; Silva, 2008).

Risk management has been related and applied to business risk analysis during software testing (Craig & Jaskiel, 2002). But no formal method or framework for doing the risk management has been presented.

Risk management for software testing seems to target resource assignment (Dustin, 2003), selecting relevant tests to be done (Tamres & Mills, 2002), selecting or prioritising what parts of the software to test or not to test (Craig & Jaskiel, 2002), and/or which components must be tested more thoroughly (Borysowich, 2005).

The majority of authors agree that the risk management process needs to have a formal structure of one type or another (Craig & Jaskiel, 2002; Dustin, 2003; Tamres & Mills, 2002). Some authors discuss the use of an informal approach to risk management but note that the key to successful risk analysis is a formalised approach (André, 2008). Others discuss risk management within a more formalised approach, and consider the steps or phases in the approach (Borysowich, 2005; Craig & Jaskiel, 2002; Dustin, 2003; Maidasani, 2007; Tamres & Mills, 2002).

2.5.3 Software Test Metrics for Decision Support

The question here is: What is the role of test metrics in decision support for software testing? To answer this question this section briefly examines software test metrics, the categories of software test metrics, and their different purposes (including some possible groups of measurements). Building on this, an industry perspective of software test metrics is explored. This industry perspective is compared with academic research on software test metrics.

Software test metrics have value in management, monitoring, planning and control (Mathur, 2008). Software testing metrics are generally categorised as project, process and product. One other author suggests an additional category of software test metrics: organisational (Mathur, 2008). In software testing, metrics are used for project management (Bradshaw, 2004), process assessment and improvement, and making product related decisions (Mathur, 2008). Test metrics may be used to evaluate the output and/or efficiency of the software testing team's testing work (Bradshaw, 2004; Hass, 2008). Test metrics should be easy to understand and objectively quantifiable (Bradshaw, 2004). Numerous measurements can be collected during the performance of test procedures for different purposes; and they can be divided into groups. The groups include measurements about progress, coverage, incidents and confidence (Limaye, 2009).

In a software testing project in the real world, during testing and after completing the software testing life cycle, what is measured, and the use of those measurements, may have little

decision support consequences (Bradshaw, 2004). A metric must have value to the project and its outcomes (Bradshaw, 2004).

In industry: of the many test measurements collected, only those metrics used to monitor test project progress (for example, using the metric: total number of tests completed/total number of tests) (Bradshaw, 2004) will have some type of decision support consequences (Bradshaw, 2004). These metrics will be used to inform management about test project progress, and thus provide decision support for management. Management may decide to assign more resources to the testing project, or if the testing is going faster than anticipated, resources might be reassigned to other projects. So the software test manager must respond to management directives about movements of resources for the test project, and make decisions accordingly. Thus, from one perspective, there is indirect decision support for the software test manager from the metrics used to monitor test project progress.

The literature review on software test metrics revealed only one paper, sourced from academia, which indirectly addressed the decision support implications of test metrics, for an aspect of the software testing life cycle (Chen, Probert & Robeson, 2004). However, there was little or no further investigation discussed in this paper about what specific decision support test metrics provide.

It was claimed that "Effective test process measurement is useful for designing and evaluating a cost-effective test strategy." (Chen et al., 2004, p. 111). These authors presented research which showed that effective test metrics provide the decision support required to design and evaluate a cost-effective test strategy (part of the test plan, which is involved in the planning step of the software testing life cycle). However, this was a secondary conclusion of the research, not given explicit attention after the initial claim was made in the abstract of the report.

The pilot project used to evaluate a set of metrics for cost, time and quality, which existed in an IBM internal test strategy document, had a test process measurement focus (Chen et al., 2004). The focus was twofold: on evaluation of the test process and on test process improvement. Preliminary results from complementary test metrics, for two releases of a real-life IBM product, were analysed and compared. One conclusion from this analysis was that the test metrics used in the pilot study provide useful information to managers for

decision making (Chen et al., 2004). The implication was that the test metrics (defined in the IBM test strategy document), used to measure the software test process, provide decision support information for the test manager to evaluate the test strategy, and redesign that test strategy if required.

Usefulness of metrics to help create the test strategy has been investigated by Afzal and Torkar (2008). They examined metrics to support the test planning and test design processes, which are essential elements of the test strategy (Afzal & Torkar, 2008). Based on a survey of the relevant literature on test planning and test design, a set of software attributes was identified, and metrics were formulated from the attributes. The resulting metrics support the decision making involved in test planning and test design (Afzal & Torkar, 2008). It was suggested that an organisation can build an effective organisational level test strategy, which includes metrics identified for the test planning and test design processes (Afzal & Torkar, 2008). The effort to create an organisational test strategy should lead to informed decision making about different software testing activities (Afzal & Torkar, 2008).

The appropriateness of test metrics as a guiding and decision tool was explored by Rivers and Vouk (1999). Rivers and Vouk (1999) presented an approach based on a test efficiency metric, which provided dynamic feedback and testing process decision support during resource constrained testing (RCT) of software (Rivers & Vouk, 1999). The authors suggested that testing efficiency metrics are more appropriate than traditional, operational profile based failure intensity decay metrics (which encourage the re-test of previously tested functions). The reason advanced to explain this suggestion was that modern business and software development practices tend to discourage the re-test of functions that have already been tested (Rivers & Vouk, 1999). Consequently, in RCT environments, failure intensity decay metrics are open to interpretation, and may be an impractical decision tool. A large commercial software system was used to confirm the authors' suggestion that test efficiency metrics are more appropriate to RCT environments (Rivers & Vouk, 1999).

Decision support for test management motivated the proposal of Ramler (2004). In his PhD proposal, Ramler suggested a framework to support rapid and informed decision making for test management, within the context of the iterative and evolutionary approaches to software development practice (Ramler, 2004). The goal of the proposal was to develop a framework of metrics to support the test manager in decisions about how to determine an optimal set of

highly effective tests within strict schedules and tight budgets, and the multiple test cycles of an iterative and evolutionary software environment (Ramler, 2004). It was noted that the value of testing has to be taken into account in any decision making about optimal test sets. Subsequently, the focus was on how to identify the value of individual tests – where it is assumed that the most valuable tests uncover defects in the software (Ramler, 2004).

A four part solution approach was proposed to identify valuable tests. The solution approach consisted of a framework of metrics, measurement results associated to tests, successive adaptation to an optimal set of tests and automated support for decision making in test management (Ramler, 2004). The conceptual framework will include a set of metrics and valuation criteria, and will be open – neither including nor excluding certain metrics (Ramler, 2004). The measurements (metrics) will be associated to the entities that are actively managed during testing, such as test cases. Successive and rapid adaptation to an optimal set of tests is advocated to account for the volatile environments of iterative and evolutionary development (Ramler, 2004). The automated support will include a powerful visualisation capacity to foster quick exploration of alternative options for decision making (Ramler, 2004).

Preliminary results were provided as statements of activities undertaken in the initial stage of work on the proposal (Ramler, 2004). Value criteria that drive decision making were investigated, a preliminary step for work on the conceptual framework of metrics (the framework purpose is to determine the value of single tests and thereby provide alternative solutions for test management decision making) (Ramler, 2004). The technical feasibility of extending management tools to accommodate automation was studied (Ramler, 2004). The automated tool extension is required to allow test managers to handle and visualise large quantities of data (generated in iterative and evolutionary development environments), and to make rapid and informed decisions (Ramler, 2004).

The PhD proposal described above is very interesting, but appears to be a work in progress. A listing of current PhD projects on the Software Competence Center Hagenberg website indicates that Ramler's PhD is still current ("PhD projects", n.d.). However, the information on this website is not dated, and it can only be assumed that the information is up to date. Ramler’s PhD proposal (Ramler, 2004, n.d.) offers little information in support of software test metrics for decision support. Ramler has published extensively on various topics.

However, a comprehensive search of the literature did not reveal any publications that addressed the core of Ramler's PhD proposal.

2.5.4 Decision Support for Automated Testing

Many books, journal articles, and conference proceedings discuss automated software testing. The majority of these discuss when to and where to automate software testing. A few discuss the balance that is needed between automated and manual testing, managing automated testing, and other related issues (Dustin et al., 1999; Farrell-Vinay, 2008; Weiss et al., 2009). In contrast, research specifically about decision support for automated software testing is very scarce.

A comprehensive search of the literature exposed only one study of decision support for automated software testing (Schuster, Sterritt, Adamson, Curran & Shapcott, 2000). However, this study addressed the decision support provided by a prototype decision support system – involving a rule based component and a genetic algorithm component (Schuster et al., 2000). Decision support systems involve complexities that are outside the boundary of this research (see Chapter 1, Section 1.4 and Section 2.4.1 of this chapter).

2.5.5 Artificial Intelligence Techniques for Decision Support

The question here is: What artificial intelligence (AI) techniques have been used for decision support in software testing? To more fully understand this section a brief list of the various AI techniques is presented. From here AI techniques that have been used for decision support in software testing will be discussed.

Artificial intelligence techniques that have been identified in the literature include such areas as: expert systems, machine learning, inductive reasoning, pattern recognition, artificial neural networks, Bayesian networks, fuzzy logic and fuzzy sets, intelligent agents, artificial intelligence planning, genetic algorithms, non-monotonic reasoning, logic programming, natural language processing, rough sets and computer vision (Kalogirou, 2007; Krishnamoorthy & Raveev, 1996; Russell & Norvig, 2003; Słowiński, 1992).

Artificial intelligence techniques relating to software testing have been discussed from various perspectives (Bunke, Kandel & Last, 2004; Dick & Kandel, 2005; Raza, 2009). A collection of articles on AI and software testing in Bunke, Kandel and Last (2004) provides little

evidence of research that considers, or addresses, decision support in software testing (Bunke et al., 2004).

Fuzzy cognitive maps (FCMs) are a hybrid technique, based on a combination of the AI techniques, artificial neural networks and fuzzy logic (Groumpos, 2010; Mateou, Moiseos & Andreou, 2005; Salmeron, 2010; Tsadiras, 2008; Xirogiannis & Glykas, 2007). One perspective of the FCM is that they are an AI technique. This research adopts that perspective. Fuzzy cognitive maps have been used in diverse areas of decision support that include: specific areas of medicine – labour (Stylios, Georgoulas & Groumpos, 2001; Stylios & Georgopoulos, 2010) and radiation therapy (Papageorgiou, Stylios & Groumpos, 2003); geographic information systems (Liu & Satur, 1999); intrusion mechanisms for protecting computer systems (Mu, Huang & Tian, 2004; Siraj, Bridges & Vaughn, 2001); maintenance of water quality in water mains (Sadiq, Kleiner & Rajani, 2004); and cotton crop production (Papageorgiou, Markinos & Gemtos, 2010). Despite this diversity of decision support, a thorough literature search did not reveal any reports of the application of FCMs to the software testing domain.

The literature review identified only one AI technique associated with decision support in software testing: Bayesian networks. Bayesian networks are the basis of Bayesian graphical models (BGM), which were reported to support the complex tasks that software test managers, and testers, face when they test software systems (Rees, Coolen, Goldstein & Wooff, 2001; Wooff, Goldstein & Coolen, 2002). The relevance of the BGM approach to support software testing, from a software test manager perspective, has been discussed (Rees et al., 2001).

Bayesian graphical models were constructed to probabilistically and statistically model the uncertainties in the quality of software and in the process of testing (Rees et al., 2001). The BGM are dynamic representations of the software testing problem, and can be used for different purposes, and one of these purposes is for BGM to function as a decision support system for evaluating, and choosing how to sequence a test suite suggested by a tester (Rees et al., 2001; Wooff et al., 2002).

The question that now arises is: How do BGM provide decision support for software testing? Provided an existing test suite exists (sourced from a tester), the BGM can be used to calculate the probability of tests resulting in failure (Wooff et al., 2002). This information

provides the decision support necessary for the tester to use their judgement to evaluate the tests, and decide on the utility of running a particular test at that point in the test sequence (Wooff et al., 2002). The tester can then, for example, sequence the test suite to test software areas with a high probability of failure (Wooff et al., 2002).

Part of developing the BGM involves eliciting probabilities from testers to quantify the models (Rees et al., 2001). Thus, an advantage of BGM is that they capture a degree of expertise and knowledge of the software tester, and maintain it in a knowledge base for further use (Wooff et al., 2002). This knowledge base is a useful resource, which can be used by the software owner to help counter the problem of the loss of tester expertise, for example, through personnel changes (Wooff et al., 2002).

The BGM approach to software testing comes with an extra cost: the time required to quantify the uncertainties in the models, and the learning of Bayesian statistics required to use this approach (Rees et al., 2001). Nonetheless, the authors view the BGM approach as promising, and they suggested that some training on Bayesian statistics should be part of the basic education for testers (Rees et al., 2001). The BGM exploit the expert judgements of the testers. These expert judgements represent a potential limitation of the BGM approach. If the judgements are overly simplistic, the BGM will give an inaccurate representation of the faults in the software (Wooff et al., 2002).

2.6 Frameworks for Decision Support in Software Testing

The literature review has found a variety of decision support frameworks that have been applied to different areas. Medical applications include automated screening of diabetic retinopathy (Kahai, Namuduri & Thompson, 2005), and the decisional conflict between patients and physicians (Légaré, O'Connor, Graham & Wells, 2006). Some decision support frameworks are directed at specific issues, such as the prevention of water pipe damage in the insurance industry (Strauss, Stummer, Bauer & Trieb, 2009), and high technology manufacturing and service networks (Lendermann et al., 2005).

The literature review found little to no evidence of any research that deals with decision support frameworks related to software testing. Furthermore, research that applies to the development, or utilisation, of a decision support framework specifically designed for software test managers, has not been identified. Apparently, the only previous work in the

literature focused on decision support frameworks for software testing, are reports produced from this research (Larkman et al., 2011; Larkman et al., 2010a, 2010b).

Ramler (2004) proposed a conceptual, open and customisable test metrics framework to support the decision making of test management, in the context of iterative and evolutionary software development (Ramler, 2004, n.d.). See Section 2.5.3 for analysis of Ramler's PhD proposal, which hinges on the idea of the decision support framework. The framework will support test managers in focusing on an optimal set of highly effective tests in order to increase overall efficiency in software testing (Ramler, 2004, n.d.). Thus, the objective of the framework is to continually focus testing on the value it provides for the software project, by focusing on the most valuable tests (Ramler, 2004). Subsequently, the framework has to address a central question: How can these valuable tests (which should be focused on), be identified? (Ramler, 2004). Although an interesting approach to decision support for test management, it appears that Ramler has not published any papers dealing with core of the research work described in his PhD proposal.

2.7 Why Decision Support Frameworks are Needed for Software Testing

The analysis of the software testing literature in Section 2.5 and Section 2.6 identified a gap in the literature – the scarce research about decision support for software testing, where that research is consistent with the boundary and the focus of this research. This section attempts to clarify the research gap by summarising the essential points of the literature analysis. The research gap is also put into a broader, more understandable picture.

Figure 2.2 illustrates a summary of the literature analysis on decision support for software testing, supplemented by several contextual areas, and their respective issues. The areas for the context are drawn from some of the salient ideas captured in discussions of software (see Chapter 1, Section 1.1 and Chapter 3, Section 3.3.1), software testing (see Chapter 3, Section 3.3.1), decision making (see Chapter 3, Section 3.4), decision support (see Section 2.4 of this chapter) and the software test manager (see Chapter 3, Section 3.7). The summary of the literature analysis and the information about the contextual areas, are extracted from evaluation of references identified in the sections mentioned in the preceding sentence.

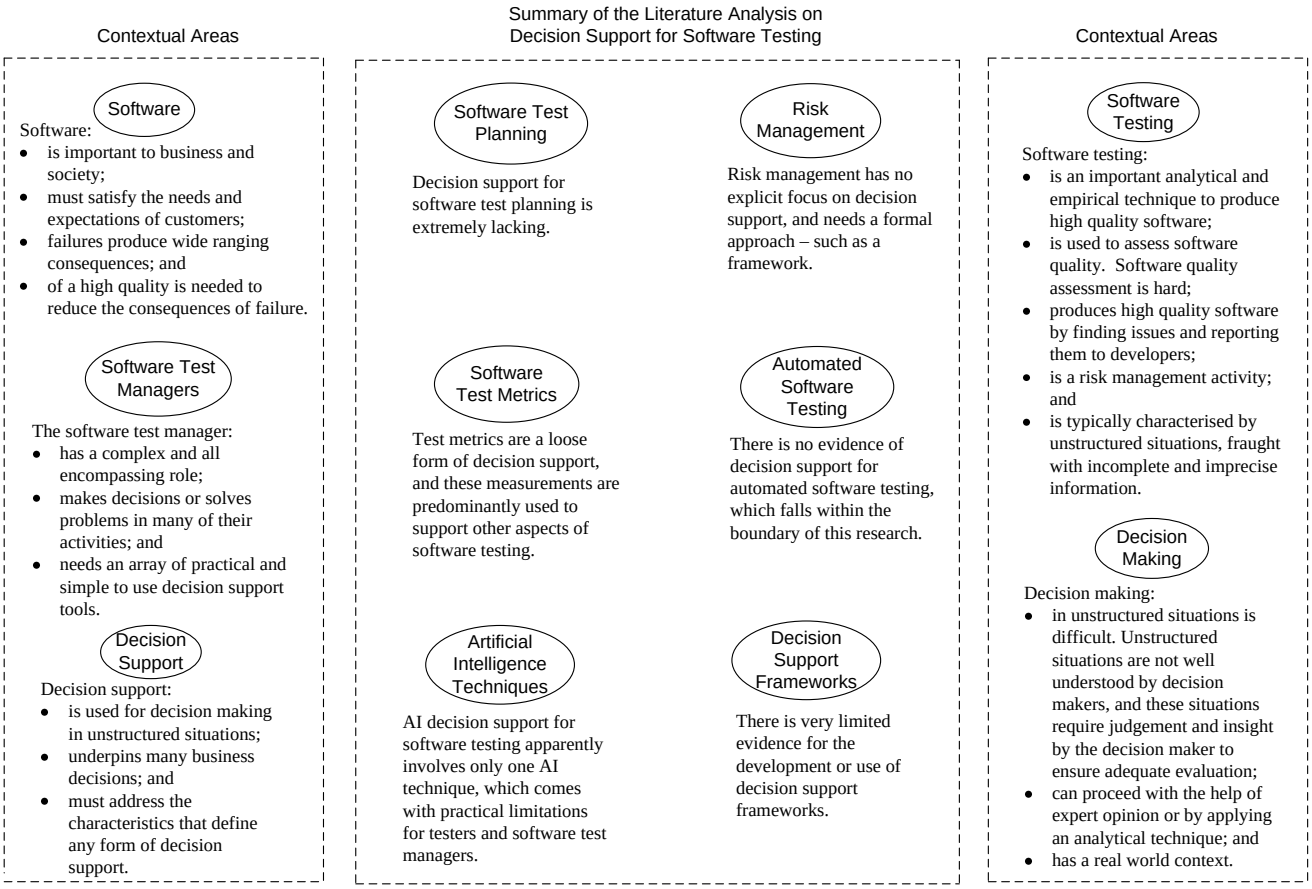


Figure 2.2: Summary of the literature analysis supplemented with contextual areas

Figure 2.3 is driven by the quest for simplicity, but not at the expense of clarity. Figure 2.3 abstracts the essential elements from Figure 2.2, shapes these elements to increase structure and foster cohesion, and thereby provides a visual signpost for the reader to see the response to the research gap, and to see the broad basis for the direction of this research.

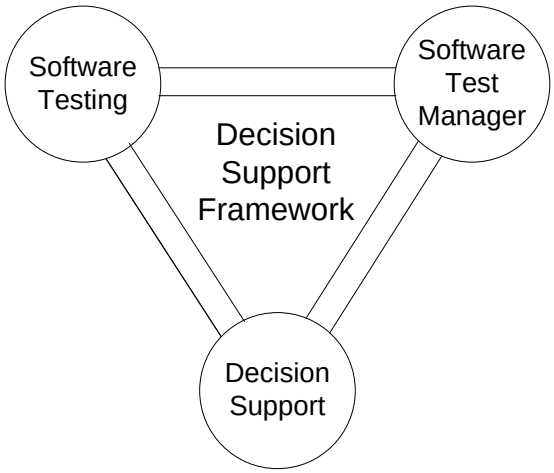


Figure 2.3: High level view of the research direction

The construction of Figure 2.3 is partly based on the central finding of the literature analysis of decision support for software testing. The central finding is the need for decision support frameworks as a useful approach to address the significant lack of decision support for software testing – especially in the areas of test planning and risk management, which are essential activities of the software test manager. The perspective of Figure 2.3 is on this research. Therefore, the singular form of the phrase "decision support frameworks" has been highlighted in the figure. Figure 2.3 has further identified three domains (software testing, software test manager and decision support), which interact and connect with the decision support framework, and provide it with an indispensable context.

2.8 Reasons for Areas of Neglect in Decision Support for Software Testing

Software testing is a complex process and is becoming more complex each year (Dustin, 2003). Software testing comes with some difficult issues, among these being the definition of the type of software to be tested (Dustin, 2003). Decision support has been slow to address the reasons for the neglected areas. Reasons for this can be seen in the categorisation of types of software and a breakdown of issues within those types (Borysowich, 2005).

One of the main reasons for the neglected areas of decision support in software testing is the sheer number and types of software that needs to be tested. There are various categories and perspectives offered on types of software (James, 2009; Jones, 2007; Leffingwell & Widrig, 2000; Myers, 2004; Patton, 2006). None of these categories and perspectives includes everything, but all together they provide the basis on why decision support for software testing is complex. Myers (2004) discusses some of the issues of types of software to be tested. These conclude that to build a decision support for software testing would be difficult, as it would need to address many types of software and the evolving nature of information technology.

A second reason is the number of categorisations of the types of software. Because there are so many categories of software that are tested, it would be difficult to develop a decision support system for software testing. Some of these types of software areas are outlined below.

Oak (2011) categorised types of software as:

- programming software;
- system software; and
- application software.

He then goes on to look at some other well-known forms of computer software, like inventory management, utilities, Enterprise Resource Planning (ERP), accounting, and some others (Oak, 2011).

Kayne (2011) looked at the different types of software from a little different perspective. He categorised them into:

- retail software;
- Original Equipment Manufacturer (OEM) software; and
- shareware (Kayne, 2011).

The literature review has shown software categories including cripple ware, demo software, adware, spyware, freeware, public domain software and other categories (Bangia, 2007; Dhunna & Dixit, 2010; Holtsnider, Wheeler, Stragand & Gee, 2010; James, 2009).

It can be seen why there are neglected areas of decision support in software testing given the number of diverse software categories. Decision support for software testing would have to be specific to a type of software.

When software is sent to testing, the type of software determines a great deal about what needs to be addressed: the skills, environment, risks, planning, and what can and cannot be used to support the testing of that type of software. This provides a third reason why decision support for software testing has been a neglected area. All of the following could potentially require a different set of circumstances in the testing of software.

- System software – operating systems, compilers, interpreters, assembler etc.
- Utility software – used for developing, writing, debugging, and documenting programs.
- Application software – programs that users use on a computer to perform some task – word processors, spreadsheets, and database managers.

- Hardware type and configuration where the software is to be installed and used.
- Network type and configuration where the software is to be installed and used.
- Environment – setup and used in, including security issues.
- Target audience – knowledge and understanding, and access.
- Data – amount or/and type, where it comes from, format, etc.
- Documentation internal and external to the software that needs to be tested along with the software.

Decision support for software testing needs to consider the issues of the type of software to be tested. Overall this research has shown that due to the complexity and issues of software, there is very little that has been done in the form of decision support, or guidance, to assist the test manager in successful software testing.

Borysowich (2005) provided the following list of areas associated with types of software:

- size of the proposed system;
- size limitations of the proposed system;
- functional complexity of the proposed system;
- number of proposed system inputs;
- input data control procedures;
- number of proposed system databases;
- complexity of proposed system databases;
- number of proposed system outputs;
- output information control procedures;
- batch/on-line nature of proposed system transactions;
- number of programming languages used;
- difficulty of use for the chosen programming languages;
- suitability of programming languages to the software project;
- number of Application Programming Interfaces (APIs);
- estimated number of API variables/function calls;
- number of Dynamic Link Libraries (DLLs);
- estimated number of DLL calls;
- data transport mechanisms;
- diversity of targeted software platforms (Win 95, Win NT, etc.);

- diversity of targeted hardware platforms (Compaq, IBM, etc.);
- status of targeted platforms;
- familiarity of development team with targeted hardware and software platforms;
- level of application partitioning;
- level of process replication;
- server status, type, and use;
- database types, size, complexity, and type of use;
- level of data replication;
- number of stored procedures;
- database performance requirements;
- proposed network structure;
- network performance requirements;
- system interfaces;
- system infrastructure complexity;
- user documentation types, location, complexity;
- user training materials and type of training;
- on-line documentation and help requirements;
- installation requirements;
- system reliability requirements; and
- system back-up requirements (Borysowich, 2005).

As can be seen from the list above, the reasons for neglected areas of decision support in software testing are the type of software and the complexity within the types of software to be tested. Not all of these can be applied or are applied to all types of software, but they provide a foundation and reasons for the neglected areas in decision support for software testing.

2.9 Summary

Despite the array of techniques for preventing bugs and for facilitating bug detection, software development remains error-prone. Even with all the quality techniques practiced today, the lion's share of bugs are found by testing. Decision making, in the real world, continues to be an area where research into providing assistance is needed. Decision making and the real world are constantly changing and continue to be uncertain. This is especially true where structured, semi-structured or unstructured situations arise.

Software testing is an area where due to the types of software, decision making requires additional input in assessing successful software testing. It is in this area where there could be issues, problems, requirements and opportunities for the use of a decision support framework. Decision support frameworks to date have not been directed at software testing. Thus, due to the complex nature of today's software testing, the software test manager needs tools that are directed at assisting them in doing their job. A decision support framework for software testing is needed so that the software test manager can better plan before and during software testing. Risk management of successful software testing is an area where a decision support framework can provide the software test manager with information to help them do their job.

The literature review encompasses reference material from different areas, and several distinct, but coherent perspectives emerge from the analysis of that literature. Table 2.1 shows these perspectives.

Table 2.1: Perspectives of the relevant literature

Perspective	References
AI techniques have been defined and discussed from various perspectives in the literature. AI techniques for decision support have been addressed by various authors. From information derived from these references, there is little use of AI for decision support as it applies to the software testing process, or any subset thereof.	Bunke et al., 2004; Dick & Kandel, 2005; Groumpos, 2010; Kalogirou, 2007; Krishnamoorthy & Raveev, 1996; Liu & Satur, 1999; Mateou et al., 2005; Mu et al., 2004; Papageorgiou et al., 2003; Raza, 2009; Rees et al., 2001; Russell & Norvig, 2003; Sadiq et al., 2004; Salmeron, 2010; Siraj et al., 2001; Słowiński, 1992; Tsadiras, 2008; Wooff et al., 2002; Xirogiannis & Glykas, 2007.
The neglect of decision support tools, frameworks, and approaches for software testing has provided various perspectives on the testing of software. Authors have taken these perspectives and identified the areas of software that can illustrate the complex factors to consider in software testing.	Bangia, 2007; Borysowich, 2005; Dhunna & Dixit, 2010; Dustin, 2003; Holtsnider et al., 2010; James, 2009; Jones, 2007; Kayne, 2011; Leffingwell & Widrig, 2000; Myers, 2004; Oak, 2011; Patton, 2006.

Decision support has had extensive research for many years. This research has been directed towards one facet or another, such as the decision maker within the broader field of decision support, or understanding what makes something decision support, or a broad interpretation of decision support via the organisation. Herein there is insufficient research that has been directed towards software testing.	Alter, 1980, 2004; Keen & Scott Morton, 1978; Sprague & Carlson, 1982; Turban et al., 2011.
An understanding of decision support and its development of knowledge has been discussed from various perspectives. Most perspectives are directed towards the decision maker using the knowledge to improve their decision making. None of the perspectives taken has illustrated that decision making can be improved within the software testing environment.	Alter, 1977, 2002; Hayen, 2006; Khosrowpour, 2000; Power, 2001; Wang et al., 2001; White, 2002.
The use of automated testing techniques has been long discussed from various perspectives, such as the balance needed between automated testing and manual testing. Despite this range of perspectives, automated testing has seemingly not been considered from the viewpoint of decision support.	Dustin et al., 1999; Farrell-Vinay, 2008; Schuster et al., 2000; Weiss et al., 2009.
Decision support has been looked at in and around software testing, but mostly from the planning for system tests. The need for decision support for software testing has been noted, but research along this line appears to be scarce.	Ammann & Offutt, 2008; Engström & Runeson, 2011; Kaner et al., 2002; Mosley & Posey, 2002.
Frameworks for decision support have been done from various perspectives and in various areas, such as manufacturing and service networks. There is little evidence that a decision support framework exists for software testing.	Kahai et al., 2005; Légaré et al., 2006; Lendermann et al., 2005; Strauss et al., 2009.
There are a variety of metrics that are used during the testing of software, and after completing the testing of software. The value and use of these metrics has been noted. But the use and relationship of software metrics to decision support in software testing has had little research.	Afzal & Torkar, 2008; Bradshaw, 2004; Chen et al., 2004; Hass, 2008; Limaye, 2009; Mathur, 2008; Rivers & Vouk, 1999.

Risk management has been defined from various perspectives. Risk management has been applied to various aspects of business during software testing. Risk management within the software testing environment has had several views. None of these views have been formalised or incorporated within a software testing framework.	Bannerman, 2008; Charette, 2005; Craig & Jaskiel, 2002; Dedolph, 2003; Dustin, 2003; Maidasani, 2007; Redmill, 1999; Silva, 2008; Tamres & Mills, 2002.
Test planning is essential for successful software testing, and is an important responsibility of the software test manager. Research on high level test planning, consonant with this research, is apparently lacking.	Ammann & Offutt, 2008; Engström & Runeson, 2011; Kaner et al., 2002; Mosley & Posey, 2002.

Each perspective in Table 2.1 has two distinct parts: a description of the perspective and an analysis of the perspective. The analytical part of each perspective was extracted, and all these parts were modestly rearranged to reveal the following salient points:

- the need for decision support has been suggested for software testing, but research addressing that need seems to be scarce;
- the software testing environment consists of a set of complex factors that software testing should consider;
- the extensive research on decision support research does not appear to have been directed towards software testing, nor has that research shown that decision making can be improved within the software testing environment;
- there is seemingly limited research on decision support for software testing. AI techniques have not been used much in decision support for software testing, the use and relationship of software metrics to decision support in software testing has received scant research, and automated testing has not been considered from the perspective of decision support;
- there is little evidence that a decision support framework exists for software testing;
- risk management, in a software testing context, has not been addressed by a software testing framework; and
- research on high level test planning, consistent with the definition of test planning used in this research, is apparently lacking.

Remember that the points extracted, and rearranged above, represent analytical views of the descriptive part of each perspective shown in Table 2.1. Analysis of these extracted points indicates that they can be reduced (or collapsed) to give two results. Therefore, analysis of the synthetic view provided in Table 2.1, exposes two results. First, the perspectives are apparently unrelated. Second, there is an unmet need for a mechanism to integrate the different perspectives, and provide decision support in the areas of test planning and risk management for the software test manager.

- ||| -

CHAPTER 3. RESEARCH FOUNDATION

The following diagram provides an outline and overview of the conceptual structure of this chapter.

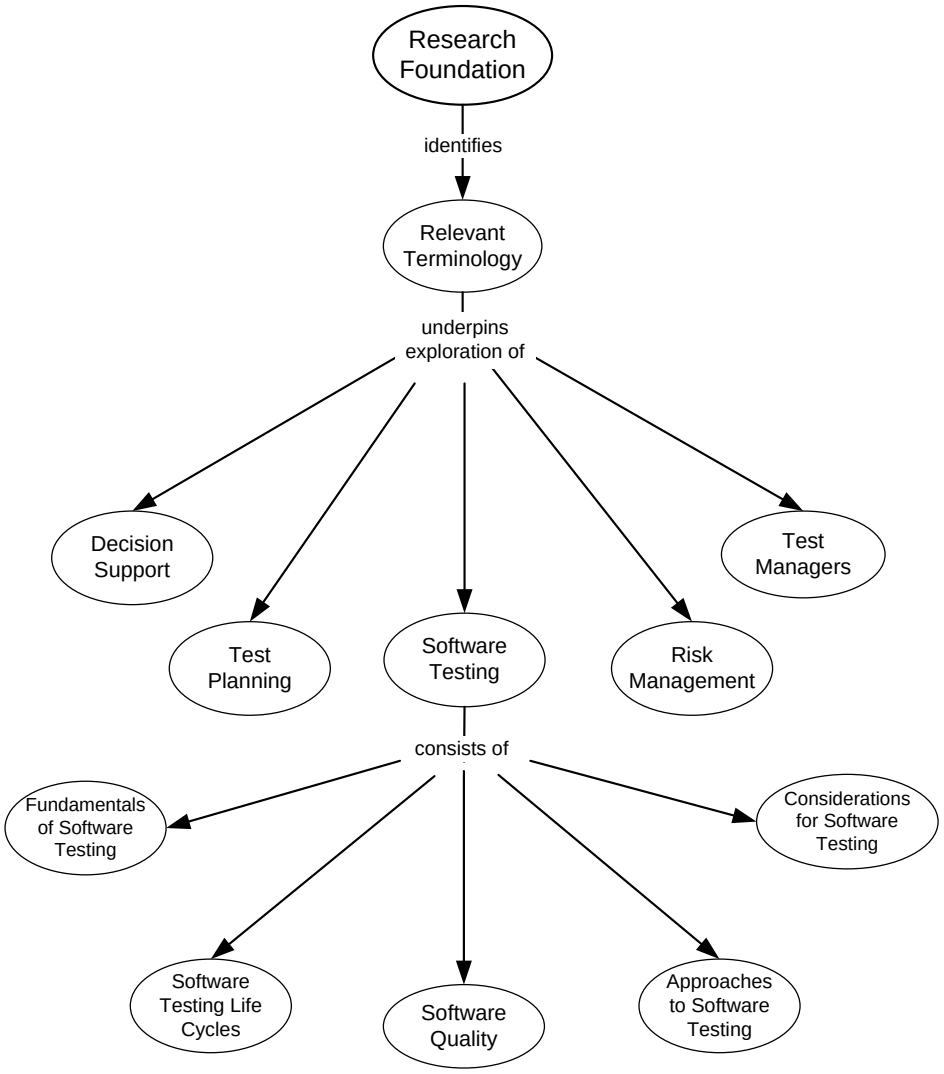


Figure 3.1: Conceptual structure of Chapter 3

3.1 Introduction

This chapter discusses the research environment, conditions and surroundings that establish the foundation and the context for this research. The research foundation lays the groundwork for a better understanding of the research boundary and what is within the boundary, and outside the boundary. The research foundation is identified by specific background material, terminology that is defined for this research, and different perspectives on the elements in this research.

The research terms defined and used for this research are developed through analysis of the relevant literature. However, the definitions of the research terms are consciously directed at this research, and clearly identify the perspective of the author. The definitions are explicit, and identified by being presented separately from the surrounding text, indented and formatted in italics. Furthermore, each research term definition is appended with the main references that were the basis for building the definition. The research terms (supported by their definitions) are used throughout the thesis unless specified otherwise.

3.2 Research Terminology

Explicit definitions of the relevant terms used in this research are important for several reasons. Definitive terminology clarifies and confirms the author's position, helps avoid circular definitions of terms, and helps to facilitate the reader's understanding of the foundation for this research. The terminology and concepts presented in this chapter are used consistently throughout the thesis. The meanings of the terms discussed in the background material are described, and the concepts or issues underlying each term are discussed.

3.3 Software Testing

The complexity of software based products, shortened development cycles, and higher customer expectations of quality have placed a major responsibility on the areas of: software debugging, software testing, and product verification (Hailpern & Santhanam, 2002). Thus, more and more organisations are relying on software testing to find and report on issues in order to deliver better software products.

Mantere (2003) states that for software testing, the gap between the state of the art and the state of practice is exceptionally wide (Mantere, 2003). The differences between software testing goals in the research community and industrial test practices are emphasised in that gap (Mantere, 2003). According to Bertolino (2007) software testing in industry is still largely ad hoc, expensive, and unpredictably effective (Bertolino, 2007). These thoughts are largely echoed by Juristo, Moreno and Strigel (2006), who state that testing practices in industry generally are not very sophisticated or effective (Juristo, Moreno & Strigel, 2006). It has been observed that technology transfer between testing research and industry is insufficient (Juristo et al., 2006). Industry usually has practical problems that are too specific or technical to be targets of the scientific research conducted in academia (Mantere, 2003). Furthermore, it is claimed that in industry, software testing does not receive the same attention as software development (Juristo et al., 2006). Some well researched methods (such

as formal analysis of code) are too complex and impractical to be used in many of the real industrial software testing environments (Mantere, 2003).

3.3.1 Fundamentals of Software Testing

This section begins with a definition of software, software systems and software testing. Within these definitions, brief discussions of quality assurance, verification, validation, and reliability, as they relate to software testing, are presented. These definitions establish an essential foundation of understanding for the research and its boundary.

The following software definition (Institute of Electrical and Electronics Engineers [IEEE], 1990) includes both major categories of software (application software and system software), views software as a product, and adopts the Brooks (Brooks, 1995) perspective of the software product.

> ***Software:*** *computer programs, associated documentation, and data used to perform some task or tasks by a computer, or by other electronic devices (Brooks, 1995; IEEE, 1990).*

Software is often referred to as source code, which is translated into machine readable code (also known as object code). However, source code and machine readable code are just different forms of computer programs; a set of instructions and data definitions constructed by programmers (IEEE, 1990). Software definitions may mention procedures. In this context procedures are named elements of computer programs that perform a specific action (IEEE, 1990). One or more actions are required to accomplish a given task.

Brooks (Brooks, 1995) takes a software development perspective and views the software product as a generalised program that includes documentation. Documentation relates to anything that allows anyone to create, use, fix, maintain, and extend a software program. A more recent definition (Patton, 2006) of software is directed at the software product that is delivered to the end-user. According to Patton a software product includes samples and examples, setup, help files, readme file, labels and stickers, advertisements, user manuals, error messages, known issues, and product support information (Patton, 2006). This definition goes beyond the boundary of this research but does illustrate how extensively some authors view software.

Application software is another term that is often used interchangeably with software. Application software is a computer program designed for a specific purpose, to do work for end-users, or sometimes to do work for other application software ("Application", 2011; "Application software", 2011; IEEE, 1990). Application software is also referred to as a software application, an application program, an application or an app ("Application program", 2011; "Application software", 2011).

The term end-user has been raised and needs to be defined. The end-user is the person who uses or operates the software ("End user", 2010). The end-user (or just user) of the software is also known as the consumer ("Consumer", 2011), and can be a customer or a client, depending on the context in which the software is purchased and used. The end-user may or may not install the software. Software end-users can also be developers or testers, who operate the software for development and testing purposes respectively. End-users use the software and can also be asked by testers to test the software during its development. The end-user may be different from the purchaser of the software: the purchaser can be a person or organisation. The purchaser of the software is commonly known as the customer or client ("Client", 2011; "Customer", 2011).

The following definition of a software system should not be confused with system software. System software facilitates the operation and maintenance of a computer system. Examples of system software include operating systems and utilities (IEEE, 1990).

> ***Software System****: a set of coordinated, interacting software programs required to accomplish a specific function, or set of functions to achieve a specific purpose (Brooks, 1995; IEEE, 1990).*

A software system is a collection of interacting software programs and components organised to accomplish a specific function, or set of functions that may be required for some work or large task (Brooks, 1995; IEEE, 1990). A software component in a software system has certain characteristics not observed in a standalone software program. A software system component must be: written to ensure that all inputs and outputs conform syntactically and semantically to well-defined interfaces. These components are designed to use only a finite set of resources such as: memory, input and output devices, and computer time (Brooks, 1995). Testing a software system is an extensive activity because the testing must

accommodate the combinatorial growth of test cases, which result from the necessity to test one software component with other components in all expected known combinations (Brooks, 1995). Testing a software system is time consuming. The time required to find subtle issues that arise from the unexpected interactions of the software components can be very long (Brooks, 1995).

Software systems are pervasive in modern society, and people rely on software systems for many daily life, work and societal activities (Tian, 2005). Therefore, software systems must meet or exceed people's needs and expectations. To satisfy these needs and expectations, software must be of high quality to minimise the diverse consequences of software failure. Testing is an analytical way to help improve software quality and plays a critical role in ensuring that systems and devices meet the needs and expectations of users.

Software testing is technically and economically an important aspect of high quality software production (Mantere, 2003). Over the years comparisons of the cost of software testing with the development costs of a software project have been made by several authors. Recently, a comprehensive Innovative Defence Technologies (IDT) survey, which had 700 responses, asked what percentage of overall project schedule was spent on testing. Forty six percent of the responses said 30% to 50% was spent on testing, and a further 19% of the responses said 50% to 75% was spent on testing (Dustin et al., 2009). According to Dick and Kandel (2005), nearly half of a software project's resources is consumed by software testing, and up to 80% of resources are used for software testing in safety-critical projects (Dick & Kandel, 2005). Debugging, testing, and verification activities of software can account for up to 75% of the total development costs (Hailpern & Santhanam, 2002). Beizer (1983) observed that software testing consumes half the labour needed to produce a working program (Beizer, 1983). Despite the slightly different perspectives of these authors, it is clear that the time and cost of software testing is a significant portion of a software project's overall cost and schedule (Dustin et al., 2009).

An increasing number of industrial products contain embedded electronic devices (Mantere, 2003). As the software of these embedded systems grows in size and complexity, software problems become more difficult to resolve and occur more frequently (Mantere, 2003). The corollary of these observations is a heightened need for software testing to increase confidence in the final product.

> ***Software Testing:*** *a set of activities performed on software and software systems with the purpose of finding errors in that software and software system. The process is directed at quality assurance, verification, validation, and reliability of the software and the software system (IEEE, 1990, 2008).*

This software testing definition emphasises testing as a *set of activities*. Implicit in the definition is what software testing is, why we test software, the purpose of software testing and what errors are. These concerns and related concerns are discussed next.

Software testing identifies issues that result when software does not meet end-user requirements or expectations (Dustin et al., 1999). Software does not meet end-user requirements when it fails in some way after being executed. End-user requirements, at a high level, include function and behaviour: system function is what the system is intended to do, and system behaviour is what the system does to implement its function (Avižienis, Laprie, Randell & Landwehr, 2004). Software inspections are useful but are no substitute for testing executed software. The correctness of a program cannot be determined by simply inspecting the software source code (Dick & Kandel, 2005). Most software failures are rarely obvious, but are simple and subtle: and often it is hard to distinguish between what is a failure and what is not a failure (Patton, 2006).

Software is correct when its behaviour (as perceived by its users) implements its system function (Avižienis et al., 2004). Dustin (1999) observes that software works correctly when the software:

- produces correct output, given valid input;
- correctly rejects invalid output;
- does not hang or crash, given valid or invalid input;
- runs correctly for as long as expected; and
- behaves as specified (Dustin et al., 1999).

Dustin (2009) maintains that in general, there are four types of problems that can arise during testing of software:

- a component has a failure of some kind, causing the operation of the software to be aborted. The user interface will typically show that an error has occurred;

- the test execution produces incorrect results, which occur when the actual test results are different from the expected outcome. A component in the software system has processed data incorrectly, causing an error in the results;
- a component fails during execution but the user interface does not indicate that an error has occurred, known as a "false positive". For example, data is entered but is not stored in the database, yet no error is reported to the user; and
- the system reports an error, but processes everything correctly – the test produces a "false negative". An incorrect error message is displayed (Dustin et al., 2009).

The word “issue” is a general term that includes a wide range of testing conditions that can occur within software or a software system. Issues are things that are inconsistent, incomplete, inappropriate, or do not conform to good practices of the intended application. An issue can be:

- some type of problem with the way the software acts;
- an installation requirement not being met (this could be the software, the hardware that the software is being installed on, disk space, memory needed, etc…, it could even be that the installation does not affect the security posture of the system or other installed software;
- a coding logic error;
- an error as a result of the code;
- a fault in the user interface;
- an unacceptable output;
- a condition that is not consistent with an intended requirement;
- an unacceptable work flow;
- a problem with the security posture caused by the software; or
- an adverse effect on some other software or software system.

IEEE uses the word “anomaly” to refer to conditions that deviate from requirements and/or expectations of the software. It is noted that an anomaly (issue) is not necessarily a problem in the software product (correct behaviour may be manifested), and an anomaly (issue) may be caused by something other than the software being tested (IEEE, 1990).

Patton (Patton, 2006) maintains that the most generic terms for software failure include bug, problem and error. Failure is used by Patton in a broad sense, and software failures can be seen to encompass all the issue conditions outlined above. Other terms for failure listed by Patton include defect, fault, inconsistency, incident, and anomaly (Patton, 2006). The term bug is commonly used in industry, but has attracted criticism in academic circles for its lack of a clear and consistent definition. Recently, Patton has offered a well thought out definition for bug. Patton says that a software bug occurs when one or more of the following rules are true (Patton, 2006):

- the software does not do something that the product specification says it should do;
- the software does something that the product specification says it should not do;
- the software does something that the product specification does not mention;
- the software does not do something that the product specification does not mention but should; and
- the software is difficult to understand, hard to use, slow, or – in the software tester's eyes – will be viewed by the end user as not right (Patton, 2006).

Terms for the components of failure exist in the software dependability community, and IEEE definitions (IEEE, 1990) are consistent with this approach. These terms include the following.

- *Mistake*. A human action that produces an incorrect or unexpected result.
- *Fault*. An incorrect step, process, or data definition in a computer program. The manifestation of a mistake.
- *Failure*. The inability of a system to perform its required functions, thereby giving an incorrect or unexpected result. The result of a fault.
- *Error*. The amount by which the result is incorrect or deviates from expectation. The difference between a computed, observed or measured result and the correct result.

The purpose of testing software is to identify issues, identify them as early as possible, and ensure that as many issues as possible are addressed before the software is released. Testing should also demonstrate and allow the organisation to gain confidence in the correct behaviour or function of the software (Frankl, Hamlet, Littlewood & Strigini, 1998; Hermadi, 2004; Patton, 2006; Tian, 2005; Wegener, Baresel & Sthamer, 2002). Testing cannot assure software quality (Kaner et al., 2002) but has the potential to produce software of a higher quality, which is the ultimate purpose of software testing (Beizer, 1990).

Software testing involves the execution of software in controlled experiments, and the observation of its behaviour (Patton, 2006). If an issue is observed, the execution record is analysed to locate and fix the associated issue that caused the condition.

In software testing, the goal is to design a minimal set of test cases to reveal as many issues as possible (Hermadi, 2004). That is, the tester has to select inputs which show that the software does not work: or to put another way, the tester is searching for inputs that are counterexamples to the assumption that the software works (Marick, 1995). A test case is developed to relate to one issue (or sometimes several issues), thereby testing an input and its expected output (Hermadi, 2004) for the software under test. A test case is successful if it detects an issue with the software (Hermadi, 2004) or confirms that the software meets its requirements.

> ***Test Case:*** *a set of values linked to requirements, execution conditions, expected results, and any values necessary for a tester to completely execute and evaluate the software or software system (Ammann & Offutt, 2008; Hermadi, 2004; IEEE, 1990).*

Software testing comes with some limitations. Software testing remains labour intensive and time consuming (Hermadi, 2004), despite technological advances: these characteristics can be viewed as limitations. Testing can show the presence of software issues, but cannot establish the absence of issues (i.e. issues may still exist in the software), due to the number of permutations there is no way to identify every issue or the absence of issues. Moreover, just because all the tests of software are passed it does not mean that there are no issues with the software. A theoretical limitation is that, in general, software testing cannot find all the issues in a program (Ammann & Offutt, 2008; Dick & Kandel, 2005; Patton, 2006). There are several reasons for this inability to test software exhaustively, or the inability to enumerate the inputs for reasonably sized programs (McMinn, 2004). The number of potential inputs and outputs is very large, the number of paths in software is very large (a path is the sequence of instructions performed to execute the program) (IEEE, 1990), and the software specification is subjectively constructed and subjectively interpreted (Patton, 2006). Therefore, the number of test conditions is very large, so that there is insufficient time to execute all the test conditions.

Because software cannot be tested completely, a decision has to be made about what features to test, and what features not to test, which invokes risk (Patton, 2006). Software testing needs sufficient resources to effectively test the software; and these resources are constantly changing or diminishing. Consequently, a software tester needs to make careful risk based decisions before and during software testing. The increasing complexity of software is, more often than not, in conflict with the need to minimise development costs, and achieve release schedules.

Software specifications are constantly changing, and software features will often alter as a consequence of specification changes (Patton, 2006). These changes affect resources and come with a risk to successful software testing. This means that software testing has to be flexible, especially in the test plan and test execution phases (Patton, 2006).

The major source of correct behaviour expectation is the requirements specification (Hailpern & Santhanam, 2002; Patton, 2006). Some other sources could include previously verified software products, reference documents, or standards (IEEE, 1990, 2010).

The definition herein of software testing includes quality assurance, verification, validation, and reliability. To more fully understand software testing, these terms are briefly discussed below.

> ***Quality Assurance (QA):*** *the process of ensuring that software or a software system and its associated documentation are in all respects of sufficient quality for their purpose (Daintith, 2004a).*

Quality Assurance (QA) is intrinsic to the software development life cycle, and occurs throughout all facets of that life cycle, including the entire software testing life cycle. It is an underlying process whereby measures are made to ensure that the software is designed and developed to meet or exceed software requirement specifications.

Testing is an important component of quality assurance, and is the most commonly performed quality assurance activity (Tian, 2005). Tian (2005) views validation and verification as distinct QA activities from testing. Testing is neither the only viable or most effective QA technique under all circumstances (Tian, 2005).

> ***Verification****: is the process to determine if the products of a given software development phase satisfy the conditions established before the start of that phase (Ammann & Offutt, 2008; IEEE, 1990; Naik & Tripathy, 2008).*
>
> ***Validation****: is the process to determine if a software product, at the end of its development, meets its intended use: where intended use is defined by requirements and expectations of the user (Ammann & Offutt, 2008; IEEE, 1990; Naik & Tripathy, 2008).*

Verification and validation are viewed as two basic forms of testing aimed at quality assurance (Mantere, 2003). Verification is seen as human testing because it involves examination of documents, specifications, and code on paper (Mantere, 2003). Verification is similar to static white box testing (Mantere, 2003). Validation is seen as computer based testing because it normally involves execution of the software (Mantere, 2003). Validation is similar to dynamic black box testing (Mantere, 2003).

Verification and validation are related. Both are used to test and ensure software quality and reliability. The concepts that underlie these two processes are similar and need to be clearly distinguished. Verification is the process to check the correctness of a software product from a particular development stage against the requirements imposed before the start of that development stage (Ammann & Offutt, 2008; IEEE, 1990; Naik & Tripathy, 2008). The software products can be intermediate products (such as requirements specification, design specification, code or user manual), or even the final product (Naik & Tripathy, 2008).

Validation examines whether a software product meets its intended use (Ammann & Offutt, 2008; IEEE, 1990; Naik & Tripathy, 2008). The intended use of a product is defined by the customer or user requirements and expectations (Naik & Tripathy, 2008). Validation focuses on the final software product, and is usually dependent on knowledge of the domain for which the software was written to be used in (Ammann & Offutt, 2008).

Boehm observes that verification is about building the software product correctly, and validation is about building the correct software product (Boehm, 1981).

Over time, software has been known to have unforeseen issues or not adapt to its intended use. This is often referred to as software reliability.

> ***Software Reliability:*** *The probability of failure-free software operation for a specified period of time in a specified environment (Daintith, 2004b).*

Software reliability can include the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems over some period of time.

Software testing raises several other concerns that warrant comment. Software is very complex: some large software systems can have 10^{20} or more states (Dick & Kandel, 2005). Therefore, software testing can be non-exhaustive, and will most likely remain that way (Dick & Kandel, 2005). Consequently, software test cases are only samples of the complete input space of a program (Dick & Kandel, 2005). A program state is a snapshot into the state of an executing program on some input (Friedman & Voas, 1995). The program state is all of the information needed to restart an execution of the program (Friedman & Voas, 1995).

Issues, of one type or another, are often present in software even though it has been thoroughly tested. The issues reported by users in a software product could be because the user executed untested or incompletely tested code, the order in which statements were executed by the user was different from that during testing, the user applied a combination of untested input values, or the user's operating environment was never tested (Whittaker, 2000). Handling of issues is important to ensure that failed test cases will not block the execution of other test cases (Tian, 2005).

Issue removal (i.e. fixing or repairing the issue and potentially related issues) is not necessarily straight forward. There are several possibilities to explain this. The issue may have not been incorrectly identified. Issue removal may have only partly removed the fault, or even introduced new faults. Another alternative is that the detected issue could be masking another issue. Therefore, complete removal of one issue could expose another, possibly more serious one (Dick & Kandel, 2005). So the successful removal of one issue could make the software system less reliable than it was previously (Dick & Kandel, 2005).

An essential part of validating if an issue has been removed is regression testing, where a subset of the test suite is re-run to determine if the test cases that executed correctly before the repair, still execute correctly after the repair (Dick & Kandel, 2005).

3.3.2 Software Testing Life Cycles

The software testing life cycle fits into one part of the system development life cycle (SDLC). The SDLC consists of a set of steps that help guide software managers in developing applications. A definition of a generic SDLC was developed for this research.

> ***System Development Life Cycle****: a series of steps that form a process in creating or altering systems into existence that includes models and methodologies that people use to develop systems (Baltzan, Phillips, Lynch & Blakey, 2010; Dennis, Wixom & Roth, 2006; Hoffer, George & Valacich, 2011; Kendall & Kendall, 1998; Shelly & Rosenblatt, 2010).*

The generic SDLC described in Table 3.1 is based on several books (Baltzan et al., 2010; Dennis et al., 2006; Hoffer et al., 2011; Kendall & Kendall, 1998; Shelly & Rosenblatt, 2010)

Table 3.1: Generic system development life cycle

Phase	Activities	Outcome
Requirements gathering and analysis	Establish a high level of the project, define the requirements, and analyses the user needs	Initial requirements documentation
Planning	Identify resources needed and scheduling options.	Development plan, resource allocation plan.
Design	Describe features and operations in detail, including screen layouts, business rules, process diagrams, and pseudo code	Document design features, relate back to the requirements documentation
Develop and code	Code written	Code written and documented
Testing	Check software for issues including errors, bugs, and interoperability	Tested software ready for release and deployment
Deploy and maintain	Release software	Software put into production, ongoing support for released software

Software testing has its own life cycle that intersects with every stage of the SDLC. The basic requirements in the software testing life cycle are to control and deal with the testing of software, including manual and automated testing (Editorial, n.d.). The scope and methodology may vary from product to product, customer to customer, and organisation to

organisation (Jaideep, 2008b). This section presents a generic software testing life cycle (STLC) that will be referred to in later chapters in support of the material discussed in those chapters. The definition of the generic STLC developed for this research is as follows.

> ***Software Testing Life Cycle****: a set of steps or phases that identifies what test activities to carry out and when to accomplish those test activities (Editorial, n.d.; Jaideep, 2008b)*

The following table outlines a generic STLC (Editorial, n.d.; Jaideep, 2008b). Other perspectives on generic STLCs exist, some with fewer phases, some with more phases, and some with extensive details. For the purpose of this research the following table has been used.

Table 3.2: Generic software testing life cycle

Phase	Activities	Outcome
Planning	Approach to testing, high level test planning, development of a test plan, review of high level design, review of requirements, high conduct level risk management assessment	Initial development of a test plan, refined specification, initial risk assessment
Analysis	Detailed test plan, functional validation matrix	Updated test plan, initial requirements verification matrix, initial test cases
Design	Test cases developed, select which test cases to automate, risk assessed detailed, test environment established	Test cases documented, test data sets obtained, risk management issues assessed and documented, test environment documented
Conduct testing	Scripting of test cases to automate, begin testing of the software	Test procedures/scripts, drivers, test results, bug reports, and updated requirements verification matrix
Testing cycle	Complete testing cycle(s) as needed	Test results, bug reports, regression testing
Final testing	Execute remaining stress and performance tests, complete documentation	Test results and different metrics on test efforts
Release tested software	Document release, evaluate testing processes	Plan for improvement of testing process

Table 3.2 indicates that the final phase of the generic STLC is to release the tested software. However, software release (as well as software release planning) is not part of the STLC per se. Software release is more of a SDLC than a STLC item (Richardson, O'Malley, Moore & Aha, 1992). After software has been fully tested, test management writes a report, and the software is given back to software development or the area within the organisation responsible for it (Kerkhoff, 2002; Richardson et al., 1992). We will discuss the test management report, and related issues first. The test management report is about reporting any outstanding issues (such as severity of bugs, and workarounds and information that could go into the release notes) (Kerkhoff, 2002; Richardson et al., 1992). Management of the organisation must make a decision to release the software or not. The recommendation comes from the System Development Manager or Operations Manager – whoever is ultimately in charge of delivering the product (releasing the product to customers or the wider community) (Kerkhoff, 2002; Richardson et al., 1992). The test manager may be asked for his/her recommendation, but the final decision is rarely theirs (Kerkhoff, 2002).

Now we will explore what happens when the fully tested software is handed back to the responsible area of the organisation. The area responsible for the software completes several tasks: such as all the paperwork, making appropriate copies of the software to be safeguarded, backing up and locking the development environment, locking up the release number or system number, and producing the software release and marketing (or customer report) information (Kerkhoff, 2002; Richardson et al., 1992). Rarely would the preceding tasks be done by the testing department (Kerkhoff, 2002; Richardson et al., 1992).

3.3.3 Software Quality

The consequences of software failure are related to software quality. Poor quality software inevitably leads to failure of one type or another; and the severity of the consequences depends on the type of failure. Although the quality of engineering products is a seemingly straight forward intuitive concept (we *know* intuitively what quality is when we see it), product quality is hard to define exactly (Dick & Kandel, 2005). Software is a non-physical product, a logical and very flexible human construct. A technical product quality definition is the "degree of excellence" of the product, taking account of all relevant characteristics of the product (Dick & Kandel, 2005). Some of a system's characteristics that contribute towards overall quality are reliability, functionality, performance, maintainability, conformance,

portability and usability (Dick & Kandel, 2005; Garvin, 1984; Kitchenham & Pfleeger, 1996). Quality is an evaluation of a product in its entirety (Dick & Kandel, 2005).

There are multiple issues related to software quality. Some of these issues include quality of design, quality of conformance, and software quality assurance. For this research software quality and its various issues are left to the test manager and are not part of this research.

3.3.4 Approaches to Software Testing

This research is not concerned with which approach is used in software testing. However, it is important to note the different approaches employed in software testing, which depend on the type of software being tested, combined with the type of testing being employed. In particular, the number of different approaches discussed in literature. It is also important to note that no matter which approach is used, risk management is an area that needs to be addressed.

There are many approaches that have been used in testing of software that have been identified in literature. A list of some of the approaches is given in Appendix A.

3.3.4.1 Testing to Pass and Testing to Fail

There are two basic approaches to software testing: test to pass – where the software is tested under normal operating conditions; and test to fail or error forcing – where an attempt is made to break the software (Patton, 2006). Test to pass tests are performed first to see if the software works as specified, under normal conditions (Patton, 2006). A class of test cases aim to force error messages, and the test to pass/test to fail distinction is unclear, and unimportant for these test cases (Patton, 2006). Test cases should be designed to force errors identified in the specification, and errors not considered by the specification (Patton, 2006).

Some industrial experience indicates that the above two part approach to software testing is not used in practice (Patton, 2006). Software developers will not know, or are not allowed to know the actual user conditions for the software. Thus, a test to pass approach is difficult to achieve. The test to fail approach seems to be more common, where the software is tested against the formal requirement specification, followed by ad hoc testing (Meyer, 2008; Patton, 2006). Ad hoc testing is unstructured testing performed without planning and documentation, intended to run only once, unless defects are found ("Ad hoc testing", 2011).

Software testing has four basic strategies (Patton, 2006).

- *Static black box testing.* Involves an examination of the specification, with the aim of finding problems before they are written into the software.
- *Dynamic black box testing.* Involves testing the software without knowing how it works (testing without looking at the code).
- *Static white box testing.* Involves an examination of the details of the code through formal reviews and inspections.
- *Dynamic white box testing.* Involves testing the software when you know how it works, and basing test cases on that information.

3.3.4.2 Static and Dynamic Testing

Static testing is testing something that is not running – examining and reviewing it (Patton, 2006). Dynamic testing tests software that is executed and used as a customer would (Patton, 2006). Static testing detects faults directly through examination of the software specification, design, architecture or code. In comparison, dynamic testing detects faults indirectly through the execution of software (Tian, 2005).

Typical static analysis methods include tests and reviews of the specification, formal reviews of the code through code inspections and code walkthroughs, and informal peer reviews of the code. The aim of white box testing is to exercise the various logic structures and logic flows in the software. In black box testing, the data, and the logic flow through the various software states is tested to assess the software functionality.

3.3.4.3 Static Black Box Testing

Static black box testing is testing the specification for errors. The specification is not a running program; it is usually a static document (Patton, 2006). The specification can be in any format: written, graphical, a combination of written and graphical, or unwritten (Patton, 2006).

The first step is to perform a high level review of the specification. The aims of the review are twofold. First, to achieve a better understanding of the specification, as a preliminary to a detailed examination of the specification (Patton, 2006). Second, to identify oversights and omissions in the specification (Patton, 2006). Important aspects of a high level review of the specification are as follows. Understand the customer's expectations of the software, which will be assisted by gaining some familiarity with the field to which the software applies

(Patton, 2006). Research existing standards and guidelines to find out how they might apply to the software (Patton, 2006). For example, applications of standards and guidelines to the software could include: corporate terminology and conventions, industry requirements, government standards, and security standards (Patton, 2006). Review and test similar software to achieve a better understanding of the software described by the specification (Patton, 2006).

After the high level review of the specification, the specification is tested at a low level (Patton, 2006). Low level testing of the specification helps ensure that the details are defined (Patton, 2006). Two complementary checklists facilitate this. A well-developed specification is characterised by eight attributes: complete; accurate; precise, unambiguous and clear; consistent; relevant; feasible; code-free; and testable (Patton, 2006). If the detailed material in the specification does not have these attributes, an error has been found (Patton, 2006). A checklist of words that often signify a problem in the specification complements the attributes checklist (Patton, 2006). Problem words include words that: are absolutes (always, never); are persuasive without explanation (certainly, obviously); are vague (sometimes, often); are unquantifiable (fast, small); hide functionality that needs to be specified (handled, rejected); describe incomplete "If…Then…Else…" clauses – where the "Else…"is missing; and indicate lists that are not testable, and so on (Patton, 2006). Any problem words in the specification need to be carefully examined in context. If the problem words identified in the specification are not made clear or explained further, an error has been revealed (Patton, 2006).

3.3.4.4 Dynamic Black Box Testing

Testing software without knowing how the software works (in the absence of an insight into the underlying code) is black box testing (Patton, 2006). For black box testing to be effective a definition of what the software does (requirements document or product specification) is needed (Patton, 2006). In black box testing, the data, and the logic flow through the various software states is tested to assess the software functionality (Patton, 2006). An understanding of the software functionality is required – knowledge of the expected outputs for the corresponding inputs or operations (Patton, 2006).

3.3.4.5 Static White Box Testing

Static white box testing involves formal reviews, which encompass peer reviews, walkthroughs, and inspections (Patton, 2006). Details about these formal review methods follow.

In formal reviews, problems or omissions in the code are sought (Patton, 2006). Software may operate properly after the problems are rectified, but problems may still exist because the code is not written to meet a specific standard or guideline (Patton, 2006). An analogy is to write words that convey a meaning that can be understood, but the words do not meet the grammatical and syntactical rules of the English language (Patton, 2006). Standards are established, fixed, obligatory rules. Guidelines are the suggested best practices, the preferred way of doing things (Patton, 2006).

A formal review should ensure that the code meets the software design requirements, and meets a specific standard or guideline (Patton, 2006). In addition, some generic code problems should be looked for when the code is verified in a formal review (Patton, 2006). A checklist of generic code problems follows.

- *Data reference errors.* Data reference errors occur when a data type (such as a variable, constant, array, string or record), has not been properly declared or initialised for how it is being used and referenced (Patton, 2006). Data reference errors are the primary cause of buffer overruns – the errors underlying many software security issues (Patton, 2006).
- *Data declaration errors.* Data declaration errors are caused by the improper declaration or use of variables or constants (Patton, 2006).
- *Computational errors.* Computational errors are calculations that do not give the expected result (Patton, 2006).
- *Comparison errors.* Comparison errors occur when the comparison operators (such as less than, greater than, equal, not equal, true, or false) are used incorrectly in comparison expressions (Patton, 2006). For example: incorrect choice of the comparison operator; no allowance for precision in comparisons; comparisons between different data types; and incorrect order of evaluation in a Boolean expression (Patton, 2006).
- *Control flow errors.* Control flow errors result from loops and other control structures in the language not behaving as expected (Patton, 2006).

- *Subroutine parameter errors.* Subroutine parameter errors result from the incorrect passing of data to and from subroutines (Patton, 2006).
- *Input/Output errors.* Input/output errors are caused by incorrect inputs to the software, or incorrect outputs from the software. These errors could, for example, be related to reading from a file, accepting input from a keyboard or mouse, and writing to output devices like the printer or screen (Patton, 2006).
- *Other checks.* Other checks, expressed as questions, include: Does the software work with other languages besides English? Does the software work with different character encodings (like extended ASCII and Unicode)? Is the software portable to other CPUs and compilers? Has compatibility been addressed so that the software will operate with, for example, different amounts of available memory, different CPU internal hardware, and different peripherals? (Patton, 2006).

3.3.4.6 Dynamic White Box Testing

Dynamic white box testing tests software using information from knowledge of the code, and test cases are determined based on that information (Patton, 2006). The aim of dynamic white box testing is to exercise the various logic structures and logic flows in the software. Dynamic white box testing covers four areas (Patton, 2006):

1. direct testing of low level functions, procedures, subroutines, or libraries;
2. testing the software as a complete program, but adjusting the test cases based on knowledge of the software's operations;
3. access to read variables and state information from the software to help determine if test cases are working as expected. The extra benefit of this access is the ability to force the software to operate in ways that would be difficult to achieve if it was tested in any other way; and
4. measuring what code is exercised, and how much is exercised when test cases are executed. Then adjusting the test cases to remove redundant test cases and add missing test cases.

Dynamic white box testing and debugging (a programming technique) should not be confused. Dynamic white box testing and debugging appear to be similar techniques, but they have different goals (Patton, 2006). The goal of dynamic white box testing is to find software errors. The goal of debugging is to fix software errors (Patton, 2006). However, white box

testing and debugging do overlap in one respect. They both *locate* software errors, and determine why the errors occur (Patton, 2006).

3.3.5 Testing Considerations

Test considerations include both the input and output characteristics of the software as well as the inclusion of an appropriate method or approach used for testing the software. The discussion of the appropriate testing method or approach is not part of this research.

Test case selection can be facilitated by equivalence partitioning. Equivalence partitioning is the process of systematically reducing the very large number of possible test cases to a much smaller and manageable, but equally effective, set (Patton, 2006). Equivalence partitioning is a risk inherent process because a choice is made not to test everything (Patton, 2006).

To make software testable for data input, in light of the enormous amounts of data handled by even simple programs, the possible test cases are reduced by equivalence partitioning, based on the concepts of boundary conditions, internal boundary conditions, nulls, and invalid data (Patton, 2006).

The different data sets (for example: numeric, character, position and quantity) that the software operates on should be tested for boundary conditions (Patton, 2006). The characteristics of these data sets need to be considered: such as min/max, empty/full and first/last (Patton, 2006). It is a good idea to not only test the boundary values, but values on both sides of the boundaries (Patton, 2006). For any boundary condition, test the valid data just inside the boundary, and test the invalid data just outside the boundary (Patton, 2006).

Boundary conditions internal to the software are often not readily apparent to end-users (Patton, 2006). Internal boundary conditions require a general knowledge of how software operates, but do not require an ability to read code (Patton, 2006). Two types of internal boundary conditions are powers of two and ASCII character representation (Patton, 2006). Computers and software are binary number based. Powers of two terms (such as bit, byte, word, Kilo and Mega) represent values that are boundary conditions that need to be tested for as inputs to the software (Patton, 2006). The non-contiguous ASCII character set is relevant to software that accepts text entry, or performs text conversion (Patton, 2006).

The creation of an equivalence partition to test for default, null, or zero data should be considered. The previous equivalence partitions (boundary testing, internal boundary testing, and null testing) test software to work as expected (Patton, 2006). Invalid data is a way to test software for failure, and this is viewed by Patton as a useful real world technique (Patton, 2006).

3.4 Decision Support: the Role of Decision Making and Decision Makers

Decision makers in the real world operate in complex, dynamic and uncertain environments (Salmeron, 2010). The decision issues and problems in these environments are often characterised by situations that are not well structured (Salmeron, 2010). A decision support system is a class of information system that support business and organisational decision making activities (Alter, 2002; Turban et al., 2011). A systematic decision support tool can help decision makers in their decision making process. The decision support definition adopted by this research is:

> ***Decision Support:*** *any computerised or non-computerised means to assist an organisation's decision makers to improve their decision making within structured, semi-structured, or unstructured situations/issues/problems/requirements /opportunities (Alter, 2002; Salmeron, 2010; Turban et al., 2011).*

This definition of decision support requires a grasp of:

- what decision making entails;
- the concept of structured, semi-structured, and unstructured situations/issues/ problems/requirements/opportunities;
- the context of decision making from a real world perspective; and
- an idea of the real world issues faced by decision makers in their decision making process.

Structured, semi-structured, and unstructured situations refer to situations and problem structures associated with decision support. A structured situation consists of elements and relationships between elements where all or nearly all of the variables are well known and understood by the problem solver. Semi-structured situations contain some elements and relationships where some of the variables are known, and they are understood by the problem solver. Unstructured situations contain few to no elements or relationships that are well

understood, and the variables need to be addressed for each situation by the problem solver. Decision makers must show judgement in evaluating the situation, and must show insight into the situation (Kendall & Kendall, 1998; Skyttner, 2005).

Decision making is the process of developing arguments for alternative courses of action and choosing amongst alternatives, with the aim of achieving a goal or goals (Bohanec, 2001; Turban et al., 2011). The decision, in this research, is the choice of the preferred alternative, the solution to a problem or a requirement, or identifying an opportunity (Bohanec, 2001; Turban et al., 2011). The description of decision making just provided indicates clearly that decision making is a *process*. That process intrinsically involves problem solving, irrespective of whether the decision maker is faced with a situation that reflects a problem, a requirement or an opportunity (Bohanec, 2001; Turban et al., 2011). Problem solving deals with systems which do not meet their established goals, do not yield the predicted results, or do not work as planned (Turban et al., 2011). Problem solving may also deal with responding to requirements or identifying (or exploiting) new opportunities (Turban et al., 2011). This research considers the entire decision making process as equivalent to problem solving – a view adopted by Turban et al. (Turban et al., 2011), and notes that the terms "decision making" and "problem solving" can be used interchangeably. However, the term "decision making" is adopted throughout this thesis.

Decision making can proceed by informal deliberation, by applying an analytical technique or with the help of expert opinion (van Gelder, 2010). Deliberation is the careful consideration of options and related issues expressed as relevant arguments and evidence (van Gelder, 2010). Everyone has and uses informal deliberation – some type of decision making process, which may be habitual, unconscious, or default – the process automatically adopted when no other decision making process is selected (van Gelder, 2010).

Incorrect decisions are made all the time, and there is an urgent need to improve decision making (van Gelder, 2010). Informal decision making methods are very unreliable, and can lead to bad decisions (where the wrong choice was made), based on thinking that was clearly ill-informed, sloppy, disorganised, incomplete or biased (van Gelder, 2010). Analytical techniques for decision making employ a disciplined and systematic approach, and were partly developed to overcome the unreliability of informal methods (van Gelder, 2010).

However, even these analytical techniques are not exempt from the possibility of establishing bad decisions.

Decision makers operate in complex real world situations. These situations are constantly changing and are often very uncertain (Salmeron, 2009). Decision making in the real world is frequently dynamic and often constrained by limited or restricted resources (Salmeron, 2009). Real world decision situations exist in the context of systems, which are usually characterised by a number of interrelated and interacting concepts that evolve over time (Salmeron, 2009).

In the real world the decision making process poses several challenges for decision makers. Decision makers need to construct a representation of the decision problem, establish alternative courses of action, and imagine or calculate the outcome of choosing an alternative (Salmeron, 2009).

3.5 Test Planning

At least one of the steps in a software testing life cycle involves test planning. The generic STLC in Table 3.2, for example, has a planning step. Test planning is a very important (Ammann & Offutt, 2008), but difficult activity (D'Angelo, 2008). Test planning involves such things as assessing risk, identifying and prioritising test requirements, estimating testing resource needs, developing a project plan, and assigning testing responsibilities to members of the test team (Mosley & Posey, 2002). Ammann and Offut (2008) emphasised that it is important for test planning to proceed simultaneously with requirements analysis and design analysis, and not be delayed until late in the project (Ammann & Offutt, 2008). The output of the test planning step is the development of the test plan.

The test plan is an essential part of test planning. Test planning has been defined as simply: "figuring out what to do next" (Copeland, 2004, p. 183). The test plan is the set of ideas that guides the test process (Kaner et al., 2002). It is derived from the software requirements and linked to the test requirements and test results (Mosley & Posey, 2002). Test planning needs to consider such areas as: test resources, potential for automating testing, staffing, test requirements, test strategy, test design, infrastructure support, user needs, test environment, technical support, data requirements, expected operational environment, and management support (Ammann & Offutt, 2008; IEEE, 2008; Mosley & Posey, 2002). Mosley and Posey (2002) view the test plan as a dynamic, working document, created up front, and updated

when software requirements change so that the changes can be reflected in the test requirements (Mosley & Posey, 2002).

> ***Test Planning****: the activity of identifying and gathering the information that pertains to the particular software to be tested (Ammann & Offutt, 2008; Copeland, 2004; Mosley & Posey, 2002).*

In some cases authors refer to the test planning activity as the development of the test plan. The test plan has been discussed from various perspectives and within multiple contexts (Everett & McLeod, 2007; Hass, 2008; Ireland, 2005; Limaye, 2009; Sree, 2008).

3.6 Risk Management

Managing risk in software testing is considered to be a major contributor to product success (Bannerman, 2008). Risk is inherent in all forms of decision making, but can be minimised by the use of a framework specifically designed to address the need to control risk in software testing.

> ***Risk****: the possibility of harm or loss. Typically risks are future uncertain events with a probability of occurrence, a consequence that follows if the event occurs, and the consequence has a potential for loss and an associated cost (Boehm, 1991; Dedolph, 2003; Vijay, 2007).*

Software testing comes with risk: risk to the organisation, risk to the customers, risk to users, and risk to product success. Whatever tools that a software test manager can use to help them minimise and control risk in software testing is a valuable tool.

> ***Risk Management****: the identification, assessment, and prioritisation of a possible event or circumstance that can have negative influences on the meeting of an objective or goal in the testing of software (Boehm, 1991; Dedolph, 2003).*

This section draws from the investigation of decision support in Section 3.4. Issues from that section which are relevant to risk management are highlighted by re-stating them, and further discussed from a risk management perspective.

Decision making is the process of developing arguments for alternative courses of action and choosing amongst alternatives, with the aim of achieving some type of a goal or objective

(Turban et al., 2011). Decision making can proceed by informal deliberation or with the help of an analytical technique of one type or another (van Gelder, 2010). With deliberation in decision making comes risk. Deliberation is the careful consideration of options and related issues expressed as relevant arguments and evidence (van Gelder, 2010). If decision making proceeds only by informal deliberation, there will be risk that may compromise the decision making process, thus possibly leading to an incorrect decision. Therefore, the risk needs to be identified, assessed and understood. Once the risk is assessed and understood it can be controlled and minimised.

Risk management is necessary because incorrect decisions are made all the time, and therefore there exists a need to improve decision making (van Gelder, 2010). Informal decision making methods are very unreliable, and can lead to bad decisions (where the wrong choice was made), based on thinking that was clearly ill-informed, sloppy, disorganised, incomplete or biased (van Gelder, 2010). Risk management incorporates the use of analytical techniques that can help to overcome the unreliability of informal deliberation, and lead to better informed decisions (van Gelder, 2010). Nonetheless, analytical techniques are not immune from establishing bad decisions.

3.7 Software Test Managers

This section investigates software test managers from two perspectives to provide a clearer understanding of their place in the context of this research. The needs of software test managers, which are the foundation for their ability to test software and software systems efficiently, are examined in detail. Then the decision making role of software test managers is considered in terms of the demands that are placed on them.

3.7.1 The Software Test Manager

As technology continues to evolve, mature, change, and new technologies are added, the software test manager must constantly keep up to date to ensure that the testing efforts meet the challenges, and needs of the world of software testing. The roles and responsibilities may include such areas as developing test strategies, creating test plans, managing test teams, dealing with stakeholders, maintaining test metrics for reporting, developing test reports, developing test procedures, creating test cases, maintaining and supporting testing tools, conducting staff performance reviews, and understanding complex testing cycles (Johnson, 2011; SMS Management & Technology, 2011). The material above was partly based on issues identified on the Testing Institute website (http://www.testinginstitute.com).

As software is ever changing, so the role and responsibilities of the software test manager must change accordingly. For this research the following definition has been created based on position descriptions in position advertisements, and material available online from professional bodies, especially the Testing Institute (http://www.testinginstitute.com)

> ***Software Test Manager****: has to evolve and successfully respond to the diverse requirements of rapidly changing software technologies, and of complex software testing cycles, while they also have to effectively lead the test team, and manage, implement and maintain an effective testing process. The software test manager must be mindful of the planning, scheduling and infrastructure needs to support the test process, and be able to communicate clearly with all stakeholders at all levels (http://www.testinginstitute.com; Johnson, 2011; SMS Management & Technology, 2011).*

3.7.2 The Needs of Software Test Managers

The software test manager needs skills and experiences that are directed at them being able to do their job efficiently. This is inextricably related to the role, responsibilities, functions, and activities of the software test manager. From an analysis of the literature and advertised job descriptions for software test managers, various needs of the software test manager were identified. Some references used in this activity are advertised job descriptions. These job descriptions are taken from job positions advertised online. The job positions are only available for about two weeks to four weeks (depending on where the advertisements are sourced from, private industry or government). Therefore, the job positions are not recoverable beyond the limited availability of the advertisements. Alternative references that are recoverable for the longer term, and can be accessed by the reader anytime, have not been found. The needs of the software test manager include the following.

- *Skills***.** Basic professional skills such as reading and writing. Reading has to be analytical, focused and with meticulous attention to detail – required for unambiguous communication with management and testers and for understanding a variety of detailed reference sources (such as specifications, requirements and product documentation) (Black, 2002). Writing needs to be clear and effective, necessary for editing bug reports and for creating reports to be delivered to management (Black, 2002). Skills in the technology upon which the system was built – a grasp of programming languages, system architectures, networking, database functionality and

implementation and so on. These skills will enhance understanding of technology dependent bugs (Black, 2002). Skill with the application domain is required for a good understanding of what the system is meant to do (Black, 2002). A handle on the business, technical or scientific problem that the system is attempting to solve will provide a better idea of what a correct result would look like when the system is executed (Black, 2002). Skills in the testing process, which will augment the ability to identify the need for special skills. For example, the test team may not have the skills to perform certain tests, such as load and performance tests (Black, 2002).

- *Experience with the testing process and experience as a manager*. Testing process experience and managerial experience varies from one software test manager to the next software test manager.
- *Understanding of the requirements of the test artefacts*. Test artefacts are likely to have different characteristics that lead to different testing requirements (IEEE, 2008). Understanding software and software systems is necessary to facilitate informed decision making about test artefacts.
- *The ability to direct, control, administer plans and regulate the ongoing evaluation of the testing process and related artefacts*. The software test manager has to not only understand the testing process, but manage the testing process to achieve the best outcome (advertised job descriptions).
- *The ability to contend with organisational pressures***.** The software test manager is under significant pressure to ensure that the information they communicate to management, about software to be released, is factual and does not compromise the integrity of the organisation and its products.
- *Analytical and investigative ability*. This combined ability is necessary to develop test plans and to aid in making better informed decisions (advertised job descriptions). The software test manager has to evaluate different planning alternatives and provide decisions to management on the most appropriate one for the given software to be tested (advertised job descriptions).
- *The ability to measure a test process*. Test process measurement is useful for designing and evaluating the test strategy (Chen et al., 2004). Feedback from the analysis of test measurements helps management make better informed decisions about software issues for product release and quality improvement (Chen et al., 2004). Test process measurement allows the software test manager to detect trends and to

anticipate problems (Chen et al., 2004). The application of metrics requires a good basic understanding of statistics and mathematics.

3.7.3 Software Test Managers as Decision Makers

The decision making role of a test manager is demanding and can be very complex. Each piece of software or software system comes with its own needs and requirements. Rarely would software or a software system that needs testing have the same characteristics, thus the test manager as a decision maker faces a continually evolving decision making role. They need to be able to direct, control, administer plans and regulate the ongoing evaluation of the testing process and related artefacts.

Test managers, or their equivalent, oversee the test effort for a complete project, or for multiple projects. For each software project the test manager is faced with several, possibly competing demands on their expertise and time. The test manager is responsible for a range of activities, must possess well developed skills, has to contend with significant organisational pressures, and needs the ability to balance these demands and perform appropriately. These different demands all require the ability to make informed decisions (to various extents). The demands which involve decision making include:

- developing the test plan;
- setting the test strategy;
- arranging the required testing resources;
- managing the test team; and
- measuring the test process.

3.8 Summary

This chapter has presented the terminology within the research foundation and its boundary. This provides a means for the reader to place the thesis in context and to help understand some of the essential elements in this research. The terminology has been defined to avoid ambiguities and clarify usage in the thesis. Any concepts or issues underlying the terms have been discussed.

Software testing is a major undertaking that involves risk, and is a verification and validation activity that is directed at successful software testing. The importance of software testing is to ultimately help produce a higher quality software product. The general term “issues” was introduced as a way to understand the numerous conditions that can occur when software,

which is tested, fails requirements or is inadequate for its intended use. Software testing was given a context: its life cycle was described, and where that life cycle fits into the software development process was identified. The significance of quality to the consequences of software issues and the difficulty in defining software quality were underlined.

Decision support was defined, and the decision making and decision maker issues underlying that definition were discussed. Risk management was examined in the software testing context, with a focus on the risks involved in decision making. The chapter closed by examining software test managers from two perspectives: their needs to enable them to do their job and their role as decision makers.

- ||| -

Chapter 4. Frameworks for Decision Makers

The following diagram provides an outline and overview of the conceptual structure of this chapter.

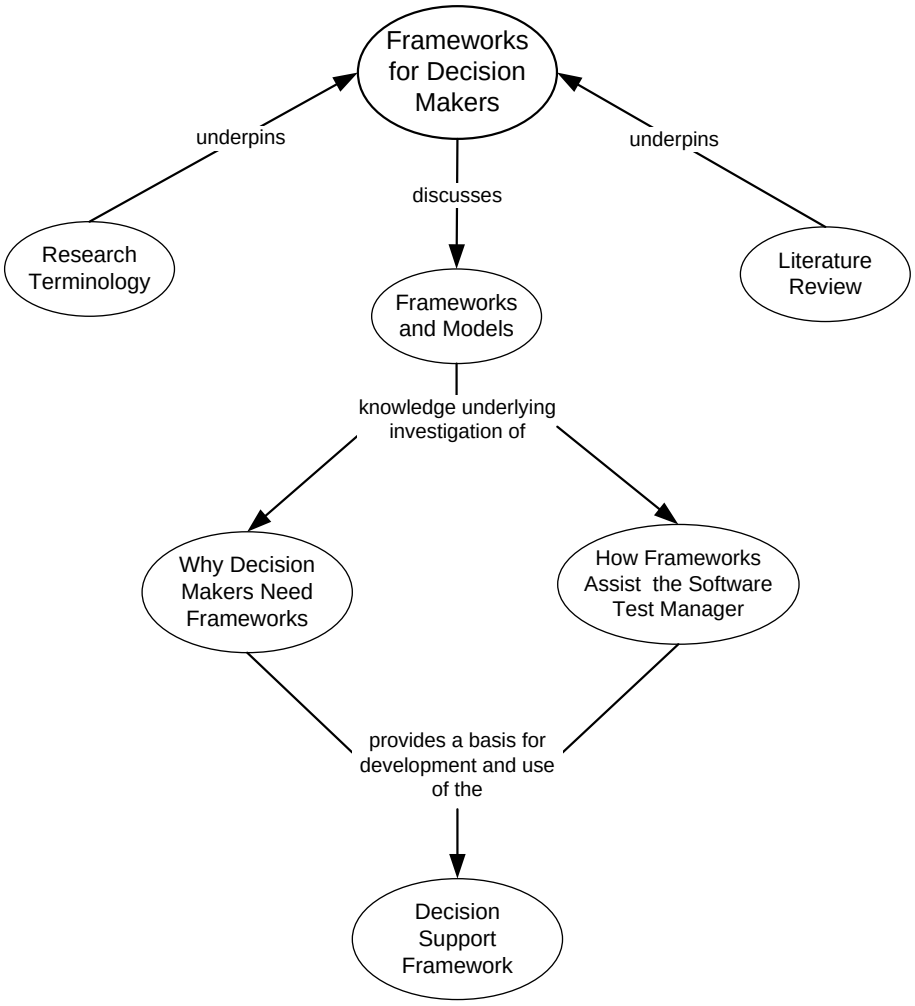


Figure 4.1: Conceptual structure of Chapter 4

4.1 Introduction

The research has shown that the development and utilisation of a decision support framework for software test managers is a useful tool. This chapter provides an understanding of what is meant by a framework. The differences between frameworks and models are explored. This is done to clarify the importance of and the need for a framework to assist and guide software test managers in their planning and risk management activities. This chapter then discusses frameworks as they relate to decision support systems in general. From the literature review (Chapter 2) it has been shown that there is inadequate research in this area, especially relating to software testing. The discussion continues and addresses two questions: 1) Do software

test managers need software testing frameworks to assist them? and 2) How can a decision support framework help a software test manager?

This chapter is important as it ties in chapter 2 (literature review) and chapter 3 (terminology), and provides a basis for chapter 5 (the decision support framework) and chapter 6 (using the decision support framework).

4.2 Decision Support Revisited

This section raises several decision support issues, important for a better understanding of the chapter. As discussed in Chapter 2, decision support, regardless of its nature, supports technological and managerial decision making by assisting an organisation's decision makers in applying their knowledge to structured, semi-structured, or unstructured issues that they need to deal with. Risk management presents a greater challenge and need in the uncertain nature of structured, semi-structured, and unstructured situations/issues/problems/ requirements/opportunities.

4.3 Frameworks and Models

This section discusses frameworks and models. A distinction between frameworks and models is discussed showing how each can be used in a decision making process. This distinction is very important in the creation and utilisation of frameworks. Decision makers need to be aware of what a model is, what a framework is, and how and where each can or should be used.

4.3.1 Frameworks

Frameworks are a useful way to deal with the complexities of a phenomenon, and work with that phenomenon.

> ***Framework****: is a set of ideas and assumptions to organise a thought process about a phenomenon, such as a type of thing, event, or situation (Alter, 2002).*

A framework can be a real or conceptual structure. A framework can be an abstract logical structure (Burns & Grove, 2009). Frameworks use a variety of terms in their structure (Alter, 2002). The structure consists of a high level set of requirements that provides for the inclusion of such concepts as: elements, components, objects, entities or factors (each author defines the terms applicable to their framework structure). This provides a guide or support to achieve an overall goal or objective ("Framework", 2008; Jentzsch, 2008).

The term concept is central to frameworks and needs to be defined.

> ***Concept****: is a perceived regularity in a phenomenon. A concept has a separate meaning, is described abstractly, and is given a label to identify it (Burns & Grove, 2009; Novak & Cañas, 2008).*

This definition is based on Novak and Cañas (2008) "A concept is a perceived regularity in events or objects, or records of events or objects, designated by a label. The label for most concepts is a word," (Novak & Cañas, 2008, p. 1), and Burns and Grove (2009) "A concept is a term that abstractly describes and names an object, a phenomenon, or an idea, thus providing it with a separate identity or meaning." (Burns & Grove, 2009, p. 126).

Frameworks do not have to be specific to a particular environment, application or business issue. The framework requirements may be abstract or concrete, are highly integrated and related within some phenomenon, which has a defined boundary. Frameworks are a mechanism for abstracting the essential qualities from the phenomenon of interest, and therefore they help users to understand and manage the complexity of the phenomenon that they deal with. The essential qualities of the phenomenon of interest are captured and illustrated in some way in the framework ("Framework", 2008; Jentzsch, 2008). Relationships have direction and strength (which may be positive or negative) (Burns & Grove, 2009), and therefore are more accurately described as directional signed relationships (DSR). Directional signed relationships are used to provide an understanding of the influences within the phenomenon, and DSR are identified between framework elements and concepts.

A well-defined framework then:

- identifies a well-defined boundary of interest;
- makes it easier to work with complex business issues and concepts;
- unites the various discrete objects, elements/components that populate it;
- makes it easier for others to understand the application by presenting the boundary, elements, and relationships of the structure;
- provides a basis for employing some type of analysis and investigation of the phenomena of interest; and

- provides a basis for using some type of what-if analysis on the influence of the elements towards the achieving of the goal.

According to Alter (Alter, 2002, p. 44):

> A framework is typically used to create a model, a useful representation of a specific situation or thing. Models mimic reality without dealing with every detail of reality. Models typically help people analyse a specific situation by combining a framework's ideas with information about the specific situation being studied.

A framework is based on *what* is to be achieved, not *how* it is to be achieved; and therefore a framework implicitly decouples the *what* from the *how* (Jentzsch, 2008). A framework can be modified – component descriptions changed, components added or deleted, and relationships adjusted. Nonetheless, at any particular point in time, a framework identifies an invariant set of components and therefore infers a discrete boundary (Jentzsch, 2008). A framework can be, for example: a set of ideas and assumptions; a defined approach; a set of rules; a set of policies; a set of questions for data collection and understanding of an issue or domain; a high level outline or set of processes to be followed to achieve some outcome; or a group of programs ("Framework", 2008; Jentzsch, 2008).

Thus, a framework forms a boundary upon which some form of analysis is performed to supply the information to achieve an objective or goal. The user can populate the framework with entities to generate a model that is directed at the specific phenomenon they have an interest in.

4.3.2 Models

In business, models are used in a variety of ways to help describe or explain some small part of the real world.

> ***Model****: is a simplified representation of a specific event, thing or situation in the real world. Models help inform the user, emphasise some features of reality, and downplay or ignore others (Alter, 2002; Turban et al., 2011).*

Generally, graphical illustrations or some type of mathematical illustration represent models. Since models are typically generic, a model can be applied to different situations. Models may be applied to different specific, concrete situations, as a way to inform the user of that

situation. This makes most models generic, highly useful and adaptive to the situation at hand. For example, the IPO (Input-Processing-Output) model can be used and applied to different situations.

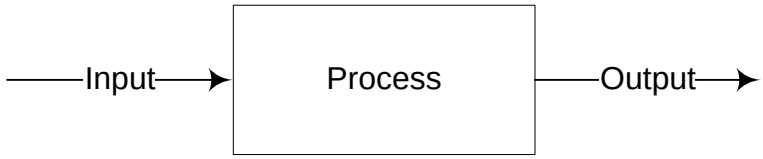


Figure 4.2: Input process output (IPO) model

The IPO model can be used in diverse situations. The IPO model can be applied to data, where data comes in, is processed, and the output is information. Or the IPO model can be applied to a legal case, where case details come in, are processed, and some decision results (output) are observed.

Models do not generate a framework, nor do they infer a framework. A model can be generated by a framework. If a model is generated by a framework it is more specific to a boundary within a phenomenon of interest (Jentzsch, 2008).

4.4 Decision Frameworks

A decision framework is a framework (Alter, 2002) that provides a way to organise a thought process about a phenomenon, and provides a guide or support for the decision making required to achieve an overall goal. A phenomenon is a particular type of thing, event or situation. Decision frameworks consist of a structured set of high level requirements, which allow objects or entities to be plugged into the framework. Decision frameworks have all the characteristics of frameworks, and are a mechanism for abstracting the essential qualities (typically represented in issues and concepts) from a phenomenon of interest, to help users understand and manage the complexity of that phenomenon. The important qualities of the phenomenon of interest are captured in the components of the decision framework and in the relationships amongst the components.

> ***Decision Framework****: a way to organise a thought process about a phenomenon to provide a guide, or support, for the decision making to achieve an overall goal or objective (Alter, 2002; Jentzsch, 2008).*

One perspective (Jentzsch, 2008) of what a decision framework does is it:

- makes it easier to work with complex issues and concepts;
- unites the various discrete objects or entities chosen to populate it, and when populated, builds upon the populated framework to extend it and create something that is more useful to the user;
- forces implementation so that consistency and more flexible environments can be promoted;
- makes it easier for others to understand the environment to which the framework is being applied to, or directed to;
- makes it easier for the next user to have a beginning to analyse a phenomenon – because the framework identifies a set of requirements that simplify the phenomenon; and
- provides a mechanism for that user to investigate, utilise, and work with phenomena.

4.5 Decision Support Frameworks

Decision support frameworks are both decision frameworks and frameworks. They come with all the ideas and advantages of decision frameworks (see Section 4.4) and frameworks (see Section 4.3.1).

> ***Decision Support Framework****: is a guide whereby the user can take what they know about the type of phenomenon, and the environment in which the decision support framework will be used, and enhance one or more relevant aspects of decision making information to achieve an overall goal (Alter, 2002; Jentzsch, 2008).*

A decision support framework emphasises the provision of support to decision makers, in terms of increasing the effectiveness, or clarification, of some part of their decision making effort. This support involves identifying the concepts (and their details) to be considered, showing how the concepts are related within the decision support framework space, and providing a way to analyse and interpret that space. Analytical methods accompany the framework, and these methods allow users of the decision support framework to analyse and interpret the impacts of decision making alternatives, and select appropriate decision making options to achieve an overall goal.

4.6 Why Decision Makers Need Frameworks

Decision makers use thought processes to make decisions and solve problems. It is important to realise that everyone has and uses some type of decision making process, framework, or referent (something referred to, such as a person, object or event), even if it is unconscious, habitual, or unexamined (Blanchard & Peale, 1988).

Business decisions require facts, options, outcomes and consequences to be carefully considered. The necessary courses of action are neither obvious, nor are the consequences of their implementation always clear. "It is easy to charge ahead without thinking and then rationalize your behaviour after the fact. But the fact of the matter is: There Is No Right Way To Do A Wrong Thing" (Blanchard & Peale, 1988, pp. 18-19).

Software test managers are decision makers. They are responsible for making decisions that are business related. Software test managers are presented with structured, semi-structured, or even unstructured situations, issues, problems, opportunities and requirements where they may or may not have supporting documentation, upon which they have to create business artefacts.

In software testing the test manager has to create such documents as software test plans, business cases, test procedures, test environment requirements, and reporting. They have to evaluate different alternatives and provide decisions to management. These decisions are business decisions that affect the organisation's budget, resources, and the delivery of software to customers and end-users. The more information software test managers have the better they can evaluate alternatives and provide better information to management.

Each new software and software upgrade comes with different requirements, many of which may not be well structured. This diversity means that software testing needs to consider the structured, semi-structured, and unstructured situations and issues that are part of each new and upgraded software application. These issues create an uncertain decision making environment for the software test manager, which makes decision making difficult. Thus, software test managers need decision support frameworks to assist them in their decision making process. Decision makers need to use explicit decision support frameworks to provide them with a better basis for test planning and risk management decisions that they have to make to achieve successful software testing. A software testing decision support

framework provides a platform to assist the test manager. These decision support frameworks provide information, from the various perspectives that software test managers require, enabling them to structure test planning and apply analysis techniques to evaluate decision making alternatives.

4.7 How Frameworks Assist the Software Test Manager

The software test manager operates in a complex world that places many requirements on them. Pre-test planning and test planning are extremely important activities. The software test manager must explore all options and deliver well researched test plans to management and stakeholders. Anything that can assist them in this endeavour would be welcomed by the software test manager.

The demands confronting the software test manager are diverse, and can be in contention with each other. These demands usually consist of a collection of activities (typically a function), and frequently require the software test manager to evaluate decision alternatives. Chapter 3, Section 3.7.3 identified some demands of software test managers that involve decision making, but did not provide details. The current section provides a better understanding of these demands by exploring some of the issues encompassed in the demands.

Test managers, or their equivalent, oversee the test effort for a complete project, or for multiple projects. In consultation with project managers and development managers, the test manager develops the test plan, and sets the strategy, the policies and processes, schedules, priorities and goals for testing (Ammann & Offutt, 2008; Patton, 2006). The software test manager has to arrange the required resources – such as people, equipment, and space – for the software project or projects being tested. The software test manager is responsible for managing one or more testers (Ammann & Offutt, 2008). The work of a tester is monitored and facilitated by the software test manager. A software tester participates in a range of possible test activities; for example, they can design test inputs, produce test or business cases, run test scripts, analyse test results, and report test results to developers and management (Ammann & Offutt, 2008). Reporting is often done using bug reporting and evaluation of the application.

An important activity of the software test manager is the process and production of the test plan. To develop an effective test plan, the test manager must devote considerable time to test

planning, and will need to investigate possible applicable options for their test planning activities. Test planning occurs before the test plan document is completed and delivered to management.

The ability to measure a test process is an essential skill for a test manager. Test process measurement is useful for designing and evaluating a cost-effective test strategy (Chen et al., 2004). Feedback from the analysis of test measurements helps management make better informed decisions about such software issues such as product release and quality improvement (Tian, 2005). The success of a software organisation depends on the ability to produce dependable and secure products, and make predictions and commitments about those products (Chen et al., 2004). The test manager is under significant pressure to ensure that information they communicate to management, about software to be released, is factual and does not compromise the integrity of the organisation and its products. Test process measurement will promote the factual nature of such communication, allow the test manager to detect trends and to anticipate problems, and help ensure that business objectives are achieved (Chen et al., 2004).

Decision support frameworks can help the test manager investigate an issue in the following general ways – decision support frameworks (Jentzsch, 2008):

- give the software test manager a ready grasp on the important characteristics of the issue to be considered, in the testing of the particular software;
- offer a holistic and coherent perspective of issues and alternatives;
- allows the software test manager to examine, reduce, or avoid unclear, inaccurate or unfocused thoughts about the testing issues – such thoughts may occur from exploring the complexities of a testing issue without knowledge of the essential qualities of that issue;
- provide a mechanism to evaluate information, and permit an informed decision about the issue; and
- allow the test manager to identify the objects or entities to plug into the framework, making the use of the framework in support of the decision making role more advantageous.

4.8 Summary

Software test managers have a lot to do before the actual testing of software begins. Pre-plans, test planning, and organisation of often scarce resources means that they have little time to review testing alternatives and deal with testing uncertainties. A software testing decision support framework provides a basis for helping the software test manager reduce the overall burden in test planning and risk management assessment, based on a sound platform for their decision making. A decision support framework for the software test manager will help support better informed decisions, and equip them with improved decision support information.

Frameworks, decision frameworks, and decision support frameworks have been discussed as they are useful tools to assist users in working with, dealing with, and developing an understanding of structured, semi-structured and unstructured situations. These frameworks provide a guide for decision support in areas where planning and risk management are needed. The concept of decision support frameworks is seen as a flexible mechanism to assist decision makers to manage and understand the complexities of a phenomenon of interest. Decision support frameworks provide a guide for decision makers to populate the framework, based on their knowledge, and use the framework as a guide and support to achieve their overall goal.

- ||| -

Chapter 5. Decision Support Framework for Software Test Managers

The following diagram provides an outline and overview of the conceptual structure of this chapter.

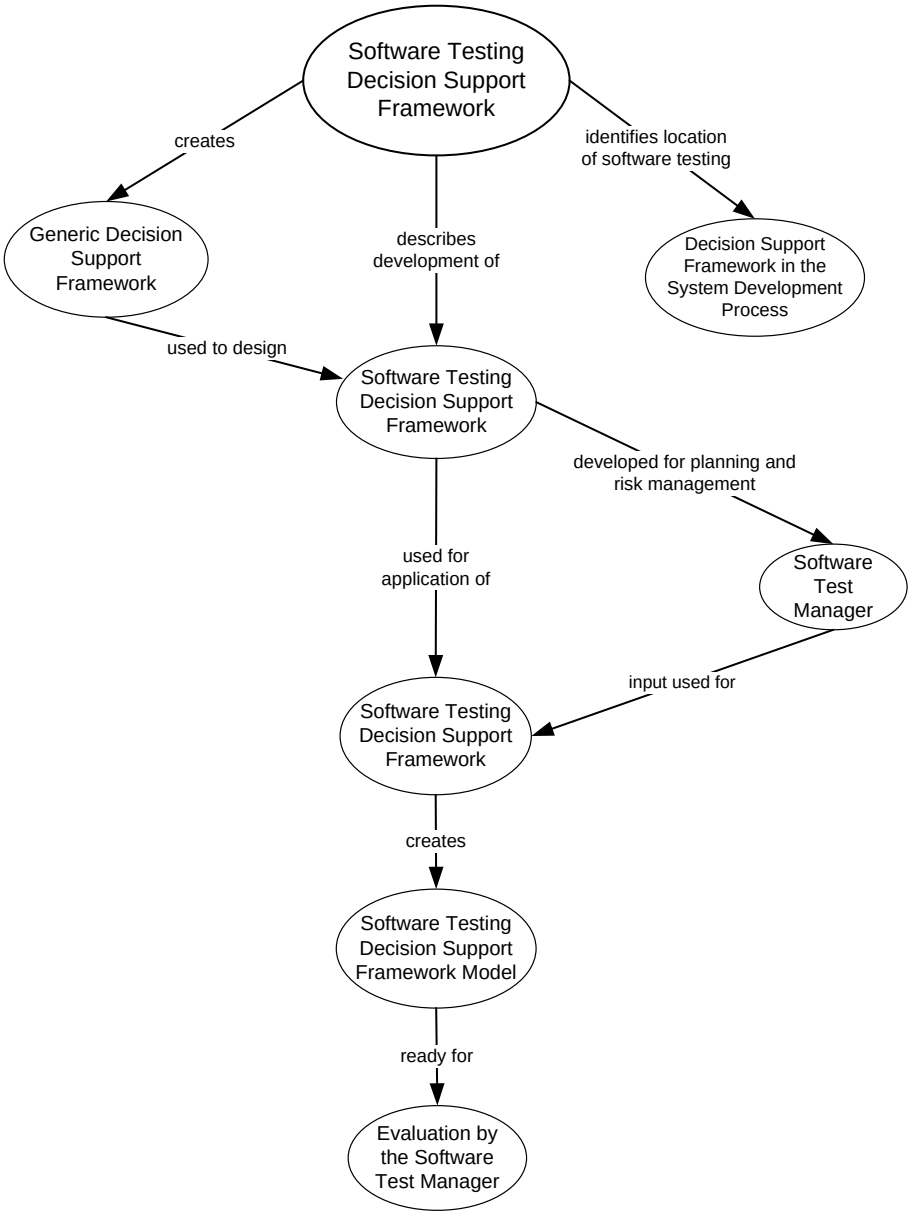


Figure 5.1: Conceptual structure of Chapter 5

5.1 Introduction

This chapter presents the design and development of a generic decision support framework as an underlying basis for the development of the software testing decision support framework (DSF) for software test managers. Next, the development of the software testing decision support framework is discussed. The details of all components of the software testing DSF

are described at length. The relationship of the software testing decision support framework to the system development life cycle is outlined. This provides the user with an overall high level perspective of the relationship of the software testing decision support framework to the application development process and software testing life cycle.

5.2 Design of a Generic Decision Support Framework

The generic decision support framework design is based on the generic structure shown in Figure 5.2. This generic structure was created based on information found in literature, information sourced from industry, and informal discussions with a software test manager.

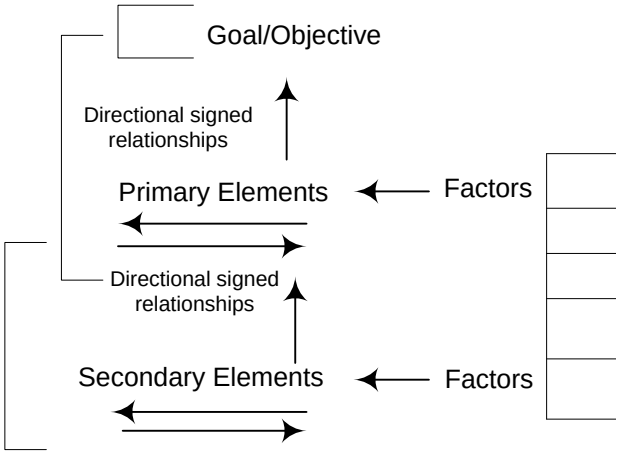


Figure 5.2: Generic decision support structure

This generic structure includes:

- a single goal/objective;
- elements that influence the success of that goal;
- directional signed relationships; and
- factors that define details about elements.

5.2.1 Elements

A decision support framework is broken down into elements. The top most element is the goal or an objective, with elements that influence that goal indicated by directional signed relationship arcs.

> ***Goal****: there is only one goal in a decision support framework. The purpose of the framework is to support the achievement of that goal.*

Decision support frameworks could potentially have hundreds of elements but must have at least two. A decision support framework is also a framework (see Chapter 4, Section 4.3.1). A framework is a set of ideas, and shows how the ideas interrelate (Alter, 2002; Power, 2001). A set (or group, or collection) is more than one of an entity. Therefore, for something to be a decision support framework it must have more than one element (idea). Further, two or more elements (ideas) must exist before they can be shown to interrelate. Decision support framework elements should not be more than three levels deep; otherwise analysis can become exceedingly complex and require elaborate software. Frameworks with too many elements may become too complex for the intended users to understand and use.

Elements are at the primary level if they directly influence the achieving of the goal. Elements are at the secondary level if they directly influence the success of one or more primary elements. Third level elements have an influence on secondary level elements (none are shown in Figure 5.2). The influence of one element on another element is most accurately described as the influence of one element to influence the success of another. The influence is identified through the relationship between the elements, based on the assigned weighting percentage for the relationship. The goal has a direct dependency on primary elements, and a transitive or intermediate dependency on second level and third level elements. Thus, the second and third level elements influence the goal via the level directly above them, as well as laterally through other elements at the same level.

The goal is the top most element. The goal element is always numbered with a letter followed by a zero (i.e. C0). Primary elements begin with the letter of the goal and are numbered with whole numbers from left to right consecutively (i.e. C1, C2, C3, … Cn). Secondary elements begin with the letter of the goal and are numbered left to right within their association with a primary element. The numbering for secondary elements is a decimal to the primary element number (i.e. C1.1, C1.2, C1.3, … Cn.m). Third level elements are numbered based on the secondary element number that they relate to (i.e. C1.1.1, C1.1.2, C1.1.3, … Cn.m.p).

> ***Element:*** *an element represents a specific concept that has a relationship to the achievement of a defined goal.*

Each element is represented by a named oval. Within the oval is a meaningful high level element name that reflects the use and purpose of the element in the decision support framework.

5.2.2 Relationships

Relationships declare that an association exists between elements and the goal, or between elements. Relationships are indicated by connecting arcs, and possess several characteristics (Burns & Grove, 2009). Relationships in the decision support framework have type, direction, symmetry and strength. The relationship type is causal – cause and effect. That is, an effect in one element results in, or influences an effect in one or more other elements. To put another way, the affected element is a consequence of the causal element. The direction of a relationship can be positive or negative. A positive relationship implies that as one element changes (positive or negative change), the related element changes in the *same* direction. A negative relationship implies that as one element changes (positive or negative change), the related element changes in the *opposite* direction. Relationships are asymmetrical or one way – if element A is affected, element B is affected, but that does not indicate or infer that if element B is affected (changes), element A will be affected (changes). Asymmetry of relationships is indicated by an arrow head at one end of the connecting arc. From each element there will be at least one relationship to another element.

> ***Relationships:*** *exist between elements and the goal, or with other elements, and are indicated by a connecting one way arc between elements.*

5.2.3 Influence Weightings

The strength or influence weighting of a relationship is the amount of influence explained by that relationship. Influence is the varying degree to which an element potentially affects other elements, or the goal. Therefore, the influence weighting of a relationship explains those portions of two elements that are associated (Burns & Grove, 2009). Some of the influence of an element, but not all, may be associated with the influence of another element (Burns & Grove, 2009). The influence of an element, not associated with another element, stems from non-manageable and often unknown external influences.

Each directional signed relationship has one and only one influence weighting. Relationships are described as "directional signed" because they have direction, and are either positive or negative. Positive relationships are indicated by the absence of the positive sign (+), and

negative relationships are indicated by the negative sign (-). Influence weightings are stated as a percentage and assigned to each directional signed relationship. Once the influence weighting has been assigned, it is not expected to vary from its initial state as that state has been determined by the user's knowledge and experience. Once the influence weightings are assigned to relationships, anything that affects one or more influence weightings indicates risk in achieving the goal, or the success of an element, and can be used for risk management assessment.

Each element relates to the goal, or another element, and is independently assessed to determine the influence weighting. The influence weighting attributed to each relationship will be ≥ 10% and ≤ 90%. This range has been identified through simulation, and reflects the range of influence weightings that would be meaningful in the real work practice for this type of decision support framework. No influence weighting will be 100% as that would indicate the goal or element is totally dependent on that particular element and all other elements would be considered secondary in their consequence to the decision. A 0% influence weighting would indicate there is no relationship between elements. A relationship could have an adverse effect on the success of an element, or achieving the goal. Such relationships are negative, and signified by a negative (-) sign. Nevertheless, negative relationships have the same range as positive relationships: the range is ≥ -10% and ≤ -90%. The range justification is the same but the reverse of the range justification used for the positive influence weighing. Each element will have at least one relationship, with the maximum number of relationships being determined by the number of framework elements in the framework and their associations.

> ***Influence weighting:*** *indicates the strength of an element's influence on another element or achieving the goal, determined by user's knowledge and experience, with the values of influence weightings expressed as percentages.*

5.2.4 Factors

Factors are a unique set of items that relate to a particular element. Factors provide detailed information about their related element. Each factor contributes to the element it is associated with, based on a contribution percentage. Factor contribution percentages are in the range of 0% to 100%. An element may have one or more factors. One or more of the factor contributions for an element could be 0%, depending on the particular circumstance. If a

factor contribution is 0%, then the factor may still be shown in the initial framework for evaluation purposes. However, because that factor provides no contribution toward the elements achieving the goal, or the success of another element, it should be removed in the completed framework. A factor could be at 100%, thus being the only factor that contributes to the element.

The sum of factors for each element must equal 100%. If the factor analysis is not equal to 100%, then additional or fewer factors for that element must be identified, evaluated, and assigned a contribution percentage such that the total equals 100%.

> ***Factors:*** *describe the details of an element's characteristics and supporting things that are needed to ensure that an element can meet its purpose.*

5.2.5 Factor Contributions

Each factor, for an element, contributes to the element being able to influence the goal or other elements. The percentage indicates to what degree the factor contributes to the element's ability to achieve its intended purpose. The sum of factors for each element will be equal to 100%. Once the factor contribution percentages have been assigned, any change will need to be assessed. If the sum of an element's factor contributions is less than 100%, then there will be an overall effect on the element and its ability to assist in achieving the goal. Thus, if the influence weighting percentage(s) associated with that element are less than their initial percentage(s) they will consequently, directly or transitively, influence the risk of not achieving the goal.

> ***Factor contribution:*** *the percentage for a factor that indicates how much the factor contributes to the element's ability to meet its influence weighting percentage(s).*

5.2.6 Illustration of the Generic Decision Support Framework

Figure 5.3 shows a generic decision support framework. The labels for the elements, noted by Cn.m.p, indicate that there are a number of possible levels and elements at each level. These element labels, n, m, and p are each an integer greater than or equal to 1. Such that C2.2.3 would be at the third level, with a relationship to the element at the second level and related to the second element at the first level. Factors, noted by Fi, indicate that an element, at any level, could potentially have many factors. The label "W_{CnCm}" is for the framework influence

weights of relationships between each respective pair of elements. The domain expert assigns the influence weights and factor contributions to the framework requirements, based on their experience. The directional signed relationships shown in Figure 5.3 are for illustration purposes only and will vary depending on the goal/objective of the framework.

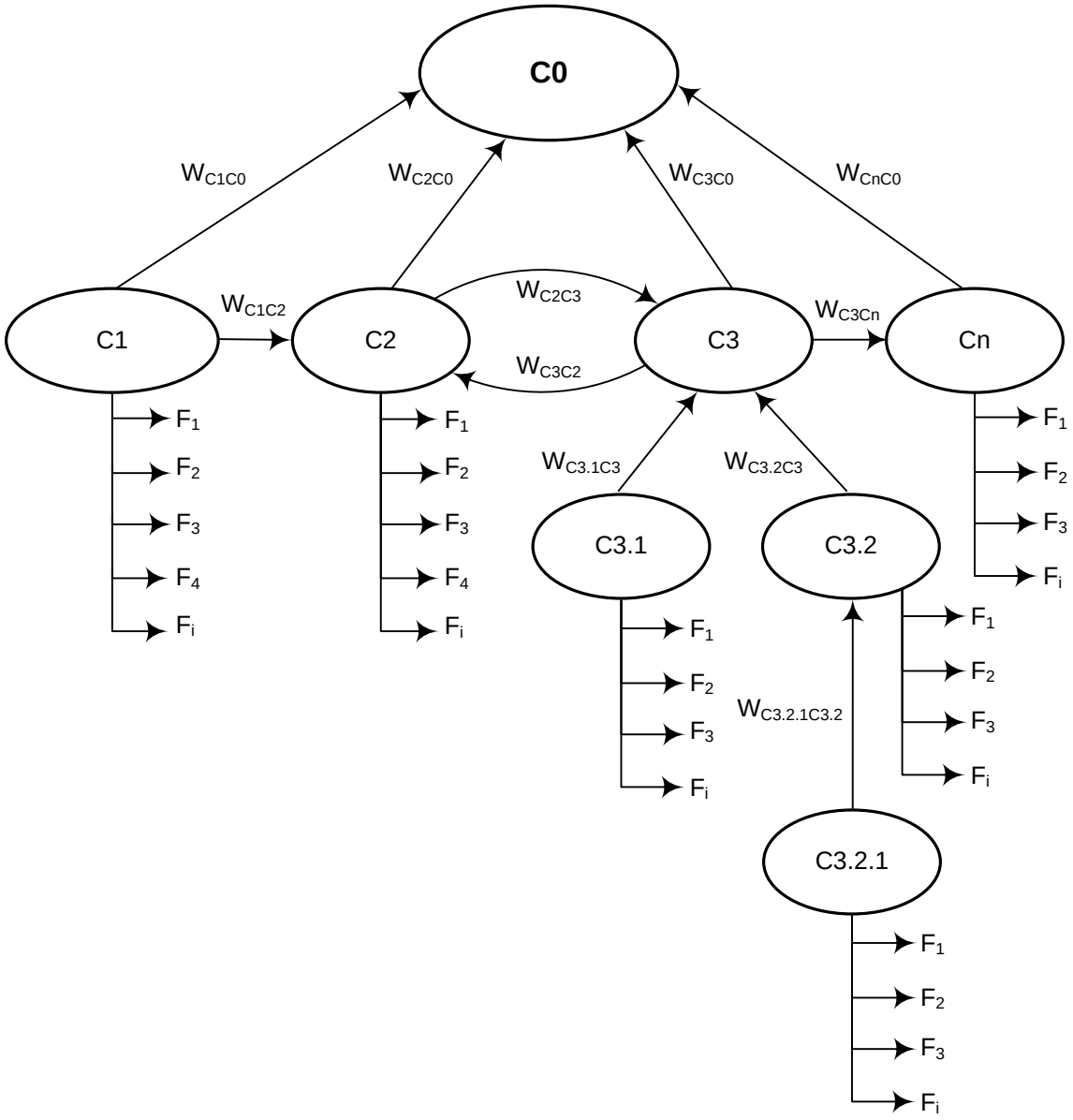


Figure 5.3: Generic decision support framework

C0 is the goal of the decision support framework. To keep Figure 5.3 simple, the generic nature of relationships in the decision support framework is only shown for primary elements. C1, C2, C3 and Cn are primary elements that have a direct relationship of influencing the goal. C3.1 and C3.2 are secondary elements with C3.2.1 a third level element. W_{C1C0}, W_{C2C0}, W_{C3C0} to W_{CnC0} are the influence weightings for the directional signed relationships from the

primary elements to the goal. W_{C1C2}, $W_{C2C3,}$ W_{C3C2} to W_{CnCm} are the influence weightings for the directional signed relationships between the primary elements.

Each element can potentially have one or more unique factors (by definition) that contribute to their element's ability to influence the goal or other elements. Figure 5.3 indicates that the elements C1 to Cn.m.p consist of factors F_1 to F_i. In mathematical form, $\sum_{i=1}^{n} F_i = 100\%$ for each element.

5.3 Development of the Software Testing Decision Support Framework

The software testing decision support framework has been developed to assist and support the software test manager in test planning and risk management in achieving the goal of successful software testing. In this section a description of the development of the software testing decision support framework and its resulting model is presented.

Until now, in this and previous chapters, the term decision support framework has been used in a generic context. However, when the term decision support framework or the acronym DSF is used in the remaining chapters of the thesis (unless explicitly specified otherwise), it refers to the "software testing decision support framework" designed and developed through this research.

5.3.1 Development Procedure

The development of the decision support framework, introduced in this research, is based on the generic decision support framework (described in Section 5.2). The developed software testing decision support framework (DSF) builds on this generic structure, and has been designed to be used with various types of software that are to be tested. The development of the decision support framework involved the following steps:

1. identifying elements relating to software testing;
2. developing definitions for each element;
3. determining relationships at all levels amongst those elements;
4. developing the requirements for influence weighting percentages for each relationship amongst the elements at each level;

5. reviewing the influence weighting associated between two, and possibly more, elements, which reflects an understanding of the relationships between causal elements and affected elements;
6. determining the factors that contribute to each element;
7. developing descriptions for each element's factors;
8. determining the requirements for the factor contribution of each factor, for its related element; and
9. identifying responsibilities and roles in completing the software testing decision support framework for the type of software to be tested.

Several iterations of the steps in the framework development procedure were required before the final software testing decision support framework was reached.

5.3.2 Element Identification

There are three primary elements and one secondary element for the software testing decision support framework. The primary elements include:

- Test management;
- Test information; and
- Test environment.

The secondary element includes:

- Technical support.

The statement and details above raise two questions. First: What do the elements of the framework represent? Second: What is the basis for the choice of the elements? These questions are now answered in turn. The elements are perceived, high level regularities drawn from the dynamic software testing space. These perceived regularities are based on previous literature and informal discussions. A major source for identifying the primary elements, which directly influence the goal of the framework, is the IEEE standard for software and system test documentation (IEEE, 2008). The mix of high level, and lower level descriptions established by this IEEE standard was the source for the Test management element, and indirectly, the Test information element. These two elements, in one form or another, figure explicitly or implicitly in other sources (Black, 2002; Dustin et al., 2009; Kaner et al., 2002; Tian, 2005). The IEEE standard was the driver for the Test environment element. The importance of the test environment is a common theme for several authors

(Borysowich, 2007; Editorial, n.d.; Farrell-Vinay, 2008; Jaideep, 2008c). The test environment needs to be setup, maintained and monitored. This straightforward observation was the genesis and basis for the secondary element, Technical support. Technical support, in the context of software testing, has received some attention in the literature, from various perspectives. These perspectives include the role of technical support in: user documentation (IEEE, 2008); sharing the test plan (Kaner et al., 2002); specific testing techniques (Kaner et al., 2002); collaboration that flows from the test plan (Kaner & Bach, 2001; Kaner et al., 2002); and maintenance contracts with outside agencies (NIST, 2002). But apparently the attention on technical support has not been from the viewpoint of the test environment. Technical support was seen as a secondary element in the framework because its major focus is support for the test environment, and therefore its influence on the goal is indirect.

Elements influence the outcome of achieving the goal of successful software testing. The software test manager has been identified to assume the decision making role of filling in the information needed in the software testing decision support framework. They are expected to have an understanding of the type of software that is to be tested, and they are expected to do element evaluation in accordance with the organisational context and the type of software to be tested. The software test manager needs to apply that information to the primary and secondary elements in the framework in arriving at the assigning of influence weights. Each element for the software testing decision support framework in placed into an oval with the appropriate text. The following initial structure shows that the C0 oval for the software testing decision support framework has been labelled with the goal.

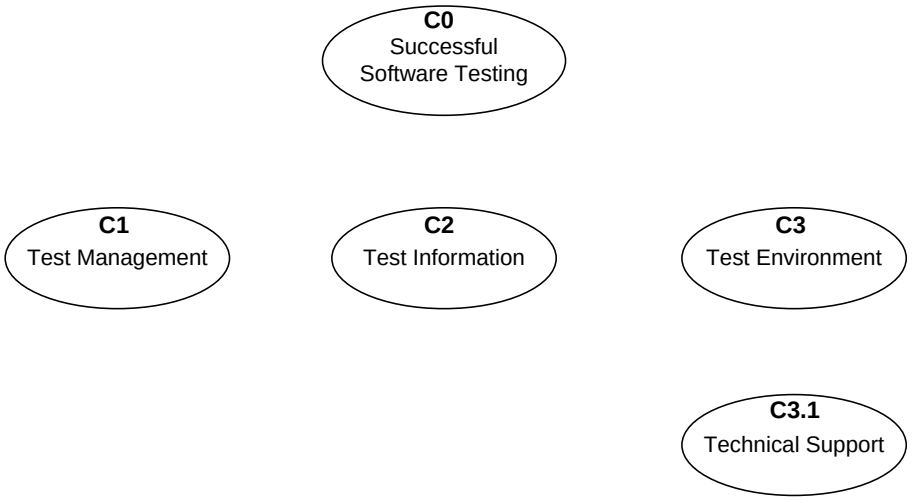


Figure 5.4: Framework elements

The primary elements are in ovals numbered C1, C2, and C3 and labelled. The secondary element oval is numbered C3.1: its numbering shows that it is a secondary element and that it has a relationship to the primary element in oval C3.

5.3.3 Elements Defined

Once each element was identified, a more complete definition was completed as described below.

5.3.3.1 Test Management

A critical issue is which test approach, method, test tool or technique does the software test manager have experience in using, and can be used to support the type of software to be tested. Use of any test approach, method, test tool or technique requires the support of appropriate test resources. The software test manager arranges and checks that these resources are available and can be used in testing the software. The software test manager is responsible for organising testing staff and ensuring that they are able to effectively use the test tools and techniques required in the testing process.

5.3.3.2 Test Information

Test requirements, essential in manual testing, are equally important for the effectiveness of any tool or technique that comes with the capability to automate tests. The important details of issues to be considered for testing are the same whether the testing is manual or automated. Test automation has to accommodate the various needs of the software under test before it can be successfully implemented. The test strategy supports the test requirements and is an integral component in the test process. An understanding of the basic elements of test design underpins the design of test cases; irrespective of the way those test cases are generated (manually or automatically).

5.3.3.3 Test Environment

The test environment is the physical and logical set up configured to support the full functionality of the software under test, and to support the tests to be executed on that software. This support includes any element required to ensure that the testing of the software provides accurate results, based on the software operating within a specific customer environment. The test environment must attempt to replicate the customer environment where the software will be used. Thus, the test environment has to be separated from the development environment, which will ensure that there is no contention for system resources, less likelihood of data corruption and that test results will be real, clear and unambiguous

(Borysowich, 2007; Jaideep, 2008d). To establish a test environment, different aspects of the test data, some test environment issues and several other environment considerations have to be addressed. It is crucial that the test environment gets adequate, continuing technical support.

5.3.3.4 Technical Support

Technical support is the underlying foundation necessary to develop and maintain the test environment. The efforts of technical support have a central focus; to minimise unforeseen side effects and disruptions to the testing process, while aiming to improve the overall quality, stability, and fitness for purpose of the software being tested. To construct a suitable test environment a set of technical support elements is needed. These elements entail relevant technical support for the test infrastructure – the test environment, testers' workstations and any facilities dedicated to testing. Part of the role of technical support is to design and create a test environment that replicates as nearly as possible the customer environment. Therefore, technical support is based on the type of software to be tested, i.e. standalone, networked, combinations, and so forth. To develop a suitable test environment technical support has to consider the environment where the software will be installed and used by the customer. Knowledge of the components of the customer environment is augmented by the concerted efforts of technical support to obtain a good understanding of the customer software requirements. Several other support items need to be examined as the test environment is being developed.

5.3.4 Determining Relationships

Relationships indicate that an element has an influence on the goal or one or more other elements. An arc is drawn from the primary element oval to the goal oval or to other elements which that element has an influence on.

In Figure 5.5, Test information (C2) has an influence on successful software testing (C0) and also has an influence on Test environment (C3). All relationships are signed directional one way. If an element has an influence on some other element and that element in turn has a reciprocal influence, then there will be two directional relationship arcs between the elements, as shown in Figure 5.5 between C2 and C3. The reciprocal influences that exist between C2 and C3 forms what Burns and Grove noted as a symmetrical relationship (Burns & Grove, 2009).

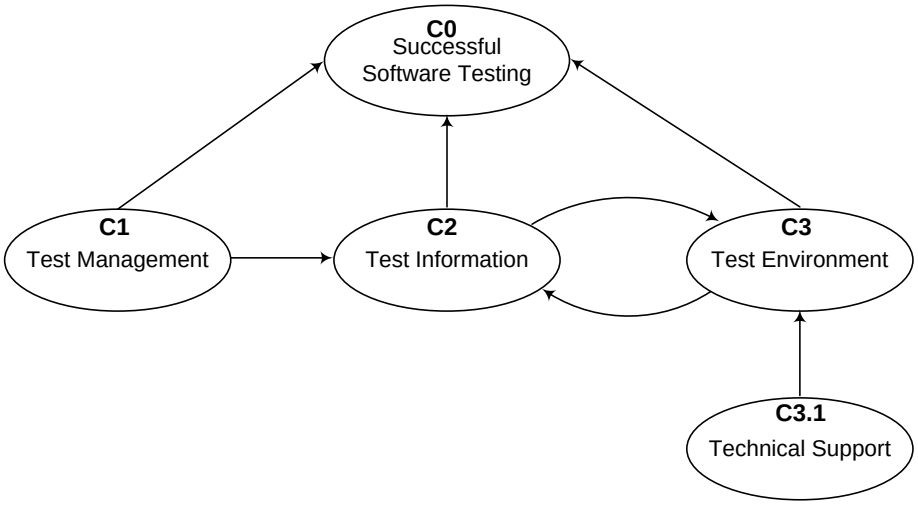


Figure 5.5: Framework relationships

The framework now shows that C1, C2, and C3 influence the outcome of the goal C0. The success of C2 is further influenced by C1 and C3. C3 is influenced by C2 and C3.1. A reciprocal influence is shown in Figure 5.5 between elements C3 and C2.

The relationships in the framework have been stated above. However, the basis for the relationships has not been explained. The basis for the relationships is twofold: general and specific. The general basis for the relationships in the framework will be dealt with first. The framework is a risk management tool. To function in this way, a prerequisite condition exists for the framework. The software test manager has to be able to explore how the interacting elements of the framework influence the goal of successful software testing. The key word here is "influence". That is, to get a handle on element influences on the goal, the software test manager needs to be able to work with framework relationships that possess the property of strength (or influence). Ultimately, this handle on the element influences, will allow the software test manager to determine the risk consequences of the changing conditions in a specific software testing situation.

The specific basis involves two perspectives: direct influences of the elements on the goal, and influences between the elements. Section 5.3.2 established some literature support for the elements of the framework. In establishing this basis, Section 5.3.2 also provided the specific basis for the direct influence of the primary elements, Test management, Test information and Test environment towards the framework goal. Section 5.3.2 further indicated, indirectly from the literature, that the secondary element, Technical support, has a direct influence towards the Test environment element. The IEEE standard on software and system test

documentation discusses numerous software testing entities, in overview, in detail, and from different structural perspectives (IEEE, 2008). This discussion suggests that Test management has a direct influence towards Test information. In addition, the discussion in the IEEE standard indicates that there would be dynamic feedback between the Test information and Test environment elements, allowing adjustment for a changing test situation. Thus, a reciprocal or symmetrical relationship is formed between these two elements.

5.3.5 Requirements for Influence Weightings

Influence weightings and directional signed relationships work together. For each relationship there is assigned one and only one influence weighting. For each element a relationship is drawn to the goal or other elements it supports and influences.

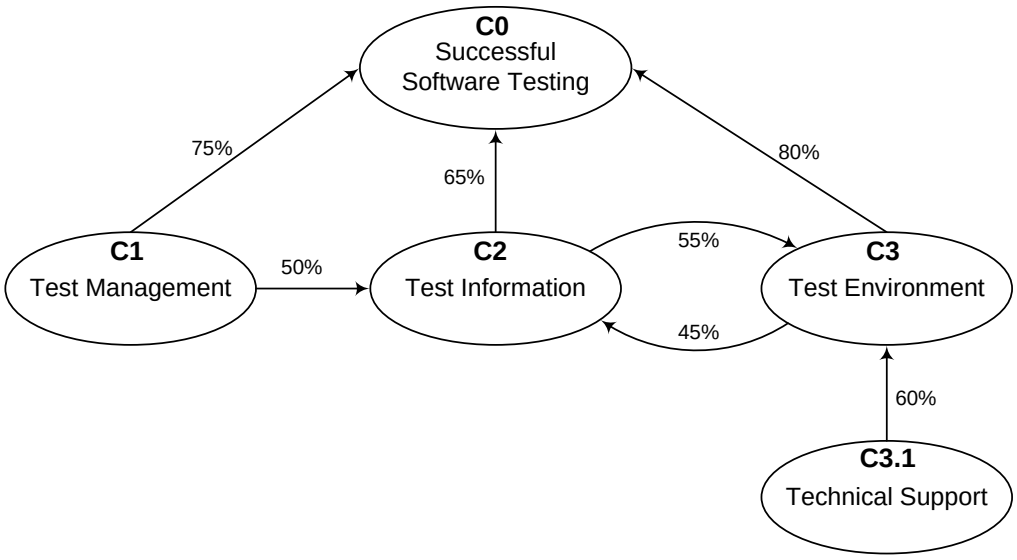


Figure 5.6: Framework relationships with influence weightings

Influence weightings are stated as percentages. Influence weights are determined by the software test manager based on their knowledge of the type of software to be tested, underpinned by their software testing experience, and within the organisational context. Figure 5.6 includes influence weightings between elements and the goal, and other elements. The influence weightings shown in Figure 5.6 are for discussion and illustration purposes only.

The influence weightings are read as follows. Test management has a 75% influence on Successful software testing (C1 $\rightarrow$ 75% $\rightarrow$ C0). Or to say it another way, the issues involved in Test management contribute 75% to Successful software testing. However 25% of the factors of Test management are not assessable to the goal, and would be considered as

factors that are not within the software testing problem domain. Thus, even if Test management's factor contributions are at 100%, and the assigned influence weighting of 75% is satisfied, there still exists a 25% risk that this element will not be able to adequately support the goal (C0).

The other influence weightings shown in Figure 5.6 include:

C1 $\rightarrow$ 50% $\rightarrow$ C2
C2 $\rightarrow$ 65% $\rightarrow$ C0
C2 $\rightarrow$ 55% $\rightarrow$ C3
C3 $\rightarrow$ 45% $\rightarrow$ C2
C3 $\rightarrow$ 80% $\rightarrow$ C0
C3.1 $\rightarrow$ 60% $\rightarrow$ C3

5.3.6 Element Analysis to Determine Factors

Each element's definition was presented earlier. Within that definition a list of factors that pertain to that element are given below.

Test Management

The factors for Test management include:

- Test manager support;
- Appropriate test resources;
- Potential for automated testing; and
- Organising test staff.

Test Information

The factors for Test information include:

- Test requirements;
- Test details;
- Software needs and test automation;
- Test strategy; and
- Test design.

Test Environment

The factors for Test environment include:

- Test environment issues;
- Test support; and
- Test data requirements.

Technical Support

The factors for Technical support include:

- Support for test infrastructure;
- End-user environment;
- Information for test environment; and
- Other support items.

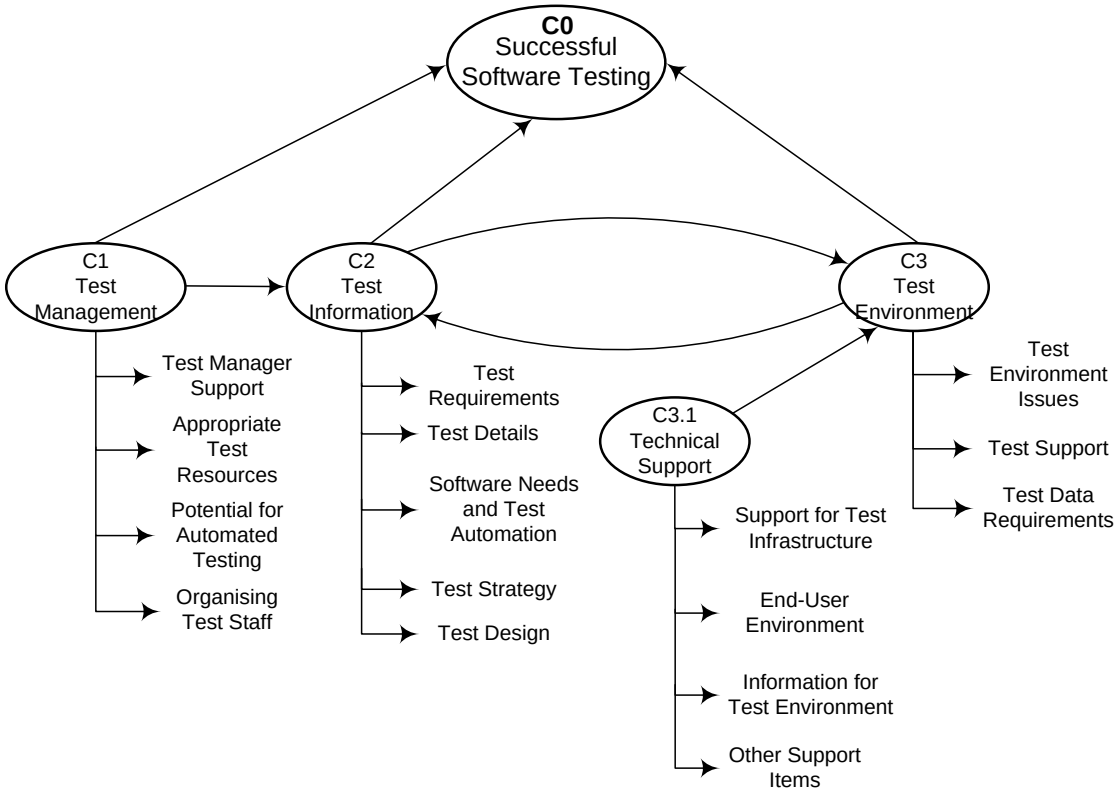


Figure 5.7: Framework elements and their factors

5.3.7 Factor Contribution Requirements

Each factor is assigned a contribution percentage that it makes to its related element. The factor contribution is assessed and assigned by the software test manager. The contribution is a percentage that defines the degree that the factor contributes to the support of its element's ability to satisfy the influence weighting percentage(s) associated with it. Factor contributions for an element can, and will vary by the type of software being tested. Figure 5.8 includes factor contributions for each factor of a particular element, for all the elements of the framework. The factor contributions shown in Figure 5.8 are for discussion and illustration purposes only.

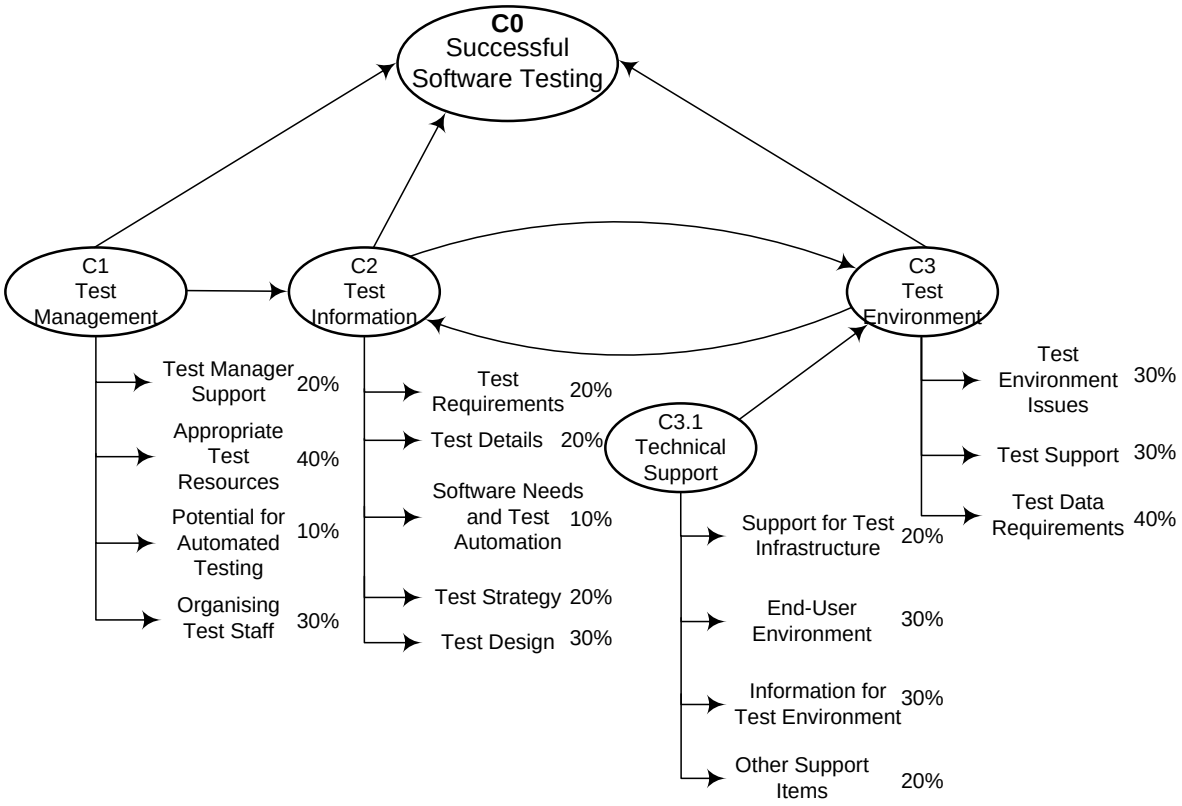


Figure 5.8: Framework element factors with factor contributions

The rules for the factor contribution percentages are as follows:

- contribution percentages are in the range of 0% to 100%;
- each element's total factor contributions percentage must be equal to 100%; and
- the percentages are the best estimate provided by the software test manager.

Although it is not envisioned that any one factor contribution for an element will be 100%, this rule is there just in case the type of software to be tested meets this situation. However, at any time for a type of software to be tested, one or more of the factor contributions for an element could be zero percent. If it is zero percent then the factor should not be shown in the software testing framework, as it says that the factor provides no contribution toward the elements achieving the goal or the success of another element.

5.3.8 Element Factors Described

This section discusses the details of the factors for each element shown in Figure 5.8.

5.3.8.1 Test Management Factors

This section discusses Test management factors.

Test Manager Support

Test manager support considers three elements. The test manager's ability to explore testing approaches or methods is one element. The ability to explore an approach or method is governed by the time and budget that is available to the test manager. The second element is how well or poorly does one particular approach or method integrate into the test environment. The final element is the possibility of implementing a testing approach or method. Implementing any new software testing approach or method will require the software development manager's approval. Generally this implementation approval has to be justified in terms of delivery dates and resource costs. Knowledge of the testing approach or method is essential, and may require training of staff, combined with good documentation to resolve any problems or difficulties with the approach or method (IEEE, 2008).

Appropriate Test Resources

Test resources include testers, test environment facilities and tools, technical support for the test environment, and special procedural requirements – e.g. security, access rights to information, and documentation control (IEEE, 2008). The preceding test resources need to be carefully balanced by the software test manager so that he/she can support any tools, testing approaches or methods selected.

Potential for Test Automation

The test manager will gather the requirements for possible test automation as part of their test planning activities. The suitability of test automation for the software is a complex issue. Two key elements that the test manager has to understand are the software requirements and the test data requirements. An understanding of the software requirements will consider such issues as the complexity and size (such as lines of code) of the software, the number of software builds, the testing phase of the software, the technology requirements of the software, the software's mission criticality and risk, and the schedule and timelines of the software (Dustin et al., 2009; Mosley & Posey, 2002). The test data requirements will include the data depth, data breadth, data scope or variation, test execution data integrity, and data conditions in the software domain (Dustin et al., 2009).

Organising Test Staff

Before assigning responsibilities for testing tasks to testers, the test manager has to identify if there are suitable and available tester staff. This includes knowing if the tester has a working knowledge and necessary skills to implement, and execute, the selected testing approach or

method. The test manager may need to arrange some type of training if the necessary skills are not available when required for the testing of the type of software (IEEE, 2008).

5.3.8.2 Test Information Factors

This section discusses Test information factors.

Test Requirements

Test requirements are important because they provide predefined expectations of the software's behaviour (via the software requirements), and because they provide a basis for test results verification: the comparison of expected behaviour with actual behaviour (Mosley & Posey, 2002). The major source of test requirements is the software requirements specification, but other sources such as software usage profiles can be used (Mosley & Posey, 2002). The results of high level project risk management can be used to prioritise the test requirements (Mosley & Posey, 2002). Each test requirement has a test objective: a statement of what a tester wants to achieve when implementing the test activity specified by a particular test requirement (Mosley & Posey, 2002).

Test Details

The essential details for the testing of software items are similar for any testing approach, manual or automatic, and include several factors. Test items can include source code, design components, specification modelling elements, or input space descriptions (Ammann & Offutt, 2008). Most software projects have different test levels. These test levels have their own resources, methods, environments and documentation (IEEE, 2008). The test level is commonly based on the software development activity (Ammann & Offutt, 2008). Common test levels include software component testing (individually, in groups, or all), integration testing, system testing and acceptance testing (IEEE, 2008). The software features to be tested and the software features not to be tested are an important consideration. Criteria to establish when a test item has passed or failed testing must also be specified (IEEE, 2008). The sufficiency of testing has to be ensured by test coverage or some other method (IEEE, 2008).

Software Needs and Test Automation

Implementation of automated testing has to adapt to the needs of the software being tested. Perspectives of software needs will differ between people, and different organisational settings will also have an effect on what software needs are viewed as important. Possible software needs are now discussed. The stage of software development when testing is first

introduced will influence the approach to automated testing. Like manual testing, early involvement of automated testing allows early detection of faults, which enables a much cheaper fix (Dustin et al., 2009). Mosley and Posey (2009) assert that if the software is subject to significant change during the development process (greater than three builds), then automated testing should be considered. The flexibility of the software's schedule and timelines will determine the time available to implement an automated testing solution. The more time available the more comprehensive the testing solution (Dustin et al., 2009). However, if the software development cycle is on a very tight delivery schedule then there will be no time to implement automated testing (Mosley & Posey, 2002). The technology requirements of the software have to be allowed for when automated testing is adopted. For example, different operating systems and different operating environments will affect how test automation is implemented (Dustin et al., 2009). The strategy used to apply automated testing depends on the complexity of the software. Frequently, the more complex the software the more complicated the automated testing effort will be (Dustin et al., 2009). The criticality of the software's business mission and the risk in the operation of the software in the user's environment will affect the automated testing approach (Dustin et al., 2009).

Test Strategy

A well thought out test strategy is crucial to a successful test process. The future directions for the testing process are established by the test strategy. The test strategy is the set of ideas that guides the test design, describes what will be tested and how, and is an important element of the test plan. The test strategy is a collection of choices that summarises the motivations for the tests, and explains and justifies the testing to be done (Kaner et al., 2002). The test process is connected to the software project mission by the test strategy, and the test strategy describes the generic methods, broad techniques and tools used to create the test case values (IEEE, 2008; Kaner et al., 2002). The test strategy is risk focused – business, planning, product, project, user and environment risks are all potentially assessed (Farrell-Vinay, 2008; Kaner et al., 2002), and the test strategy attempts to mitigate the identified risks by guiding the effective design of test cases that will find bugs. Generally, different test strategies are implemented over several test levels for software projects.

Test Design

The automation capacity of a tool or technique must be integrated with the test design if that automation capacity is to be implemented successfully. Sources for test design that should be considered are the requirements and functional specifications, source code, special properties

of the input and output domains, operational (usage) profiles, and previously stored faults (Naik & Tripathy, 2008). The test design identifies the features and combinations of features to be tested for the test item (Farrell-Vinay, 2008; IEEE, 2008). Any refinements of the high level test strategy for the project are specified by the test design. The test design identifies the specific test techniques and tools to be used, and the method of analysing the test results (Farrell-Vinay, 2008; IEEE, 2008). Criteria to establish when a test feature, or combination of test features, has passed or failed testing must also be identified (IEEE, 2008).

5.4.8.3 Test Environment Factors

This section discusses Test environment factors.

Test Environment Issues

Multiple test environments are usually needed to support different aspects and different test levels of software testing (IEEE, 2008). The test level issue was raised from the perspective of the Test information factor, Test details, which discussed the requirements for any testing approach adopted to test software (see Section 5.3.8.2). The current factor puts the test level issue in a more specific context, the test environment. The discussion of the test level issue in the previous section is reinforced and augmented. Test levels require their own resources, methods, environments and documentation (IEEE, 2008). Test levels are commonly based on the software development activity (Ammann & Offutt, 2008), and examples of these test levels include software component testing (such as unit and module testing), component integration testing, system testing and acceptance testing (IEEE, 2008). Examples of other test levels for software development activities are tests of each software requirement, software qualification testing for all requirements and system qualification testing for system requirements. More example test levels include operations, installation, maintenance, regression, and non-functional levels – such as security, usability, performance, stress, and recovery (IEEE, 2008). The test environment embraces the environment for setup before testing, the environment for execution during testing, and the environment for any post testing activities – such as recording of test results (IEEE, 2008). Configuration of the test environment for the necessary combination of hardware, software and network is required. This configuration includes the test item and any other application or system software (Borysowich, 2007). Setup and maintenance activities for the test environment raise their own issues: who will perform these activities, and when. These are concerns that have to be resolved.

Test Support

Support for testing involves establishing the test environment to run the tests, and support for the workstations that the testers will use to execute the tests. Support for the test environment includes hardware, software, network components, facilities, personnel and anything else required for the tests (IEEE, 2008). The hardware comprises at least a test server, and workstations for the testers. The hardware may include different servers, such as application, file, database and web servers. The software includes any software with which the software under test interacts, or is affected by in some way. The software may include one or more operating systems, service packs, network software, test management tools, and other test support tools. Optionally, the software may include, depending on the type of software to be tested, software for one or more web servers, browser software, database management system software, 3rd party software, editing tool software, and virtual-ware tools. If the software to be tested requires a network then the configuration/topology of the network components is an issue. Various combinations of hardware, software and network components may be required (IEEE, 2008). The facilities may include a dedicated test area, which could be internal or external to the organisation. Personnel should only consider the skills needed by testers as it relates to the type of software, and the environment, which the software will be tested in. For example, virtual software may be needed in the test environment, thus testers will need an understanding of how to use it and where to use it.

Test Data Requirements

Given the software to be tested, the required and desired properties of data needed to fully test that type of software have to be determined. Dustin et al. (2009) explored test data concerns in the context of automated testing. The data concerns or requirements identified by Dustin et al. (2009) also apply to software to be tested via non automated means. These test data requirements include such things as: the data depth, data breadth, data scope, test execution data integrity and the domain conditions of the software under test. The depth or volume of the data required to support the tests is a consideration. For example, volume or performance tests are only meaningful for quantities of data that reflect the production environment (Dustin et al., 2009). Testers need to examine the test data breadth, or the variation in the data values. The data variation can include different points in the data range (such as low, mid, high, median or average), data sets that are uniquely identified (for example, different bank accounts for different people), and all the different classifications or

categories of data that exist – such as different types of bank accounts: savings, loans, business, investment, joint etc. (Dustin et al., 2009).

The scope of the test data values needs to be studied by the testers. The data scope relates to the accuracy, relevance and completeness of the data. Complete and accurate test data allows the software to be fully exercised and supports evaluation of the test results. No missing data or no inappropriate data ensures that test data is accurate, relevant and complete. Test data with these characteristics can be used in different ways; for example, to validate a specific purpose of the software (such as an overdue payment), or to exercise the various paths of the business logic captured by the software (Dustin et al., 2009).

Data integrity when tests are executed needs to be maintained by the test team. The test team has to have the ability to segregate test data, modify that selected test data and then return the test data to its initial state, regardless of the manner in which the test data is stored (Dustin et al., 2009). When several testers are performing tests simultaneously there must be a mechanism to ensure that one particular test will not adversely affect the data required for another test. There is always the potential for one tester's execution to adversely affect another tester's outcomes. Dustin et al. (2009) suggest that one way to avoid such conflicts is to assign separate testing tasks, and ask each tester to focus on a specific area of software functionality that does not overlap with the functionality chosen by other testers.

Data sets should mirror specific conditions in the domain of the software application (Dustin et al., 2009). The data patterns of these conditions can only be reached by performing one, or many operations on the data over time. For example, a year-end closeout is a common function of financial systems. Creating test data in the year-end pattern allows the test team to test the year-end closeout state of the system, without having to enter data for a whole year (Dustin et al., 2009). Thus, testing is simplified because only a single load of test data is required, rather than performing repeated system operations to transform the data to the year-end closeout state (Dustin et al., 2009).

In some cases an adequate data set may already exist. As part of the data requirements, data preparation activities need to be considered. These data preparation activities include loading tables, establishing user access, data processing, consistency checks, data analysis and removal of any proprietary or personal information from existing customer, or system data

(Borysowich, 2007; Dustin et al., 2009). Data requirements further considers test data conversion issues and test data migration issues, wherever the source of the data.

5.3.8.4 Technical Support Factors

In this section, Technical support factors are discussed.

Support for Test Infrastructure

Appropriate support for testing means that technical support has a broad ambit for the activities they must undertake. Technical support is required to sustain the entire test environment and all it comprises. Technical support is responsible for acquiring, installing and maintaining the hardware, software, network components and anything else required to support the test environment used when the software is tested. The hardware and software necessary for each tester's workstation is the responsibility of technical support, who acquire, install and maintain that software. Technical support provide advice about the feasibility of any potential testing facility to the test manager, or whoever is responsible for managing the test facility, commonly a test laboratory. This advice considers such things as the suitability of the space to accommodate the configuration of the hardware and associated furnishings to be installed, and to fit the required cabling (Black, 2002).

End-User Environment

In some cases, technical support will not know or be allowed to know much about the end-user/customer environment. This could be due to security classification, environment issues, or that the user base is so large and unpredictable that the real users are unknown.

The test environment must reproduce (as closely as possible) the characteristics of the environment(s) in which the software will be installed and subsequently used. This approach will be more likely to yield meaningful results when software is tested, because issues contributed by the test environment will be minimised. The configuration of the customer environment can be complex, and technical support needs a clear understanding of the components (such as hardware, software and network) of the environment that will be used to run the customer's software. The hardware includes the servers – such as application, database, file and web servers, and the customers workstations – which comprise such aspects as the processor speed, size and speed of hard disk and RAM, and bus speed (Jaideep, 2008a, 2008d). The software includes the operating systems of the servers and customers workstations, browser and database software, and any other software required for the servers

or the customers workstations (Jaideep, 2008d). The network includes the type of hardware/software technology used to connect devices in the network (such as Ethernet or wireless), the communication protocol (for example, HTTP or IIOP), and the topology (such as type of physical design and type of data transmission). Technical support must determine which versions or releases of hardware and software will be used by the customer, including the patches that will be applied to the software (Jaideep, 2008d). To adequately simulate the customer environment, the test environment has to accommodate the various requirements of the different versions/releases and patches. If technical support's understanding of these issues is inadequate then the software cost can significantly increase, and there can be major delays to the project's deadlines (Jaideep, 2008c).

Information for Test Environment

Technical support has to develop an understanding of the global (user/customer) environment and identify suitable test facilities to replicate that environment as best as possible. This information is used to develop a suitable test environment, and can be gleaned from user/customer requirements for the software, as well as details about the environment where the software will potentially be used. Therefore, the requirements must be well understood by technical support. Usually the customer requirements are translated to a software requirements specification to assist developers of the software. Hopefully the software requirements specification is available; irrespective of its form, level, source, or how it evolves. A good grasp of the business issue, or business problem, faced by the customer may lead technical support to a better understanding of the customer requirements. If the customer and customer environment is known, the requirements come from the customer. If the customer's environment is unknown (such as when the software product is designed for mass distribution, or when the customer's environment has a restricted security classification) then the requirements become more difficult to determine. In this case, a beginning set of high level requirements is developed based on the experience of technical support with the requirements for other software, and the details of these high level requirements are refined over time. Until installed, no one knows for sure that the software will work as intended, and meet the customer's needs and expectations.

Other Support Items

Technical support needs to address such issues as security, environment control (temperature, humidity, dust, power failures, power surges etc.), and on-going technical support.

Security involves access to the test environment, portability of hardware (such as laptops), and 3rd party non-test related software (software not used in the testing of the software application, but used in some support role, such as word processing and spreadsheets (data manipulation, creating graphs illustrating progress, fail rate, or other management information). This software might include software for bug reporting and tracking, software to support record management, and software, such as virtual-ware, which needs to work with the testing. Hardware may also be included, in that some of it may relate to special security items.

Technical support has a pivotal role in supporting administration procedures. Administration procedures are important to and underpin any test process. Technical support can include the test procedures that are kept in a separate system from the software to be tested. Administration may also include advice on the creation of documentation in accordance with the government, customer, or corporation requirements. Administration addresses reporting mechanisms, any record management, the support for documenting tests, recording tests, and determining how test results will be configured, protected from unauthorised access, and stored (IEEE, 2008; Jaideep, 2008a).

5.4 Decision Support Framework and the System Development Process

Chapter 4 provided a basis on how frameworks that are used by the software test manager, can be valuable tools for helping in their decision making role, in particular, to support test planning and risk management assessment. Chapter 4 further discussed the value of frameworks for decision makers.

The system (or software) development process includes a life cycle. The system (or software) development life cycle has been looked at and presented from various perspectives, for many years (Hoffer et al., 2011; Kay, 2002; Kendall & Kendall, 1998; Wasson, 2006). Though the phases or steps may be labelled differently, all system development life cycles have a phase that deals with testing. From the system development testing phase there is a related software testing life cycle of one type or another (Kaner, Falk & Nguyen, 1999). There are a wide variety of software testing life cycles that are examined from different perspectives, and broken down by different phases (Fan, 2010; Kaner et al., 1999; Patton, 2006; Silva, 2008; Tamres & Mills, 2002). Given the software development life cycle and software testing life

cycle, this raises the question: Where does the software testing decision support framework fit into these life cycles?

The following diagram illustrates where the DSF fits into the overall picture of the system development and software testing life cycles.

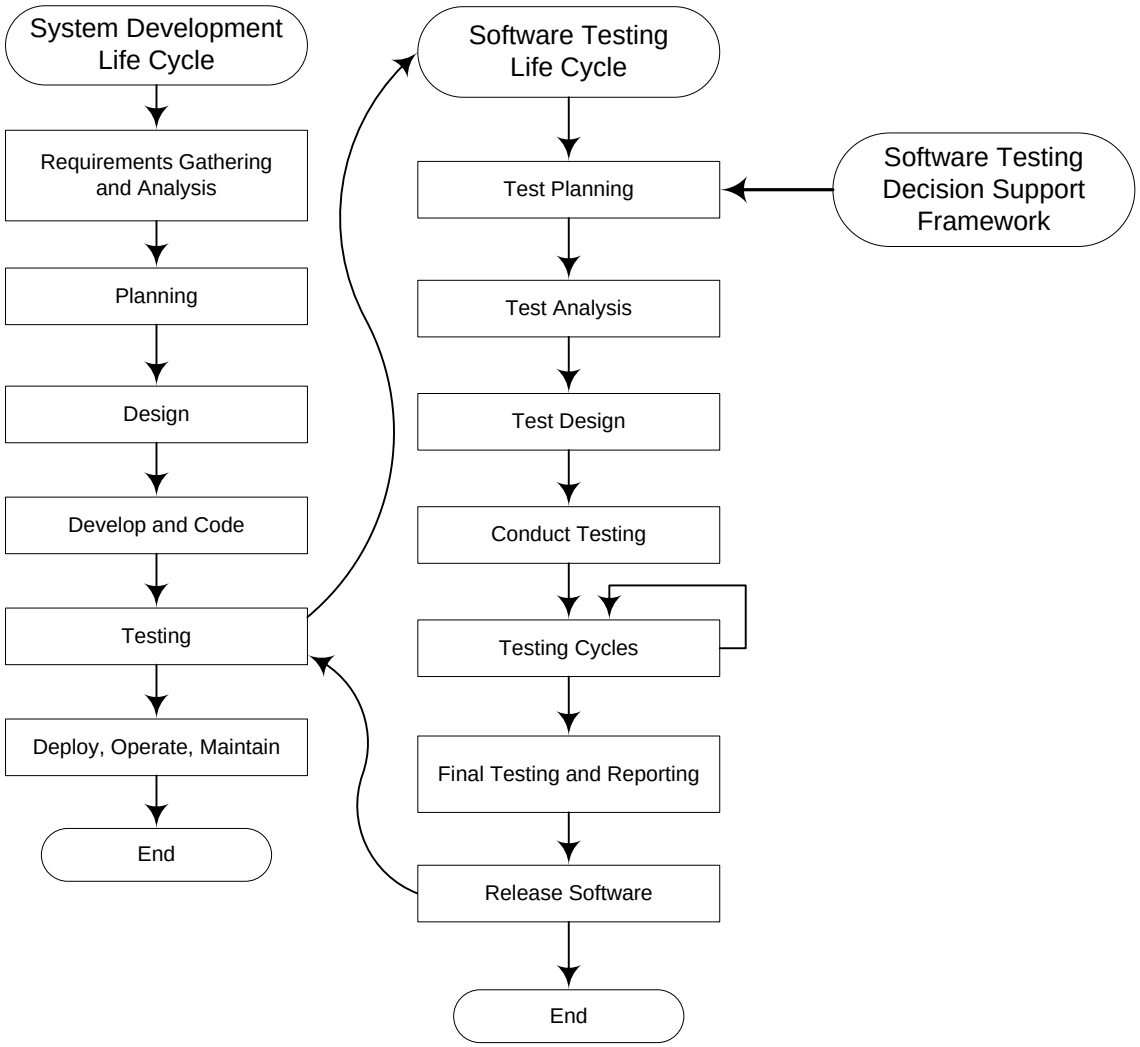


Figure 5.9: Decision support framework with SDLC

The details of a generic system development life cycle (SDLC) and software testing life cycle (STLC) were discussed in Chapter 3. The relationships between these two cycles are shown in Figure 5.9. The STLC is related to one phase of the SDLC, testing. The software testing DSF section shown in Figure 5.9 is used to assist and support the software test manager in test planning and risk management assessment for achieving successful software testing.

5.5 Software Testing Decision Support Framework Model

The previous sections have identified all the components of the decision support framework: the elements, factors, factor contributions, directional signed relationships and influence weightings. The development procedure for the decision support framework has used illustrative software test manager input (influence weightings and factor contributions), and has produced the software testing decision support framework model. This model is illustrated in Figure 5.10, and is the result of the software testing decision support framework being applied to a type of software to be tested. This illustration would be considered complete (for a particular type of software to be tested).

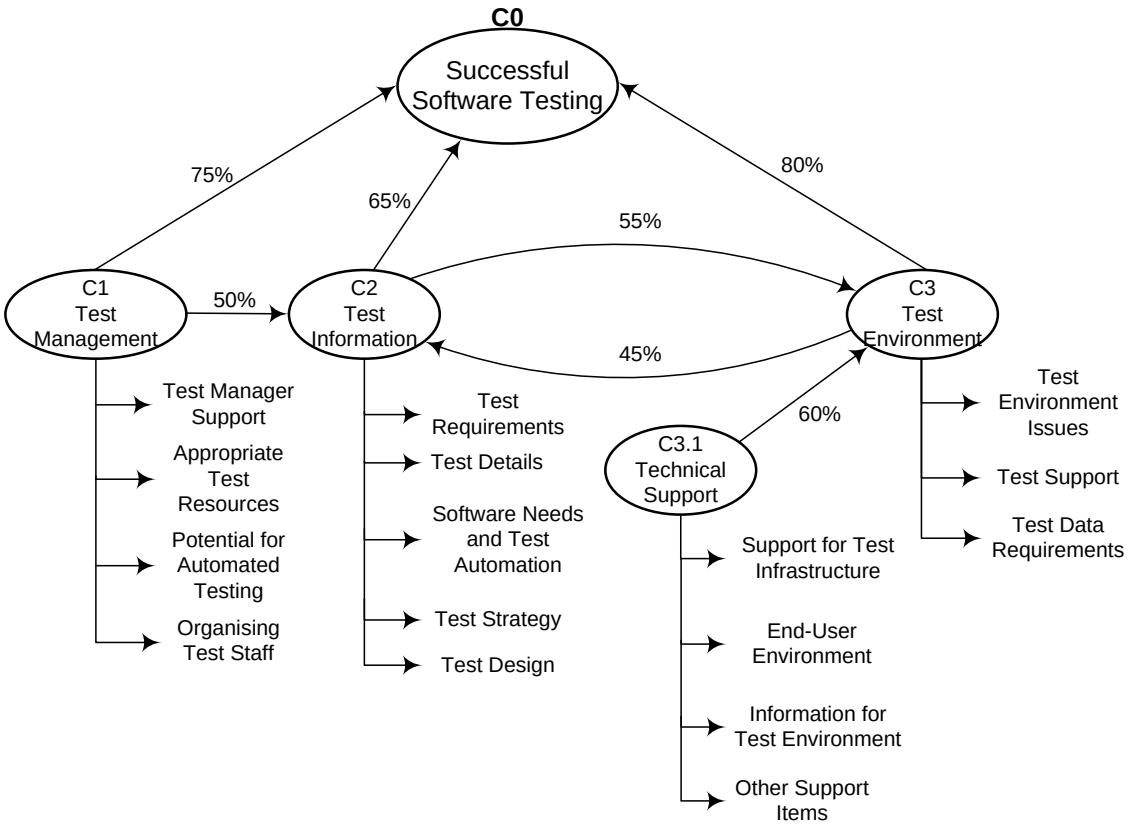


Figure 5.10: Completed software testing decision support framework

5.6 Development Procedure Outcomes for the Decision Support Framework

There are two outcomes from the procedure used to develop the software testing decision support framework. These outcomes are the software testing decision support framework and the model that the framework produces. This latter outcome was achieved by providing illustrative software test manager input (influence weightings and factor contributions) during the development procedure.

Illustrative software test manager influence weights and factor contribution percentages are shown in Figure 5.11. Technically, Figure 5.11 illustrates the completed software testing decision support framework model for some type of software which is planned to be tested, in accordance with the software test manager's input and in the context of the organisation.

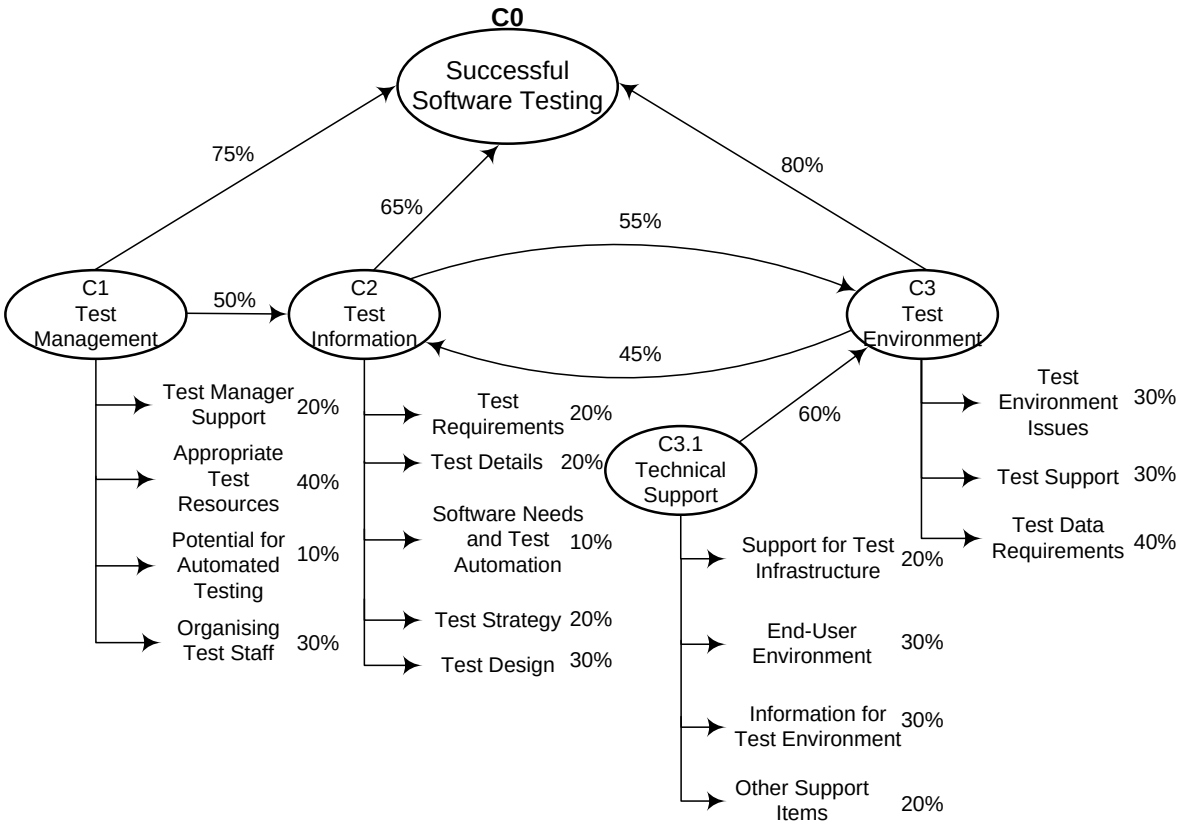


Figure 5.11: Software testing decision support framework model completed

The completed DSF model shown in Figure 5.11 includes:

- all elements;
- all directional signed relationships with their influence weightings;
- influence weightings between elements;
- influence weightings between elements and the goal;
- factor contributions for the elements; and
- factor contribution percentages.

Production of the decision support framework model has demonstrated the first step in the application of the decision support framework, by the software test manager. Figure 5.12

illustrates graphically the sequence of actions for the application of a software testing decision support framework, for any type of software application that is to be tested.

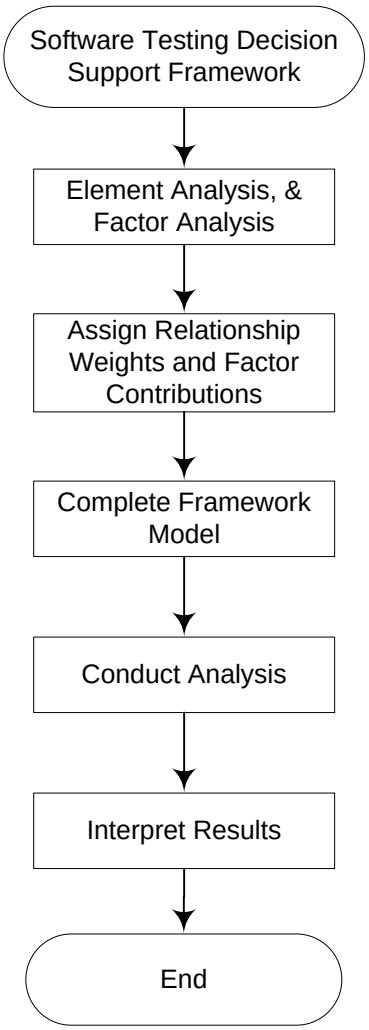


Figure 5.12: Application activities for the decision support framework

These actions fall logically into a two-step procedure. The two-step procedure the software test manager uses to apply the decision support framework is described next.

1) Use the decision support framework to model their particular software testing situation.
   a) Analyse the elements, and the factors of each element.
   b) Assign relationship weightings and factor contributions to complete the decision support framework model.
2) Evaluate the decision support framework model that results from step one.
   a) Conduct static and dynamic evaluation.
   b) Interpret the results of the evaluation.

The software test manager applies analytical techniques to the DSF model. These analytical techniques enable the software test manager to statically and dynamically evaluate, and interpret the DSF model for a particular type of software. This completes step two of the application of the DSF.

Chapter 6 will discuss the analytical techniques that the test manager could use to evaluate their particular decision support framework based on the type of software to be tested.

5.7 Summary

This chapter has presented the development of a software testing decision support framework. The framework was based on a generic decision support framework. The generic decision support framework can potentially be applied to any problem domain. It has been shown where the software testing decision support framework fits into the overall system development and software testing life cycles. The framework development has been discussed, and the model that results from software test manager input has been further described. The decision support framework elements, directional signed relationships, and element factors have been identified and discussed. The decision support framework provides a basis for the software test manager to analyse and interpret the risk associated with testing a particular type of software. This should be done from the DSF's initial state and for any subsequent changes to that state. The analytical techniques that provide a means to achieve risk management assessment are discussed and illustrated in Chapter 6.

Software testing is an expensive component of the system development process. Any decision support framework can only enhance and make the job of the test manager a little easier (Ghezzi, Jazayeri & Mandrioli, 1991).

- ||| -

CHAPTER 6. USING THE DECISION SUPPORT FRAMEWORK

The following diagram provides an outline and overview of the conceptual structure of this chapter.

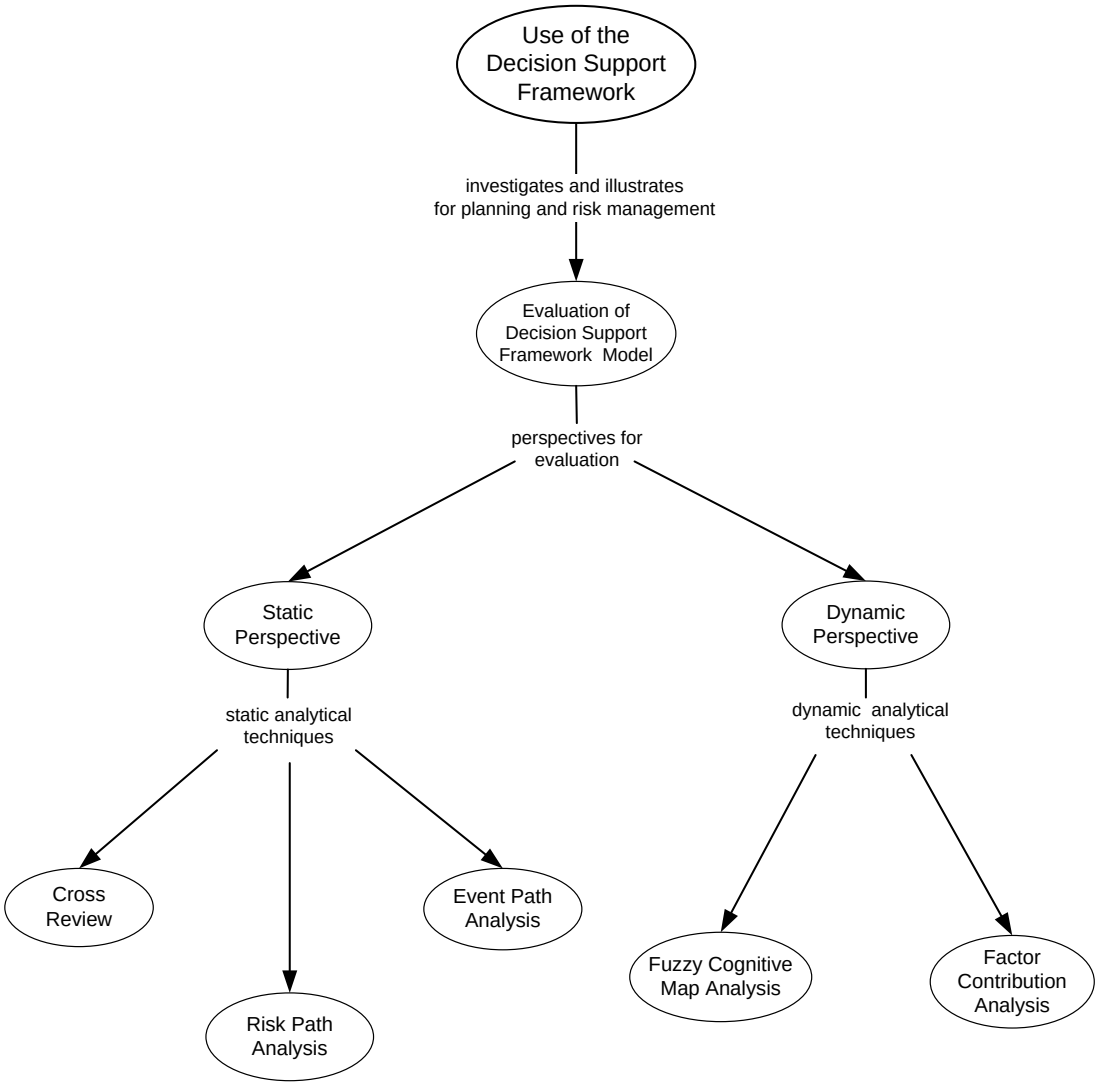


Figure 6.1: Conceptual structure of Chapter 6

6.1 Introduction

This chapter explores how the software testing decision support framework (DSF) can be used to assist software test managers in their test planning and risk management for successful software testing. Test planning for successful software testing enables the test manager to address the areas that need to be considered for the type of software to be tested. Various analytical techniques are presented and illustrated. With each technique the software test manager can ensure that risk management, using their software testing DSF model, is planned for and more fully understood. These techniques provide a more informed basis to assess the risk level in not achieving successful software testing, before testing and during testing.

These techniques can be used with post-test evaluation, if it occurs. Overall the test manager can employ a broad focus on software testing and assess the associated risks in the context of the constraints imposed within their organisation.

When the term decision support framework or the acronym DSF is used, unless explicitly specified otherwise, it refers to the specific software testing decision support framework developed in this research.

6.2 Approaches to Analysis and Interpretation

This section describes the various approaches developed to accompany the DSF for use by the software test manager.

6.2.1 Analytical Perspectives

There are two analytical perspectives that are used in analysing the DSF's resulting model:

1. static perspective; and
2. dynamic perspective.

From the static perspective, analysis of the DSF model is done to check for completeness, and in support of test planning to promote a better understanding of the particular type of software to be tested. The software test manager uses any or all of the following techniques:

- cross review;
- risk path analysis (RPA); and
- event path analysis (EPA).

From the dynamic perspective, analysis of the DSF model is done for analysing risk management issues, using either or both of the following techniques:

- fuzzy cognitive map (FCM); or
- factor contribution analysis (FCA).

Each of the techniques above adopts a different perspective and analyses different aspects of the DSF. It is up to the test manager to decide whether to use one analytical perspective, or both analytical perspectives, as developed in this research.

6.2.2 Software Scenarios

Analysis and interpretation of the DSF model, for a software application or system to be tested, will be illustrated using two software scenarios. Software scenario #1 was first shown in Chapter 5. This completed model for the DSF testing scenario is presented again in Figure 6.2.

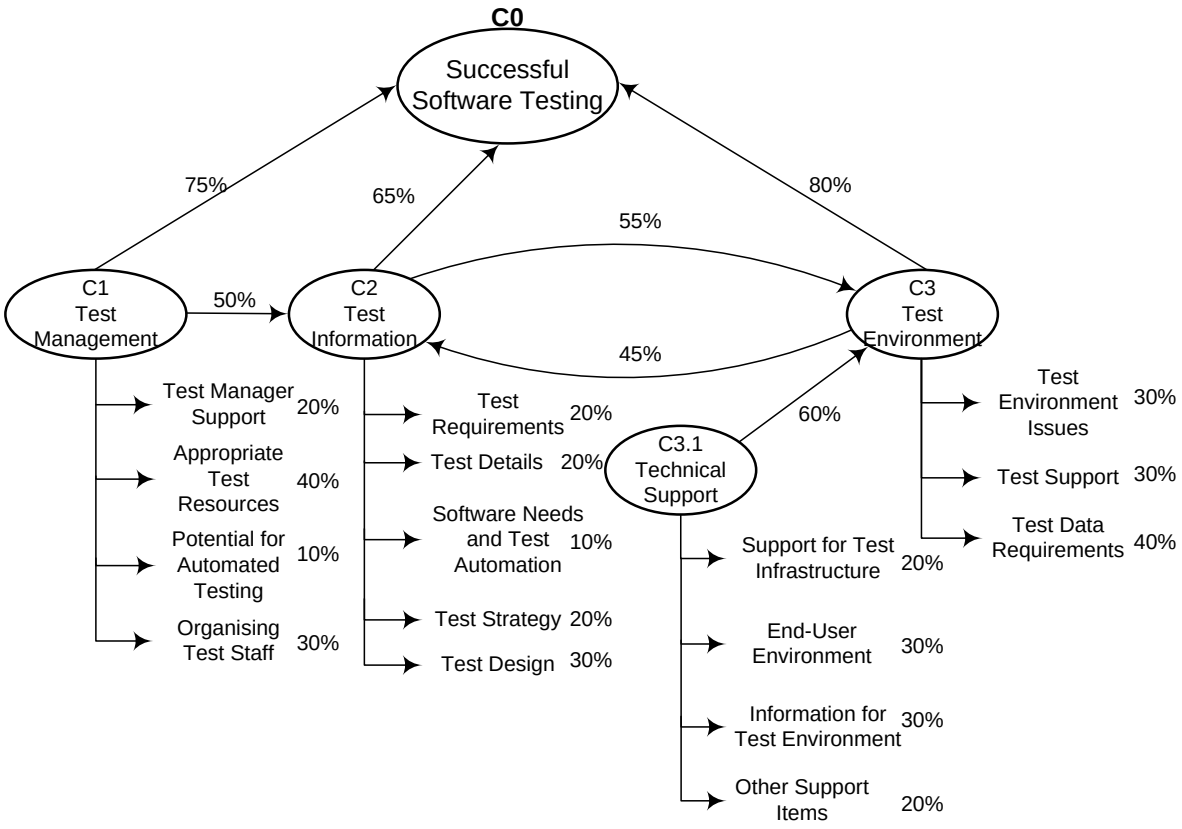


Figure 6.2: Detailed DSF model for software scenario #1

The software test manager takes their particular software to be tested and using the DSF should come up with a model like the one shown in Figure 6.2. Individual factors and their percentages for each element are not needed in using the DSF analytical techniques. Therefore, Figure 6.2 has been simplified for discussion purposes, in this chapter, to the scenario shown in Figure 6.3.

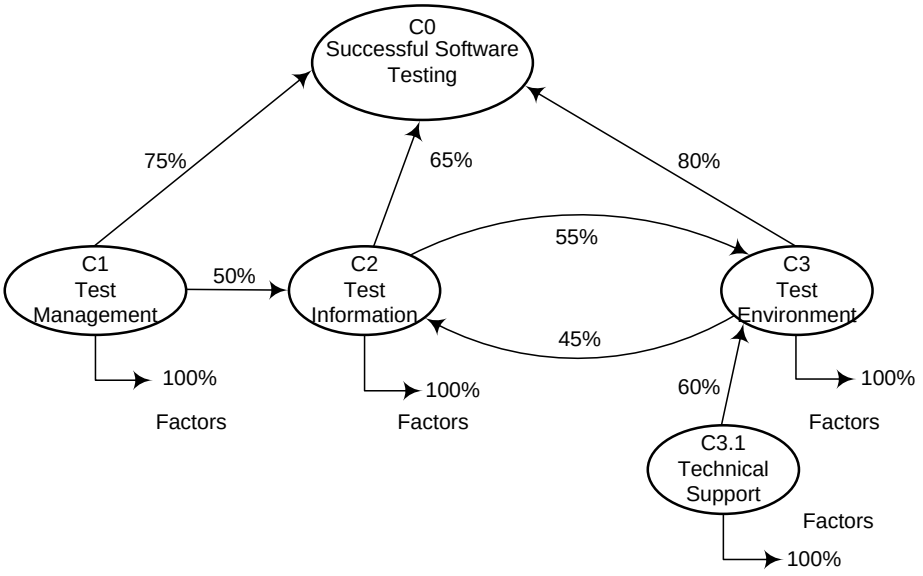


Figure 6.3: DSF model for software scenario #1

A second scenario of software to be tested, referred to as scenario #2, will be used to better understand and compare techniques presented in this chapter. Scenario #2 is illustrated in Figure 6.4. The individual element factors and their respective factor contribution percentages are not shown as they will not be used with the analytical techniques shown in this chapter.

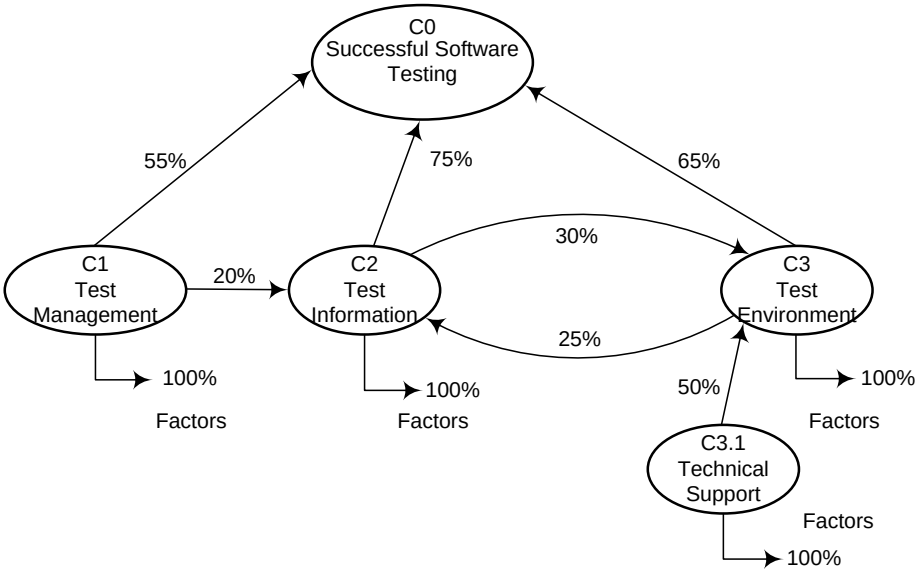


Figure 6.4: DSF model for software scenario #2

Both the static and dynamic analytical perspectives will use the scenarios illustrated in Figure 6.3 and Figure 6.4.

6.3 Static Perspective

There are three forms of static testing that could be used by the test manager in analysing the DSF for test planning and risk management:

1. cross review;
2. risk path analysis (RPA); and
3. event path analysis (EPA).

Each of these techniques is used to ensure that the DSF model accurately reflects a type of software to be tested. Irrespective of which analytical technique the software test manager is using, risk management of a type of software, that is to be tested, can be done quickly and informally. This form of risk management assessment has the advantage of not adding a great deal of effort or time to the software testing life cycle.

6.3.1 Static Analytical Techniques: Purpose and Description

The static analytical techniques provide information and support for test planning for the particular software that is to be tested. They also allow for a better understanding of the risk management issues, and their consequences, for the type of software to be tested.

6.3.1.1 Cross Review

A cross review is a general review of the completed DSF, to ensure that all issues relating to the resulting DSF model have been addressed. Generally this would be done based on the information contained in the model shown in Figure 6.2. The cross review also serves to ensure that those applicable sections of the test plan, as addressed by the DSF model, have been considered.

The cross review should be done by several reviewers or stakeholders, including senior testers, the software test manager, a senior application developer, or the software development manager. A general review of the completed DSF is done to ensure that test planning incorporates all known issues as outlined in the DSF model. The reviewers review the DSF model to provide advice on the appropriateness, and completeness, of the information in the DSF for the particular type of software to be tested. In other words, the test manager and other stakeholders review the DSF model, and make a reasonable attempt to ensure that all known aspects of the software to be tested have been taken into consideration in test planning. This review reduces the risk of missing, or incorrectly assessing influence weighting, and factor contribution percentages. However, this review may add a small amount of time to the

software test life cycle, but comes with the added benefit of developing a more well thought out test plan.

6.3.1.2 Risk Path Analysis (RPA)

Risk path analysis includes critical path analysis and total path weight analysis. The purpose of the RPA is to assist in test planning and identify the risk possibilities inherent in the completed DSF model for the type of software to be tested. RPA uses the information found on the directional signed relationships and the links between elements.

Critical Path Analysis

Critical path analysis looks at the path with the most linked elements and is therefore most likely to be affected by potential alterations in influence weightings. Critical path analysis is done in conjunction with the total path weight analysis.

Total Path Weight Analysis

Total path weight analysis develops an understanding of the influence weightings associated with the relationships (interconnected links or paths) in the DSF. A path is one or more relationships. This understanding of the influence weightings establishes a basis to determine the risk consequences of changes that might occur to the influence weights along the directional signed relationships.

6.3.1.3 Event Path Analysis (EPA)

Event path analysis analyses the effects of risk from events occurring or not occurring (i.e. when an element is activated), and the subsequent effect on achieving successful software testing.

6.3.2 Using Risk Path Analysis

This section describes and illustrates risk path analysis using critical path analysis and total path weight analysis, for the software scenarios (scenario #1 and scenario #2) shown in Figure 6.3 and Figure 6.4, respectively.

6.3.2.1 Software Scenario #1

Risk path analysis is based on the paths and influence weightings in the DSF model. To illustrate how this technique works, consider the paths (links between elements) shown in Figure 6.3.

Critical path analysis considers the paths with the most links, regardless of the influence weights. The critical paths are the paths that interconnect the most number of elements. In Table 6.1, rows C and I have the longest paths.

Total path weight analysis (TPWA) sums the influence weights for each element, via the directional signed relationships to other elements along each defined path. The analysis begins on the left most side of the DSF model and continues until all elements, via their directional signed relationships, have been identified. In any path, no element can be passed through twice as this would record the influence weight more than once. A total path weight that recorded the influence weight of an element more than once would give an inaccurate result for the risk of achieving the goal.

Table 6.1 shows the results for risk path analysis of the DSF model illustrated in Figure 6.3. The first column in the table is the row letter, which identifies each row and helps to clarify the discussion of the information in that row. The second column lists each element's beginning point for the third column. The third column begins with the element in column two and follows the path(s) from that element until it reaches the goal (the top most element). The fourth column (for illustration purposes only) shows the influence weights that have been accumulated along that path. This column is not necessary, but has been included to clarify how the final column has been calculated. The final column shows the total weight of the path. Table 6.1 has been ordered by the beginning element.

Table 6.1: DSF risk path analysis – scenario #1

	Beginning Elements	Path	Weightings	Total Path Weight
A	C1	C1 → C0	75	75
B	C1	C1 → C2 → C0	50 + 65	115
C	C1	C1 → C2 → C3 → C0	50 + 55 + 80	185
D	C2	C2 → C0	65	65
E	C2	C2 → C3 → C0	55 + 80	135
F	C3	C3 → C0	80	80
G	C3	C3 → C2 → C0	45 + 65	110
H	C3.1	C3.1 → C3 → C0	60 + 80	140
I	C3.1	C3.1 → C3 → C2 → C0	60 + 45 + 65	170

Table 6.2 (using information taken from Table 6.1) has been sorted by "total path weight" in descending order.

The analysis of multiple DSF models has been used to develop the following categories (for RPA) to manage and identify risk.

1. Low risk < 100
2. Medium risk ≥ 100 and < 150
3. High risk ≥ 150

The total path weight numbers from Table 6.1 were analysed and assigned a risk category, the final column in Table 6.2.

Table 6.2: RPA – scenario #1 sorted by total path weight

	Beginning Elements	Path	Total Path Weight	Risk Category
C	C1	C1 → C2 → C3 → C0	185	High
I	C3.1	C3.1 → C3 → C2 → C0	170	High
H	C3.1	C3.1 → C3 → C0	140	Medium
E	C2	C2 → C3 → C0	135	Medium
B	C1	C1 → C2 → C0	115	Medium
G	C3	C3 → C2 → C0	110	Medium
F	C3	C3 → C0	80	Low
A	C1	C1 → C0	75	Low
D	C2	C2 → C0	65	Low

For the DSF model in Figure 6.3, with the total path weights and risk category shown in Table 6.2, rows C and I show the paths with the highest number of elements. These are the critical paths. If any influence weight is adversely affected along these paths (shown in rows C or I) then there will be a high risk that the goal (C0) may not be achieved. Whereas changes to the total path weight in rows F, A or D, results in a lower risk of not achieving the goal (C0).

Looking at Table 6.2, C2 → C3 occurs in rows C (combined path weight 185) and E (combined path weight 135) for a total path weight of 320. Again looking at Table 6.2, C3 → C2 occurs in rows I (combined path weight 170) and G (combined path weight 110), for a total path weight of 280. All other path combinations are far less than these two. Thus, any change to the influence weights between C3 and C2 (either direction) will have the greatest impact on the risk of achieving the goal (C0).

6.3.2.2 Software Scenario #2

To better understand the different static perspectives, let us compare Table 6.2, developed from Figure 6.3, with a second scenario. The second scenario will be used to compare total

path weight analysis along with critical path. The following Table 6.3 represents scenario #2, ordered the same way as Table 6.1.

Table 6.3: DSF risk path analysis – scenario #2

	Beginning Elements	Path	Weights	Total Path Weight
A	C1	C1 → C0	55	55
B	C1	C1 → C2 → C0	20 + 75	95
C	C1	C1 → C2 → C3 → C0	20 + 30 + 65	115
D	C2	C2 → C0	75	75
E	C2	C2 → C3 → C0	30 + 65	95
F	C3	C3 → C0	65	65
G	C3	C3 → C2 → C0	25 + 75	100
H	C3.1	C3.1 → C3 → C0	50 + 65	115
I	C3.1	C3.1 → C3 → C2 → C0	50 + 25 + 75	150

Table 6.3 is ordered by beginning elements. To begin the comparative analysis, Table 6.3 is sorted by "total path weight" in descending order with the result shown in Table 6.4.

Table 6.4: RPA – scenario #2 sorted by total path weight

	Beginning Elements	Path	Total Path Weight	Risk Category
I	C3.1	C3.1 → C3 → C2 → C0	150	High
C	C1	C1 → C2 → C3 → C0	115	Medium
H	C3.1	C3.1 → C3 → C0	115	Medium
G	C3	C3 → C2 → C0	100	Medium
B	C1	C1 → C2 → C0	95	Low
E	C2	C2 → C3 → C0	95	Low
D	C2	C2 → C0	75	Low
F	C3	C3 → C0	65	Low
A	C1	C1 → C0	55	Low

The analysis uses the following categories just as was done in scenario #1.

1. Least risk < 100
2. Medium risk ≥ 100 and < 150
3. High risk ≥ 150

Since only one total path weight (in Table 6.4) exceeds or equals 150, it has the highest risk based on its total path weight and would need to be considered first. This is considered a critical path. In comparing Table 6.2 with Table 6.4, the effects can be seen of the influence

weightings assigned by the test manager for the different types of software to be tested. Row C has the highest total path weight (185) in Table 6.2, and row I has the highest total path weight (150) in Table 6.4. Thus, any effect on any of the influence weights along these paths will have the highest risk impact, positive or negative, on achieving the goal.

6.3.2.3 Scenario #1 and Scenario #2

Scenario #1 is based on testing one type of software and scenario #2 is based on the testing of a different type of software. The respective influence weightings in these scenarios are different. Critical path analysis considers the path with the most interconnected links. In Tables 6.2 and 6.4, rows C and I have the longest paths (most number of elements affected). Table 6.2 shows that row C has the highest total path weight whereas row I in Table 6.4 has the highest total path weight. Comparing Table 6.2 and Table 6.4, shows the potential effect on the risk of achieving the goal, based on change to the influence weightings that may occur along paths between elements. For scenario #1 the critical path with the highest influence weight is the path C1 $\rightarrow$ C2 $\rightarrow$ C3 $\rightarrow$ C0 (row C in Table 6.2). Any changes of the influence weightings along this path will potentially influence the achieving of the goal more rapidly than the changes that affect paths with single links, and low total influence weights, such as C3 $\rightarrow$ C0, C1 $\rightarrow$ C0 or C2 $\rightarrow$ C0.

For scenario #2 the critical path with the highest influence weight is the path C3.1 $\rightarrow$ C3 $\rightarrow$ C2 $\rightarrow$ C0 (row I in Table 6.4), which is very different from scenario #1. Change of the influence weightings along this path will potentially influence the achieving of the goal more rapidly than the changes that affect paths with single links, and low total influence weights, such as C2 $\rightarrow$ C0, C3 $\rightarrow$ C0 or C1 $\rightarrow$ C0.

Table 6.5 compares risk path analysis for scenario #1 and scenario #2. The paths in the table for scenario #1 are shown first, followed by the paths of scenario #2.

Table 6.5: Comparing RPA scenario #1 with scenario #2

Row	Table	Total Path Weight	Critical Path	Risk Category
C	6.2	185	C1 $\rightarrow$ C2 $\rightarrow$ C3 $\rightarrow$ C0	High
I	6.2	170	C3.1 $\rightarrow$ C3 $\rightarrow$ C2 $\rightarrow$ C0	High
H	6.2	140	C3.1 $\rightarrow$ C3 $\rightarrow$ C0	Medium
I	6.4	150	C3.1 $\rightarrow$ C3 $\rightarrow$ C2 $\rightarrow$ C0	High
C	6.4	115	C1 $\rightarrow$ C2 $\rightarrow$ C3 $\rightarrow$ C0	Medium
H	6.4	115	C3.1 $\rightarrow$ C3 $\rightarrow$ C0	Medium

Table 6.5 demonstrates the effects of the different influence weightings for the type of software to be tested. The effects between the two software scenarios can be seen in the different elements composing the paths, the total path weights and the risk categories that were identified for each path. This table also illustrates the critical paths and weights where the greatest risks occur on the effect of achieving goal. The software test manager looks at these paths (such as scenario #1 and scenario #2), and if changes occur, he/she can quickly analyse the potential effects on test planning and risk management in achieving the goal.

6.3.3 Using Event Path Analysis

Event path analysis evaluates success/risk on the goal as a direct consequence of one or more elements occurring. Event path analysis is based on the activation of elements and the subsequent event path that occurs, and the effect of that event path on success/risk of achieving the goal. An event is the activation of a single element, and an event path is the activation of one or more elements. An event path consists of one or more activated elements, and can be up to three elements long.

Event path analysis works with what-if scenarios of the form: What are the success/risk consequences on the goal if one or more element events occur? When an element event occurs, its affect is analysed on the element level above it, not to the elements that are lateral or below it. Thus, only direct elements to goal paths are used. Therefore, if an event occurs to a secondary level element it is evaluated to the primary level that is in turn evaluated to the goal. In any event path, no element can be activated twice as this would record the influence weight more than once. An event path weight that recorded the influence weight of an activated element more than once would give an inaccurate result for the success/risk of achieving the goal.

6.3.3.1 Software Scenario #1

Event path analysis is calculated by first determining the average of the influence weights over the number of relationships occurring along the path(s) being analysed. This average is then expressed as a percentage by multiplying by 100, and represents the success of the event(s) occurring. Success of the event occurring can be determined by the following generic formula. $Success = \left(\sum_{i=1}^{n} W_i \Big/ \#events \right) \times 100$, W_i represents the influence weight of each event. Risk of the event(s) not occurring is simply 100 minus the success of the event(s) occurring, that is Risk = 100 - Success. Risk can be determined by using the following

generic formula. $Risk = 100 - \left(\left(\sum_{i=1}^{n} W_i \Big/ \#events \right) \times 100 \right)$, W_i represents the influence weight of each event.

In understanding event path analysis consider row 2 of Table 6.6. There are two element events that occur and are to be analysed: Test management (C1) and Test information (C2). Event path analysis would be calculated for C1 and C2 as follows: C1 $\rightarrow$ .75 $\rightarrow$ C0 + C2 $\rightarrow$ .65 $\rightarrow$ C0 = 1.40. Then (1.40 ÷ # of events) × 100 = success of C1 and C2 occurring (70%). The risk is assessed at 30% (100 - success), meaning that there is an estimated risk factor of 30% that the software will not be successfully tested, based on just the two events C1 and C2 occurring. This does not mean that events C3.1 and C3 will not occur. The analysis only considers elements or combinations thereof occurring directly on the goal.

In row 10 of Table 6.6 four events Test management (C1), Test information (C2), Technical support (C3.1) and Test environment (C3) occur. It may seem that this row could have two possibilities, based on the interdependency and reciprocal influence path of C2 to C3, as well as C3 to C2. However, events are not analysed on lateral elements, thus only one possibility is applicable and that is the path towards the goal. That is, C2 $\rightarrow$C0, not C2 $\rightarrow$ C3 and C3 $\rightarrow$ C0, not C3 $\rightarrow$ C2.

For the DSF model in Figure 6.3 the assessment of the risk indicates the following:

Table 6.6: Event path analysis – scenario #1

	What-if Scenario (events occurring)	**Number of Events**	**Consequences**	**Success/Risk**
1	C1	1	C1—0.75→C0	75/25
2	C1, C2	2	C1, C2—1.40→C0	70/30
3	C2, C3	2	C2, C3—1.45→C0	72.5/27.5
4	C1, C2, C3	3	C1, C2, C3—2.20→C0	73.3/26.7
5	C3.1, C3	2	C3.1, C3—1.40→C0	70/30
6	C3.1, C3, C2	3	C3.1, C3, C2—2.05→ C0	68.3/31.7

7	C1, C3	2	C1, C3 —1.55→ C0	77.5/22.5
8	C1, C3.1	3	C1, C3.1 —2.15→ C0	71.7/28.3
9	C2, C3.1	3	C2, C3.1 —2.05→ C0	68.3/31.7
10	C1, C2, C3.1, C3	4	C1, C2 C3.1, C3 —2.80→ C0	70/30

6.3.3.2 Software Scenario #2

For the DSF model in Figure 6.4, using event path analysis indicates the following:

Table 6.7: Event path analysis – scenario #2

	What-if Scenario (events occurring)	Number of Events	Consequences	Success/Risk
1	C1	1	C1 —.55→ C0	55/45
2	C1, C2	2	C1, C2 —1.30→ C0	65/35
3	C2, C3	2	C2, C3 —1.40→ C0	70/30
4	C1, C2, C3	3	C1, C2, C3 —1.95→ C0	65/35
5	C3.1, C3	2	C3.1, C3 —1.15→ C0	57.5/42.5
6	C3.1, C3, C2	3	C3.1, C3, C2 —1.90→ C0	63.3/36.7
7	C1, C3	2	C1, C3 —1.20→ C0	60/40
8	C1, C3.1	3	C1, C3.1 —1.70→ C0	56.7/43.3
9	C2, C3.1	3	C2, C3.1 —1.90→ C0	63.3/36.7
10	C1, C2, C3.1, C3	4	C1, C2 C3.1, C3 —2.45→ C0	61.25/38.75

In row 2, Test management (C1) + Test information (C2) have a risk factor of 35% that the software may not achieve successful testing based on just those two events occurring.

6.3.3.3 Comparing Scenario #1 and Scenario #2

Table 6.8 shows the risk percentages for each what-if scenario of both software scenarios.

Table 6.8: Comparing EPA scenario #1 with scenario #2

	What-if Scenario (events occurring)	Software Scenario #1 (risk)	Software Scenario #2 (risk)
1	C1	25	45
2	C1, C2	30	35
3	C2, C3	27.5	30
4	C1, C2, C3	26.7	35
5	C3.1, C3	30	42.5
6	C3.1, C3, C2	31.7	36.7
7	C1, C3	22.5	40
8	C1, C3.1	28.3	43.3
9	C2, C3.1	31.7	36.7
10	C1, C2, C3.1, C3	30	38.75

For scenario #2, the influence weights between the elements and the goal (except for C2 $\rightarrow$ C0), or between the elements and the level above it, are significantly less than in scenario #1. This accounts for the increase in risk percentage, for each what-if scenario in software scenario #2, compared with scenario #1. The increase in risk percentage implies that some external influences to the situation, which are not directly portrayed by the DSF, play a decisive role in the successful testing of the software. If this is of a concern to the software test manager, they need to investigate this situation before completing their test plan. As indicated, the weightings for these two scenarios are arbitrary, and may not be representative of specific types of software that are to be tested.

6.3.3.4 Event Path Analysis Evaluating Changes to DSF

The test manager can further use EPA information to re-evaluate success/risk if the initial DSF model should change. What this says is, the test manager can quickly redo the calculations and arrive at new success/risk estimates. For example, what-if in scenario #2, C2 $\rightarrow$ C0 influence weight suddenly changes from 75% to 55%. The following table illustrates what would happen.

Table 6.9: Event path analysis – initial DSF for scenario #2 changed

	What-if Scenario (events occurring)	New Consequences (following change in influence weight)	Original Success/Risk	New Success/Risk
1	C1	C1—.55→ C0	55/45	no change
2	C1, C2	C1, C2—1.10→C0	65/35	55/55
3	C2, C3	C2, C3—1.20→C0	70/30	60/40
4	C1, C2, C3	C1, C2, C3—1.75→C0	65/35	58.3/41.7
5	C3.1, C3	C3.1, C3—1.15→C0	57.5/42.5	no change
6	C3.1, C3, C2	C3.1, C3, C2—1.70→ C0	63.3/36.7	56.7/43.3
7	C1, C3	C1, C3 —1.20→C0	60/40	56.25/43.75
8	C1, C3.1	C1, C3.1—1.70→C0	56.7/43.3	no change
9	C2, C3.1	C2, C3.1—1.70→C0	63.3/36.7	56.7/43.3
10	C1, C2, C3.1, C3	C1, C2 ↘ —2.25→ C0 C3.1, C3 ↗	61.25/38.75	56.25/43.75

Table 6.9 demonstrates that event path analysis is a suitable analytical technique that is sensitive enough to distinguish differences in success/risk on the goal, which reflect changed software testing conditions. The changed software testing conditions are captured in changes to one or more influence weightings of the DSF model. The changed influence weightings are determined by the software test manager based on their knowledge of and experience with software testing.

6.4 Dynamic Perspective

The dynamic perspective encompasses the use of Fuzzy Cognitive Maps (FCMs), based on the influence weightings, and the application of Factor Contribution Analysis (FCA) based on element factors.

6.4.1 Software Scenarios: a Reminder

Figure 6.3 (software scenario #1) and Figure 6.4 (software scenario #2) can be used with FCMs and FCA. For convenience, to avoid the need to refer to figures located earlier in the chapter, Figure 6.3 and Figure 6.4 are reproduced in this section as Figure 6.5 (software scenario #1) and Figure 6.6 (software scenario #2) respectively.

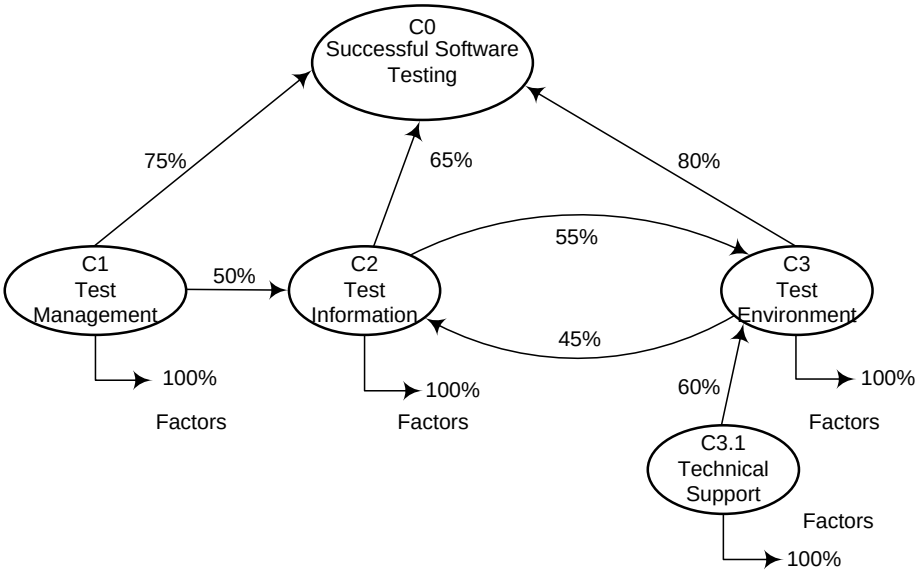


Figure 6.5: DSF model for software scenario #1 – a reminder

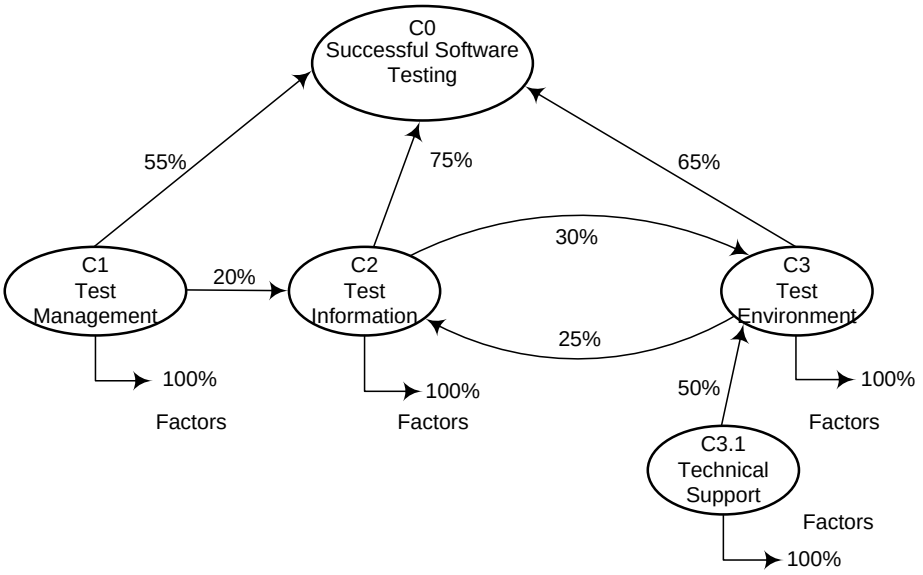


Figure 6.6: DSF model for software scenario #2 – a reminder

Individual percentages for each factor are not material to the dynamic analysis, as only the total of 100% is needed, as shown in Figure 6.5 and Figure 6.6. Once the analysis has been done, the test manager would be concerned with which factor or factors require additional and a more in-depth review.

6.4.2 Dynamic Analytical Techniques: Purpose and Description

This section discusses the purpose of the dynamic analytical techniques.

6.4.2.1 Fuzzy Cognitive Maps

In this research fuzzy cognitive maps (FCMs) are used to analyse the risk effects of individual elements of the DSF on achieving the goal of successful software testing. This involves the use of FCMs to analyse the DSF model's influence weightings to determine how single elements interact with other elements of the DSF. The output states of the FCM are analysed and form a basis for risk management of achieving the goal.

6.4.2.2 Factor Contribution Analysis

Factor contribution analysis is used to analyse risk impact of potential changes, based on one or more element's total factor contribution not meeting their goal of 100%. Thus, when the total factor contribution for one or more elements decreases: What will be the potential risk impact on achieving the goal of successful software testing?

6.4.3 Using Fuzzy Cognitive Maps

This section describes what fuzzy cognitive maps (FCMs) are and illustrates how to use FCMs to analyse the decision support framework.

6.4.3.1 Introduction to Fuzzy Cognitive Maps

Fuzzy cognitive maps (Kosko, 1991, 1997) are graph structures that provide a method to capture and represent complex relationships within an environment (which defines a boundary), to improve understanding of that environment. FCMs have been used to model problems with no data (Kosko, 1991, 1997; Smith & Eloff, 2000). Fuzzy cognitive maps can be used for what-if analysis, where several alternative scenarios to a given situation are considered (Kosko, 1991, 1997). Causal concept nodes represent the environment behaviour within a fuzzy cognitive map. Concepts take values in the fuzzy interval [0, 1] (Kosko, 1997). The concepts are connected using arcs (described by some authors as arrows or edges) showing the relations between concepts. Arcs take values in the fuzzy interval [-1,1] (Kosko, 1991). The development of FCMs is based on the utilisation of domain experts' knowledge. This expert knowledge is used to identify concepts and the degree of influence between the concepts.

Use of FCMs to analyse the decision support framework can only demonstrate general application of the decision support framework. In contrast, validation of the decision support framework involves specific, concrete applications of that framework. Concrete applications of the decision support framework can use the case study method to examine different

organisations in an industry environment. Validation of the framework is outside the boundary of this research.

6.4.3.2 The Dynamical System of Fuzzy Cognitive Maps

FCMs are suitable for what-if analysis; where alternative scenarios are considered for a given situation, for example, where risk needs to be assessed for risk management. This research adopts the method of Kosko (1991), where each scenario is used to test a selected concept. Thus, elements are selected from the DSF, and tested to determine their interactions with other elements in the decision support framework, and ultimately their impact on the goal of successful software testing.

Solutions to the scenarios are generated by the FCM dynamical system. For each input state (captured in the scenario), the FCM maps the inputs to output equilibrium states, also known as end states (Kosko, 1997). Equilibrium states are defined as states that are stable or in balance (Salmeron, 2010). For simple FCMs, such as the one constructed for this research to analyse the DSF, the path to the equilibrium state ends in a fixed point or limit cycle (Kosko, 1997). A fixed point is a unique vector, and equilibrium is established because the unique vector repeats the previous vector. A limit cycle is formed when two or more vectors are repeated, that is, the end state cycles between the fixed vector states (Salmeron, 2010). Equilibrium occurs when the limit cycle is repeated. The output equilibrium state is the answer inferred to the causal what-if question, which was used to generate the scenario (Kosko, 1997). The inference is determined using the FCM inference mechanism.

The FCM inference mechanism involves standard matrix multiplication. The initial state vector of concepts S_0, is a "1 x n" vector – [C_1, . . . , C_n] – and is multiplied by the FCM influence weighting matrix, generating a new state vector S_1 as the next step. This process is repeated until the dynamical system reaches equilibrium. The influence weighting matrix is an "n x n" matrix and each of its elements represents the influence weighting values between concept elements. Values of concepts in new state vectors are calculated using the update equation: $C_i^{t+1} = f\left(\sum_{\substack{j=1 \\ j\neq i}}^{n} W_{ji} C_j^t\right)$ (Kosko, 1997). The sigmoid function $f(x) = \frac{1}{1+e^{-\lambda x}}$ is used as the transformation function. The sigmoid function is used to transform the value of each concept of the state vector to the fuzzy interval [0, 1] (Kosko, 1997). Large values of λ approximate the binary threshold or step function (Kosko, 1991, 1997). In the step function,

$f(x) = 1$ if $x > T$ and $f(x) = 0$ if $x \leq T$, where T is the threshold value, taken from somewhere in the fuzzy interval [0,1] (Grant & Osei-Bryson, 2005; Kosko, 1991, 1997; Tsadiras, 2008). Thus, concepts are either on (1) or off (0), active or inactive. A binary threshold was adopted for this research (Kosko, 1997). The element to be tested is set to 1 in the input vector and in all the result vectors because it is a sustained input (Kosko, 1997).

6.4.3.3 Fuzzy Cognitive Maps Analysis – Software Scenario #1

This section uses fuzzy cognitive map analysis to examine what happens if individual elements of the decision support framework are fully activated (or occur fully). The dynamical system of the fuzzy cognitive map provides an inference for each element scenario, which is a global response of all the interacting elements of the framework.

To evaluate individual element scenarios the following steps are followed:

1. create an influence weightings matrix I, which lists the values of the relationships between the elements (concepts);
2. select an element to test its influence weighting on the potential decision;
3. create the initial state vector S_0 (for the framework this state vector is of the form [C0, C1, C2, C3, C3.1]). Set the element to be tested to 1 (on) and set all other elements to 0 (off). This ensures independent analysis of the test element. The element to be tested is modelled as a sustained input, so the test element is set to 1 in all the result vectors;
4. multiply S_0 by I to obtain the result vector S_1. Apply the binary threshold operation to the result vector. Apply the threshold function. In this FCM simulation the threshold is set to 0.15;
5. if the test element is not already set to 1 (on), reset it to 1;
6. repeat step 4 and step 5 with each result vector (S_n * I) until equilibrium is reached. Equilibrium is reached when a vector is repeated, i.e. the current iteration $S_n = S_{n-1}$;
7. take the previous result vector S_{n-1} for the analysis; and
8. repeat steps 2 to 6 for each scenario.

The influence weightings amongst the elements in Figure 6.5 can be displayed using the following influence weighting matrix I_1**.**

$$I_1 = \begin{array}{c} \\ C0 \\ C1 \\ C2 \\ C3 \\ C3.1 \end{array} \begin{array}{c} \begin{array}{ccccc} C0 & C1 & C2 & C3 & C3.1 \end{array} \\ \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.75 & 0.00 & 0.50 & 0.00 & 0.00 \\ 0.65 & 0.00 & 0.00 & 0.55 & 0.00 \\ 0.80 & 0.00 & 0.45 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.60 & 0.00 \end{pmatrix} \end{array}$$

The software test manager and other domain experts are required to determine the weights of the different directional signed relationships between the concept elements.

The process begins by looking at each element individually. This is done to isolate and analyse its overall influence on the goal C0. Since C0 is the goal that is being evaluated, if it was set to 1, there would be no need to assist the software testing manager, because the goal would have been 100% achieved.

First, analyse the influence of C1; so C1 is set to 1. This provides a means by which C1 can be examined independently of the other elements and their influences. This situation is represented by:

$\mathbf{S_0}$ = [0, 1, 0, 0, 0].

$\mathbf{S_0}$ = [0, 1, 0, 0, 0]

$\mathbf{S_0} * I_1$ = [0.75, 0, 0.50, 0, 0] – this becomes $\mathbf{S_1}$ (reset test element to 1), $\mathbf{S_1}$ = [1, 1, 1, 0, 0]

$\mathbf{S_1} * I_1$ = [1.40, 0, 0.50, 0.55, 0] – this becomes $\mathbf{S_2}$ (reset test element to 1), $\mathbf{S_2}$ = [1, 1, 1, 1, 0]

$\mathbf{S_2} * I_1$ = [2.20, 0, 0.95, 0.55, 0] – this becomes $\mathbf{S_3}$ (reset test element to 1), $\mathbf{S_3}$ = [1, 1, 1, 1, 0]

$\mathbf{S_3} = \mathbf{S_2}$ ∴ equilibrium has been reached.

Let us next set C2 to 1. Thus, C2 can be looked at independently of the other elements and their influences. This situation is represented by:

$\mathbf{S_0}$ = [0, 0, 1, 0, 0].

$\mathbf{S_0}$ = [0, 0, 1, 0, 0] – after each iteration, the test element will be set to 1 as it is the element under evaluation.

$\mathbf{S_0} * I_1$ = [0.65, 0, 0, 0.55, 0] – this becomes $\mathbf{S_1}$ = [1, 0, 1, 1, 0]

$\mathbf{S_1} * I_1$ = [1.45, 0, 0.45, 0.55, 0] – this becomes $\mathbf{S_2}$ = [1, 0, 1, 1, 0]

$\mathbf{S_2} = \mathbf{S_1}$ ∴ equilibrium has been reached.

Let us next set C3 to 1. Thus, C3 can be looked at independently of the other elements and their influences. This situation is represented by:

$\mathbf{S}_0$ = [0, 0, 0, 1, 0].

$\mathbf{S_0}$ = [0, 0, 0, 1, 0] – after each iteration, the test element will be set to 1 as it is the element under evaluation.

$\mathbf{S_0} * I_1$ = [0.80, 0, 0.45, 0, 0] – this becomes $\mathbf{S_1}$ = [1, 0, 1, 1, 0]

$\mathbf{S_1} * I_1$ = [1.45, 0, 0.45, 0.55, 0] – this becomes $\mathbf{S_2}$ = [1, 0, 1, 1, 0]

$\mathbf{S_2} = \mathbf{S_1}$ ∴ equilibrium has been reached.

Let us next set C3.1 to 1. Thus, C3.1 can be looked at independently of the other elements and their influences. This situation is represented by:

$\mathbf{S}_0$ = [0, 0, 0, 0, 1].

$\mathbf{S_0}$ = [0, 0, 0, 0, 1] – after each iteration, the test element will be set to 1 as it is the element under evaluation.

$\mathbf{S_0} * I_1$ = [0, 0, 0, 0.60, 0] – this becomes $\mathbf{S_1}$ = [0, 0, 0, 1, 1]

$\mathbf{S_1} * I_1$ = [0.80, 0, 0.45, 0.60, 0] – this becomes $\mathbf{S_2}$ = [1, 0, 1, 1, 1]

$\mathbf{S_2} * I_1$ = [1.45, 0, 0.45, 1.15, 0] – this becomes $\mathbf{S_3}$ = [1, 0, 1, 1, 1]

$\mathbf{S_2} = \mathbf{S_1}$ ∴ equilibrium has been reached.

Overall the analysis indicates the following results for each concept analysed.

Table 6.10: FCM results for element analysis – scenario #1

Elements Analysed	Element(s) Activated	Equilibrium Vectors	Influence Weights Results
Test management (C1)	C0, C2, C3	[1, 1, 1, 1, 0]	[1.40, 0, 0.95, 0.55, 0]
Test information (C2)	C0, C3	[1, 0, 1, 1, 0]	[1.45, 0, 0.45, 0.55, 0]
Test environment (C3)	C0, C2	[1, 0, 1, 1, 0]	[1.45, 0, 0.45, 0.55, 0]
Technical support (C3.1)	C0, C2, C3	[1, 0, 1, 1, 1]	[1.45, 0, 0.45, 1.15, 0]

The FCM converges to a fixed point for each scenario – single vectors are the result when equilibrium is reached, so definite answers can be obtained for each scenario. It can be seen that the goal (C0) is achieved in all cases (i.e. a 1 exists in each and every first cell of the equilibrium vectors); this indicates that each of the elements exhibit some influence on

successful software testing. It can be seen that C2 and C3 are activated for all equilibrium vectors.

Notice that C2 and C3.1 influence C3 and arrive at the same accumulative influence weighting. While C1 and C3 influence C2 and arrive at the same accumulative influence weighting.

6.4.3.4 Fuzzy Cognitive Maps Analysis – Software Scenario #2

This section uses fuzzy cognitive map analysis to examine what happens if individual elements of the software testing decision support framework are fully activated. The dynamical system of the fuzzy cognitive map provides an inference for each element scenario, which is a global response of all the interacting elements of the framework.

The influence weightings among the elements in Figure 6.6 can be displayed using the following influence weighting matrix I_2**.**

$$I_2 = \begin{array}{c} \\ C0 \\ C1 \\ C2 \\ C3 \\ C3.1 \end{array} \begin{array}{c} \begin{array}{ccccc} C0 & C1 & C2 & C3 & C3.1 \end{array} \\ \begin{pmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.55 & 0.00 & 0.20 & 0.00 & 0.00 \\ 0.75 & 0.00 & 0.00 & 0.30 & 0.00 \\ 0.65 & 0.00 & 0.25 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.50 & 0.00 \end{pmatrix} \end{array}$$

After applying the FCM analysis the results are listed in Table 6.9.

Table 6.11: FCM results for element analysis – scenario #2

Elements Analysed	Element(s) Activated	Equilibrium Vectors	Influence Weights Results
Test management (C1)	C0, C2, C3	[1, 1, 1, 1, 0]	[1.95, 0, 0.45, 0.30, 0]
Test information (C2)	C0, C3	[1, 0, 1, 1, 0]	[1.40, 0, 0.25, 0.30, 0]
Test environment (C3)	C0, C2	[1, 0, 1, 1, 0]	[1.40, 0, 0.25, 0.30, 0]
Technical support (C3.1)	C0, C2, C3	[1, 0, 1, 1, 1]	[1.40, 0, 0.25, 0.80, 0]

6.4.3.5 Comparing Scenario #1 and Scenario #2

A review of Table 6.9 and Table 6.10 shows that for both software scenarios, the equilibrium vectors are exactly the same for each element analysed. The deductive expectation is that the equilibrium vectors would be different for each scenario. This difference would reflect the different types of software analysed, and help to establish the usefulness of FCM analysis as

an analytical technique that the software test manager can use, to assess the risk effects of individual elements of the DSF on achieving the goal of successful software testing.

The same equilibrium vectors for each software scenario is a contradictory finding, which does not reflect the different types of software defined by these scenarios. The results suggest that FCM analysis is apparently unable to discriminate between the risk effects of the two different software scenarios. The influence weight percentages for each software scenario are significantly different (see Figure 6.5 and Figure 6.6). Nevertheless, the results move FCM analysis of the DSF model in the forward direction. The explanation for the contradiction alluded to above is as follows.

The DSF model is represented at a low level of granularity (despite the possibility of representing the factors of each element as nested concepts). The results of this initial use of FCM analysis of the DSF model have to be viewed carefully, with several conditions in mind. The results are based on the following conditions: a high level representation of the FCM model, application of the Kosko (1997) update equation (Kosko, 1997), and use of the Kosko (1997) binary threshold function (Kosko, 1997) to test the activation level of each element. These conditions can be seen as having a twofold role. First, the conditions establish the groundwork to examine further how to determine inferences that will discriminate between the risk effects of different types of software. Second, the conditions form the considerations to be borne in mind for refining how inferences are determined in future research on FCM analysis (see Chapter 8, Section 8.7.1).

6.4.3.6 Fuzzy Cognitive Map Considerations

The preceding discussion is underpinned by several considerations. The decision support framework is represented at a low level of granularity, despite encapsulating many concepts and relationships in elements C1 to C3.1 inclusive. This low granularity representation was chosen so as not to invoke inflexibility, and suggest too many concepts to confront the software test manager. Too many concepts could impose restrictions and force more difficulties for the software test manager to adjust or analyse the software testing decision support framework results. The framework is flexible and the software test manager can adjust the influence weightings of the framework to match their organisational circumstances and the characteristics of the software being tested.

6.4.4 Using Factor Contribution Analysis

In this section the discussion begins with what happens when an element's factor contribution fails to meet 100%. The discussion includes the changes to the influence weightings between elements and the goal, and the risk consequences on achieving the goal.

Factor contribution analysis is underpinned by the following basic rules and information.

1. The influence weighting attributed to each element will be ≥ 10% and ≤ 90%. This was discussed in Chapter 5, Section 5.2.3.
2. Factor contribution analysis is only concerned with an element's total factor contribution, and not the individual factor contributions for the element.
3. Risk management is based only on achieving the goal, and is not applicable to any individual element.
4. If a factor contribution loss is equal or greater than 90%, it has been determined through a set of simulations that the goal will not be achieved, thus the maximum loss for a factor contribution cannot exceed 90%.
5. Back tracking between elements is not permitted as factor contribution analysis would result in an endless loop (such would be the case for C2 → C3 → C2 → C3 → C2 → etc…).
6. The DSF model has a maximum of 16 possible factor contribution analysis scenarios. The possible scenarios are shown in Table 6.11. The analysis scenario "≤" means the total factor contribution, for that element, is less than or equal to 90%, and the analysis scenario "=" means the total factor contribution, for that element, is equal to 100%.

Table 6.12: Factor contribution analysis scenarios

	C1	C2	C3	C3.1
1	≤	≤	≤	≤
2	≤	≤	≤	=
3	≤	≤	=	=
4	≤	=	=	≤
5	≤	=	≤	=
6	≤	=	≤	≤
7	≤	=	=	=
8	=	≤	≤	≤

9	=	≤	≤	=
10	=	≤	=	≤
11	=	≤	=	=
12	=	≤	≤	=
13	=	=	≤	=
14	=	=	≤	≤
15	=	=	=	≤
16	=	=	=	=

To achieve successful software testing, factor contribution analysis is related to risk management as well as test planning. The following table shows the assessed risk categories used for factor contribution analysis, and how these categories are interpreted against changes in the influence weightings applicable to the goal.

Table 6.13: Factor contribution assessed risk categories

Risk Category	Factor Contribution Change	Influence Weighting Change Criteria (percentage of original weightings)
Low	≤ 10%	≥ 90%
Low-Medium	≤ 20% to > 10%	≥ 80% to < 90%
Medium	≤ 30% to > 20%	≥ 70% to < 80%
Medium-High	≤ 40% to > 30%	≥ 60% to < 70%
High	> 40%	< 60%

The influence weighting change criteria are based on the original influence weightings, compared with the new influence weighting(s) determined from the factor contribution change. For example, for the high risk category, the change from the original influence weighting to the new influence weighting is > 40%. This determination of new influence weighting(s) will be illustrated in the next three sections, which provides examples of how factor contribution analysis is done, and how to do risk assessment based on that analysis.

All examples and discussion in this section (Section 6.4.4) will be based on the DSF model shown in Figure 6.5 (software scenario #1), unless otherwise specified. For simplicity, each discussion will use C0, C1, C2, C3, and C3.1 instead of element names. The summary will use the element names as shown in Figure 6.5.

6.4.4.1 Factor Contribution Analysis – Test Management Scenario

The first scenario to be considered is: What happens when C1 (Test management) factor contribution fails to meet 100%? This scenario is shown in Figure 6.7.

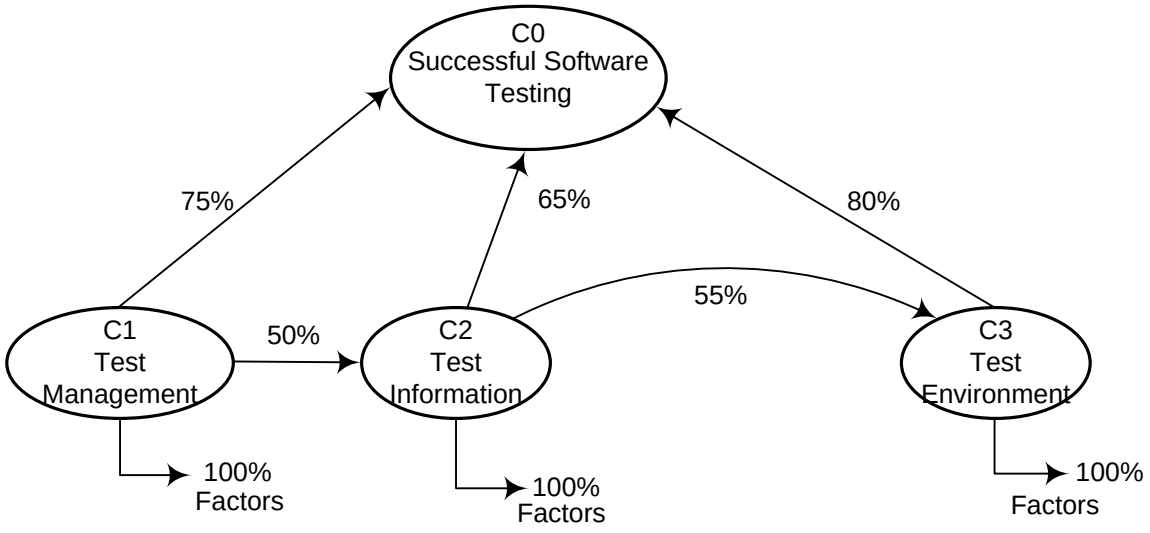


Figure 6.7: DSF factor contribution analysis – Test management

When C1 factor contribution falls below 100%, the following influence weightings are affected (shown in Figure 6.6):

C1 → C0
C1 → C2 → C0
C1 → C2 → C3 → C0

In other words the goal (successful software testing) and the elements C1, C2 and C3 will be affected. Let us say that C1 factor contribution is at 90% (not 100%). Each of the influence weighting percentages to the goal will be reduced by 10%. Thus:

C1 → C0 – 67.5%
C2 → C0 – 58.5%
C3 → C0 – 72%

The total effect on achieving the goal has been reduced from 100% to 90%. If the sum of the new influence weightings (198) on the goal is divided by the sum of the original influence weightings on the goal (220), and expressed as a percentage, then the result will be 90%. Using Table 6.12, the 10% loss in factor contribution on C1 shows that the effect has a LOW risk on achieving the intended goal.

6.4.4.2 Factor Contribution Analysis – Test Information Scenario

The second scenario to be considered is: What happens when C2 (Test information) factor contribution fails to meet 100%? This scenario is shown in the following diagram.

When C2 factor contribution falls below 100%, the following influence weightings are affected (shown in Figure 6.7):

C2 → C0
C2 → C3 → C0

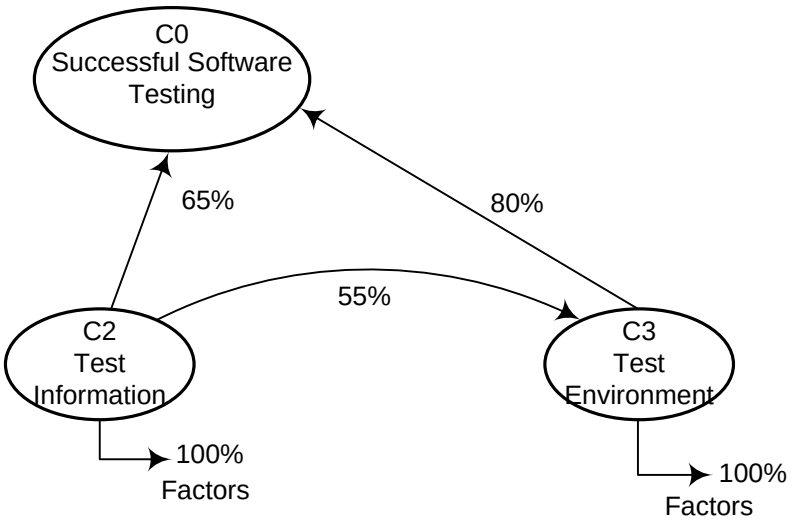


Figure 6.8: DSF factor contribution analysis – Test information

In other words the goal (successful software testing) and only the elements C2 and C3 will be affected. Let us say that C2 factor contribution is at 90% (not 100%). Then both C2 and C3 influence weighting percentages to the goal will be reduced by 10%.

Thus:

C2 → C0 – 58.5%
C2 → C3 → C0 – 72%

Note that there is no change to C1 influence on C0 as shown below.

C1 → C0 – 75%

The total effect on achieving the goal has been reduced. If the sum of the new influence weightings on the goal (205.5) is divided by the sum of the original influence weightings on the goal (220), and expressed as a percentage, the result will be 93.4%. Using Table 6.12, the 10% loss in factor contribution on C2 shows that the effect has a LOW risk on achieving the intended goal.

The next scenario to be considered is: What happens when C3 (Test environment) factor contribution fails to meet 100%? Elements C3 and C2 have an interdependent relationship in

the DSF model. Therefore, elements C3 and C2 react exactly the same to changes in total factor contribution of either element.

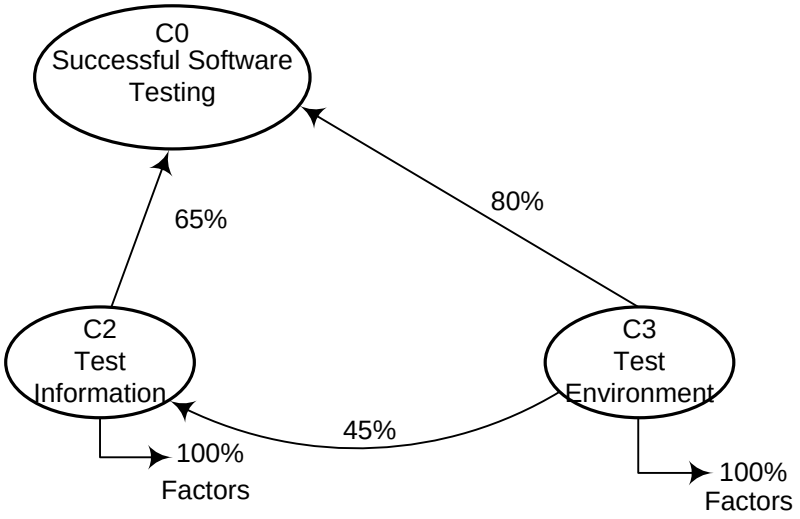


Figure 6.9: DSF factor contribution analysis – Test environment

When C3 factor contribution falls below 100%, the following influence weightings are affected (shown in Figure 6.9):

C3 → C0
C3 → C2 → C0

In other words the goal (successful software testing) and only C3 and C2 elements will be affected. Let us say that C3 factor contribution is at 90% (not 100%). Then both C3 and C2 influence weighting percentages to the goal will be reduced by 10%.

Thus:

C3 → C0 – 72%
C2 → C0 – 58.5%

Note that there is no change to C1 influence on C0 as shown below.

C1 → C0 – 75%

The total effect on achieving the goal has been reduced. If the sum of the new influence weightings on the goal (205.5) is divided by the sum of the original influence weightings on the goal (220), and expressed as a percentage, then the result will be 93.4%. Using Table 6.12, the 10% loss in factor contribution on C3 shows that the effect has a LOW risk on achieving the intended goal.

6.4.4.3 Factor Contribution Analysis – Technical Support Scenario

The third scenario to be considered is: What happens when C3.1 (Technical support) factor contribution fails to meet 100%? This scenario is shown in the following diagram.

When C3.1 factor contribution falls below 100%, the following influence weightings are affected (shown in Figure 6.9):

C3.1 → C3 → C0
C3.1 → C3 → C2 → C0

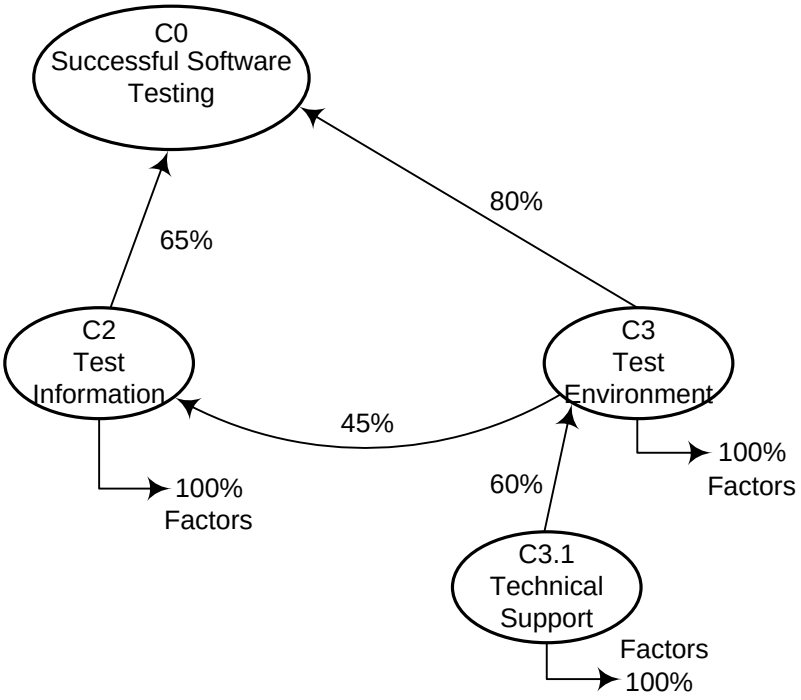


Figure 6.10: DSF factor contribution analysis – Technical support

In other words the goal (successful software testing) will be affected via C3 and C2. For example, say that factor contribution C3.1 is at 90% (a 10% decrease). Then both C3 and C2 influence weighting percentages to the goal will be affected by -10%.

Thus:

C3.1 → C3 → C0 – 72%
C3.1 → C3 → C2 → C0 – 58.5%

Note that there is no change to C1 influence on C0 as shown below.

C1 → C0 – 75%

The total effect on achieving the goal has been reduced. If the sum of the new influence weightings on the goal (205.5) is divided by the original influence weightings on the goal (220), and expressed as a percentage, then the result will be 93.4%. Using Table 6.12, the 10% loss in factor contribution on C3.1 shows that the effect has a LOW risk on achieving the intended goal.

6.4.4.4 Factor Contribution Analysis – Scenario Combinations

Scenarios with combinations of factor contributions occur when two, three or all four element's factor contributions fall below 100%. For example: What if: C1's and C3.1 factor contributions fall to 90%? Based on the DSF model shown in Figure 6.5 (software scenario #1), and using what has been discussed above, the following occurs.

Thus:

C1 → C0 – 67.5%
C2 → C0 – 52% and
C3 → C0 – 64%

In other words the goal (successful software testing) will be affected by the change to C1 and the change to C3.1. C1 and C3.1 influence weighting percentages will be reduced by 10%.

C1 is only affected by changes to its factor contributions. However, C1 affects both C2 and C3 influence weightings on the goal. C3.1 affects both C3 and C2 influence weightings on achieving the goal, but not C1. Thus, the change to C1 influence weighting on C0 will be 10%, while the change to the influence weightings of C2 and C3 on C0 will be 20%. The influence weighting change on C2 and C3 is additive. This observation can be generalised: when the influence weighting of a primary element on the goal is affected more than once, the total effect on the influence weighting is additive. The more complex example below is consistent with this generalised observation.

The total effect on achieving the goal has been reduced. If the sum of the new influence weightings on the goal (183.5) is divided by the original influence weightings (220), and expressed as a percentage, the result will be 83.4%. Using Table 6.12, the 10% loss in factor contribution on C1 and C3.1 shows that the effect has a LOW-MEDIUM risk on achieving the intended goal.

As another example: What if the following occurs?

Table 6.14: Scenario with the factor contribution of several elements reduced

Elements	Factor Contribution Reduced To	Total % Change	New Influence Weight on Goal
C1	95%	5%	71.3
C2	75%	60%	26.0
C3	90%	60%	32.0
C3.1	80%	n/a	n/a
		Total	129.3

The risk of achieving the goal has been substantially affected. If the sum of the new influence weightings on the goal (129.3) is divided by the sum of the original influence weightings (220), and expressed as a percentage, then the result will be 58.8%. Using Table 6.12, the loss in factor contribution across the four elements shows that the effect has a HIGH risk on achieving the intended goal.

6.4.4.5 Further Discussion of Factor Contribution Analysis

In the previous section the changes to factor contributions and their effect on the influence weights to the goal has been examined. In Section 6.4.4.1 the effects of the Test management factors not being 100% has been examined. In Section 6.4.4.2 the effects of the Test information and Test environment has been examined. In Section 6.4.4.3 the effects of Technical support has been examined. Finally, in Section 6.4.4.4 the effect of changes to the factor contributions of several of the elements has been examined. For the most part, issues that affect software testing generally affect more than one factor contribution. Thus, the most probable scenario that would occur in practice has been illustrated in Section 6.4.4.4.

Minor changes in Test management's factor contribution of 5% or 10% will have only a minor but rippled effect on achieving the goal of successful software testing. Test managers need to be aware of decreases in factor contributions and, where possible, adjust their test planning and risk management plan accordingly.

As changes to an element's factor contributions occur, test planning needs to consider consequences and anticipate alternative actions. For example, if the Technical environment test data factor contribution drops from 40% to 20%, test planning needs to consider what alternatives need to be put in place to achieve the goal.

Factor contribution analysis and fuzzy cognitive map analysis could be done together. First, FCA would be done, followed by FCM analysis on the resulting influence weight changes.

6.5 Summary

This chapter has described how a test manager would apply different analytical techniques to evaluate and use the DSF. These techniques can assist the test manager in their planning and risk management of successful software testing. These analytical techniques have been discussed from two perspectives:

1. static perspective; and
2. dynamic perspective.

For the static perspective, the discussion included the analytical techniques in showing how a software test manager can use the DSF model:

- cross review;
- risk path analysis; and
- event path analysis.

For the dynamic perspective, the evaluation and interpretation of risk has used the analytical techniques:

- fuzzy cognitive map (FCM); and
- factor contribution analysis (FCA).

Software test managers can select any technique that they feel is applicable and helpful to their software and organisation situation. Using several or all of the analytical techniques presented in this chapter will provide assistance to the software test manager, in planning for what-if scenarios, and in determining risk based on changes to their decision support framework model. Regardless of what technique is employed, there are possible conflicting interests between software test managers, software developers, the software development manager, other managers in the organisation, and relevant stakeholders. These techniques provide a more substantial basis upon which software testing decisions can be presented to help avoid conflicts, and to improve planning and risk management that can be provided to all stakeholders.

The DSF models shown in Figure 6.3 and Figure 6.4, and repeated for the convenience of the reader in Figure 6.5 and Figure 6.6, have been used for the discussion about how to analyse and interpret the DSF, and its resulting model, for the software to be tested. The following table provides a summary of the analytical techniques described and illustrated in this chapter.

Table 6.15: Summary of analytical techniques

Analytical Techniques	DSF Components Targeted	Used for
Cross Review	Entire model	Checking completeness of test planning
Risk Path Analysis	Element links and directional signed relationships influence weights	Test planning and risk management
Event Path Analysis	Element events occurring	Risk management
Fuzzy Cognitive Map	Influence weights element analysis	Risk management
Factor Contribution Analysis	Elements factor contributions	Risk management and test planning

- ||| -

CHAPTER 7. FURTHER PERSPECTIVES ON THE DECISION SUPPORT FRAMEWORK

The following diagram provides an outline and overview of the conceptual structure of this chapter.

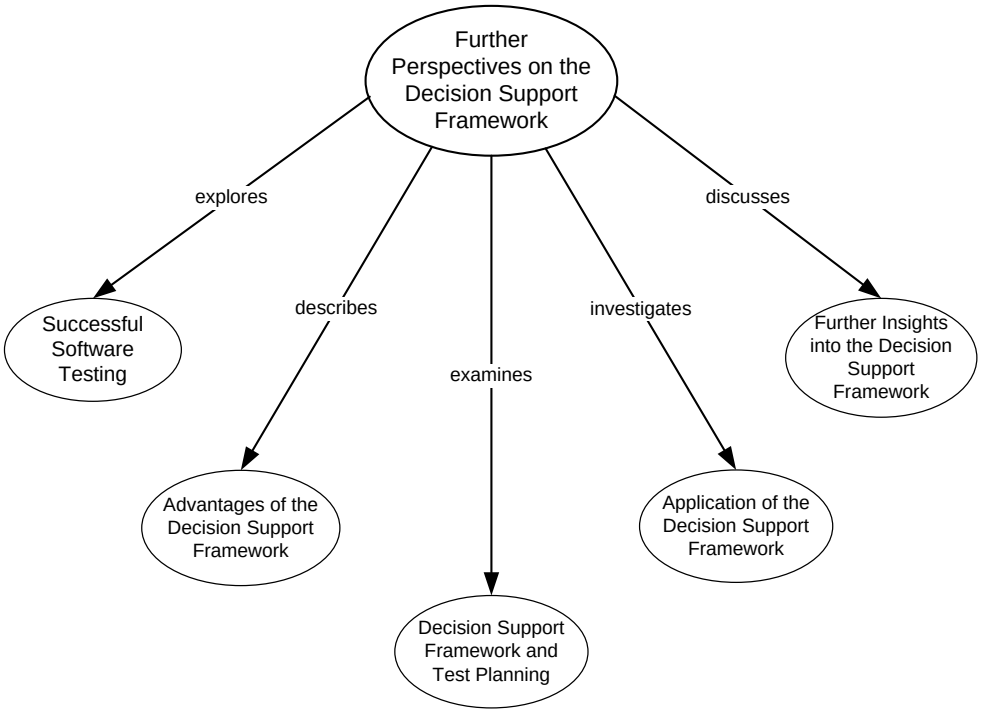


Figure 7.1: Conceptual structure of Chapter 7

7.1 Introduction

Chapter 6 illustrated how the decision support framework can be used by the software test manager. In contrast, Chapter 7 adopts a broader perspective, and provides a coherent view and a greater depth of understanding of the decision support framework and its use. This unified and increased understanding of the framework reinforces its usefulness as a tool for the software test manager. The goal of successful software testing is examined. The advantages of the DSF are stated and described. The relevance of the framework to the activity of test planning is explored. Different perspectives are used to look at how the software test manager applies the framework to their specific software testing situation. The treatment of the application of the DSF includes an in depth understanding of risk management as it relates to the DSF. This chapter concludes by discussing further insights into the DSF.

7.2 Goal of the Decision Support Framework: Successful Software Testing

The goal of the software testing decision support framework (DSF) is the analysis of *successful software testing*. But what is successful software testing? Previously, the goal of the DSF was identified and stated (see Chapter 5, Section 5.3), but not elaborated on. This section explores different perspectives that affect and are related to successful software testing, that the software test manager needs to be aware of. This strategy of exploring the different perspectives was adopted to clarify and describe the meaning of *successful software testing* in the context of the decision support framework.

The definition of successful software testing that has been developed through this research, and adopted for the decision support framework is as follows.

> ***Successful Software Testing****: is the result of a set of activities that produces a product with a minimum number of issues, on time, within a given budget, and of sufficient quality to meet the requirements and expectations of the customer.*

Successful software testing can be viewed as the result of using the decision support framework (DSF) as the basis to guide and help the user remember, and apply (wherever applicable) all elements, their factor contributions, and directional signed relationships to achieve the best outcome. There is no guarantee that the outcome of the goal or objective will be 100% met. But the DSF does provide a better basis for test planning, a better understanding of risk (that can lead to a highly reduced risk), and a better decision making basis to achieve the goal of successful software testing.

Clearly, the right product must be built to satisfy the customer. A customer perspective of successful software testing is a high quality product with the required functionality, which will operate in their environment, and will not fail. Successful software testing has an organisational context. Other roles in the organisation may impinge or relate to successful software testing. The software test manager has to understand these different organisational roles and ensure that successful testing of the software is consistent with the roles.

Depending on the organisation's structure the roles can include senior managers, operations manager, software development manager, and project managers. Senior managers view the day to day operations of the organisation with a strategic eye ("Senior management", 2011),

and will consider successful testing of the software product in a broader context. The operations manager is responsible for efficient and effective business operations ("Operations management", 2011), and will view successful software testing in terms of efficiency (using minimal resources such as finances, personnel, and time), and effectiveness (meeting the customer requirements).

The software development manager has to manage the priorities of multiple projects, make sure projects are completed, and plan for new projects (Bogue, 2005a). Successful software testing means that the software development manager will not have to readjust the priorities for the project, can help to ensure that the project is completed without any hindrance, and will not be distracted from planning for future projects. The project manager is responsible for planning, executing and closing the project, and has to manage the time, cost and quality dimensions of the project (Bogue, 2005b; "Project manager", 2011). The project manager will see successful software testing as building a product that matches the customer requirements, developing the product on time and within budget, and creating a product that does not present a risk.

7.3 Advantages of the Decision Support Framework

The Decision Support Framework (DSF) provides a sound and coherent basis upon which software testing uncertainties can be investigated in the test planning phase of the software testing life cycle. Software testing environments are full of incomplete and imprecise information, accompanied by various forms and sources of uncertainty (Dick & Kandel, 2005). The DSF forms a dynamic representation of the software testing project space. It can be used to drive or envisage test design, provide a guide to software test planning, provide decision support in the form of risk management assessment, help capture the knowledge of the test manager for the type of software to be tested, and support reports to management and other stakeholders about the activities related to the essentials of testing.

Using the DSF to support the software testing project has several advantages:

- firstly, it provides a template or pattern that the software test manager can use with any type of software that is planned to be tested (test design). When the software test manager reviews the software requirements, he/she can use the DSF as a template, a set of common testing features for software applications, to be considered when he/she approaches how they are going to design the tests for that type of software;

- secondly, the DSF serves as a guide for software test managers to identify those items that they need to consider when planning for software testing (guide to software test planning). The DSF is a guide that provides a set of elements and factors to be considered in the planning for successful software testing. These elements and factors are reminders to the software test manager of those issues they need to consider in test planning;
- thirdly, the DSF can be used for assessment of risk management issues (risk management). The test plan is developed, but before or after it is implemented the testing situation changes and the test plan does do not go 100% as planned. The DSF is used to analyse the risk of not achieving successful software testing by employing what-if analysis. Many what-if questions could be asked that need quick responses. For example, What if one of the resources becomes unavailable? What if the involvement of one of the resources is reduced to less than full time? What if there is a change in customer requirements? What if a test team member falls ill, or leaves part way during testing of a software application?;
- fourthly, the software test manager's knowledge and expertise is captured in their assessment of the influence weightings and factor contributions, which they assign to the DSF (knowledge of the software test manager). A record of the values of the influence weightings and factor contributions, for different types of software applications, has the potential to act as a resource that software test managers can use for future software projects where the DSF is applied; and
- finally, the DSF provides a foundation for software test project review from a test planning perspective, the basis for inputs to the reports for management and other stakeholders. Software test managers provide reports to management in accordance with what the organisation has established, which reflects what management wants to see and needs to know. When a test project is completed, the test manager writes a report beginning with the initial test plan and provides any evaluation based on changes that may have occurred before and during the testing. He/she outlines all issues not fixed and includes all the statistics about the testing process. The review is meant to help planning for future test projects, which involves schedules, budgets, issues, and so forth. Example reports are: change reports (effect of changing software requirements by developers or clients), resources reports (such as the effects of

competing resources, which happens when there are simultaneous multiple testing projects), and reports for risk management.

7.4 Test Planning and the Decision Support Framework

As discussed in Chapter 5, Section 5.4, all system development life cycles have a phase that deals with testing. The testing phase of the system development life cycle has a software testing life cycle of some sort (Kaner et al., 1999). The diagram below illustrates where the DSF fits into the software testing life cycle (STLC).

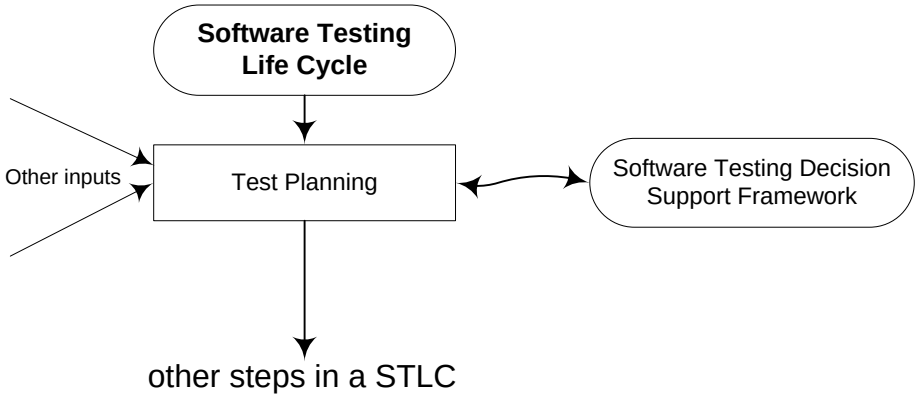


Figure 7.2: Decision support framework and the STLC

As part of test planning, the DSF activities are used by the software test manager to apply the information contained in the DSF to any type of software application that needs to be tested. Although the DSF provides a great portion of what is needed to assist in test planning there are still other inputs that are needed to fully complete the test planning process.

7.4.1 Details of Test Planning

Test planning was discussed at a high level in Chapter 3, Section 3.5. Test planning is an indispensable phase of the software testing life cycle. The discussion below provides more details about test planning.

The test plan is indeed the major output of test planning. But test planning builds on an essential set of inputs that involves many activities to identify, collect and massage the information necessary to test the specific software at hand. Test planning and preparation must happen *before* the test plan can be developed.

Test Planning (Ammann & Offutt, 2008; Copeland, 2004; IEEE, 2008; Mosley & Posey, 2002) includes the following.

- *Identifying staffing requirements*. This includes identifying how many people will be needed, the skills that are needed by those people, responsibilities of each person, and when and where each person will be needed.
- *Identifying resources needed*. This includes identifying the hardware, software, environment, and facilities that will be needed to conduct and report on the testing process and the results of testing.
- *Developing a testing schedule.* The schedule estimate can include: when and what test procedure is to be used, how long each series of tests will take, how long the entire testing effort will take, and how much regression testing time will be allocated.
- *Creating a test strategy and approach.* The test strategy includes what type of testing and how much of each type is to be done. Also included are the features to be tested and those features not to be tested, as well as prioritising software requirements for testing. Some testing strategies are given in Appendix A.
- *Assessing risk.* Assessing risk involves identifying the risks in the testing process if something does not go according to plan, and there is a potential adverse effect on achieving the software project goal. For example: risk in the schedule, resources, technical approach or for going into production (IEEE, 2008). Risk is assessed and the potential impact of each risk is specified, and then contingency plan(s) for avoiding, or mitigating the risk are drawn up (IEEE, 2008).
- *Budget estimate.* The budget estimate takes into account staffing, resources, testing schedule, and regression testing etc...
- *Management of testing.* To ensure that all software requirements specifications, design characteristics, and the proposed architecture has been considered, some type of management or project plan is created to manage the testing process. The plan could include for example, how the cross checking, for completeness and consistency between the software requirements specifications, design characteristics and the architecture is going to be done.
- *Other needs.* This includes keeping in mind how bug management is to be done, what documentation is to be used to support the testing (referenced in the test plan), what documentation will be produced as a result of the testing, security considerations, and how software issues are going to be handled.

The decision support framework (DSF) supports the many tasks found in test planning that were just described. The next section illustrates visually how the DSF relates to test planning.

7.4.2 How the Decision Support Framework Relates to Test Planning

Figure 7.3 depicts the essential components of test planning and lists the inputs to, and the outputs of test planning. Figure 7.3 is built on selected aspects of test planning discussed in several sources (Ammann & Offutt, 2008; Copeland, 2004; IEEE, 2008; Mosley & Posey, 2002)

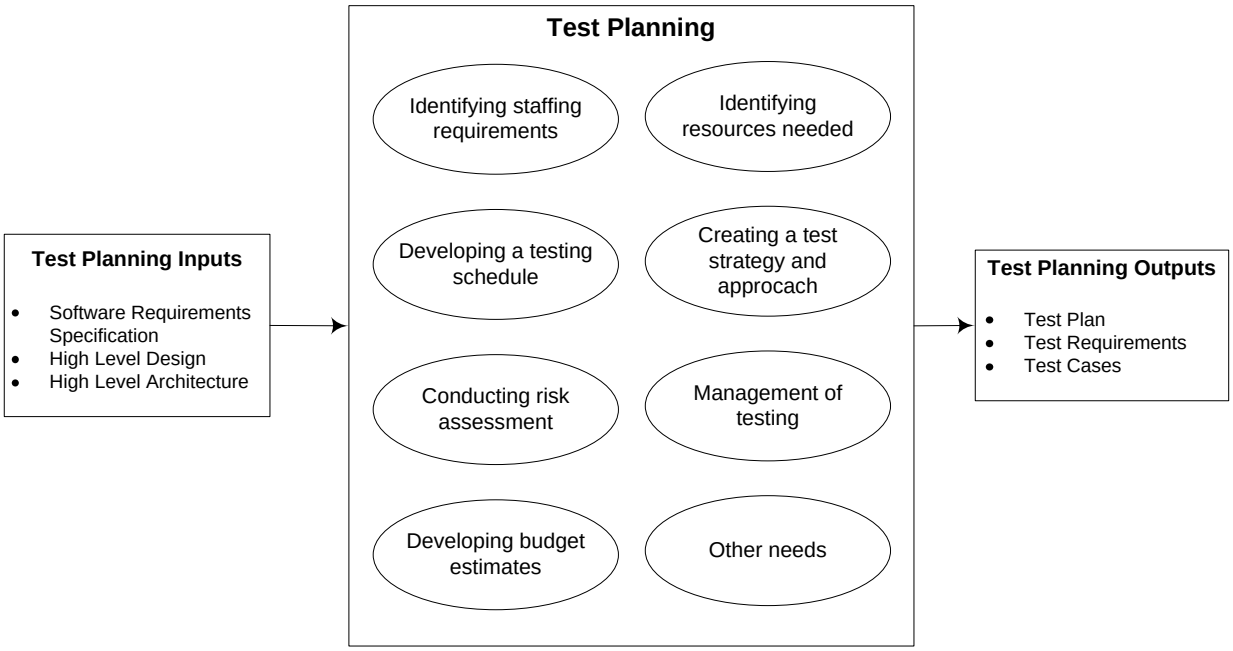


Figure 7.3: Test planning

The decision support framework (DSF) was developed to help software test managers as part of their test planning activity for successful software testing. The reason for developing the DSF masks a claim, expressed here as a question: How does the decision support framework relate to test planning? The following diagram shows how the DSF relates to test planning.

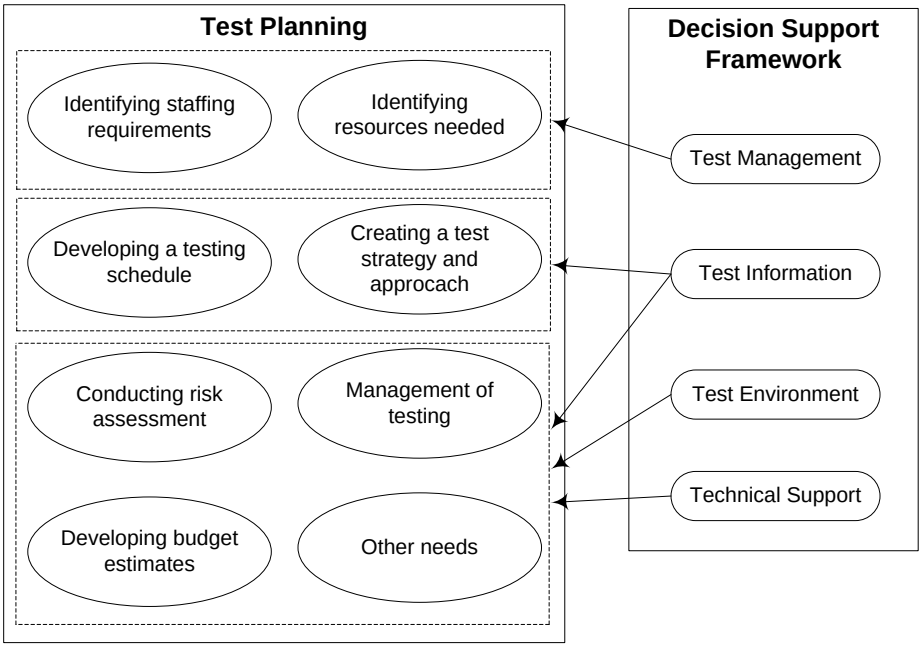


Figure 7.4: Mapping of decision support framework to test planning

Figure 7.3 shows how the elements of the DSF map to the components of test planning. It is instructive to recall the factors that comprise each element of the DSF. Knowledge of the factor names should provide a better understanding of how the DSF elements relate to test planning. The element factors were identified in Chapter 5, Section 5.3.6, and described in Chapter 5, Section 5.3.8 (some readers may need to re-read the factor descriptions to facilitate their grasp of how the DSF elements relate to test planning). The factors that make up each element are: Test management (Test manager support, Appropriate test resources, Potential for automated testing and Organising test staff), Test information (Test requirements, Test details, Software needs and test automation, Test strategy and Test design), Test environment (Test environment issues, Test support and Test data requirements) and Technical support (Support for test infrastructure, End-user environment, Information for test environment and Other support items).

7.5 Application of the Decision Support Framework

This section provides a holistic perspective, and a detailed description of the application of the DSF by the software test manager. The description demonstrates the flexibility of the DSF and how it useful for the practical conditions that a software test manager will encounter.

So far, application of the decision support framework (DSF) by the software test manager has been considered separately, according to the broad two step procedure to apply the DSF, which was described graphically in Figure 5.12 and textually in Chapter 5, Section 5.6.

Chapter 5 demonstrated the first step in the application of the DSF, which involved the software test manager assigning input to the DSF, and modelling their particular software testing situation. Chapter 6 examined the second step in the application of the DSF, where the software test manager evaluates the DSF model produced in Chapter 5.

Figure 7.4 illustrates the steps in the DSF used by the software test manager, and emphasises that the DSF ties in with the test planning phase of the software testing life cycle. This figure is an illustration to promote understanding of the section, and indicate a context for the material that is discussed.

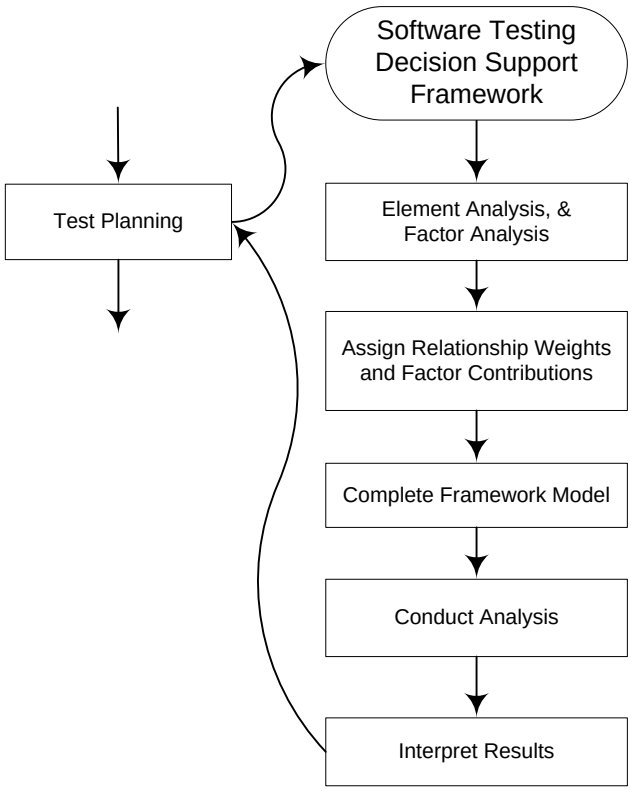


Figure 7.5: Steps in the DSF used by software test manager

7.5.1 Practical Overview and the Role of Risk Management

One of the tasks in test planning is risk management assessment, especially as it pertains to achieving successful software testing. As shown in Figure 7.3 one of the outputs from test planning is the test plan. At this stage in our discussion, the initial DSF model has been used in test planning, like the model shown in Figure 5.10. This DSF model is referred to as DSF Initial (the initial completed DSF model). The evaluation of DSF Initial is completed, becoming a major tool for test planning and the resulting test plan.

However, before testing begins or while it is in progress, situations occur that affect the basis for test planning, and the results acquired in test planning. The corollary for the framework is change to the factor contributions. Once these changes have been identified and DSF Initial adjusted accordingly, a new DSF model results: DSF Modified. The test manager has created a new model, and proceeds with further evaluation, based on DSF Modified.

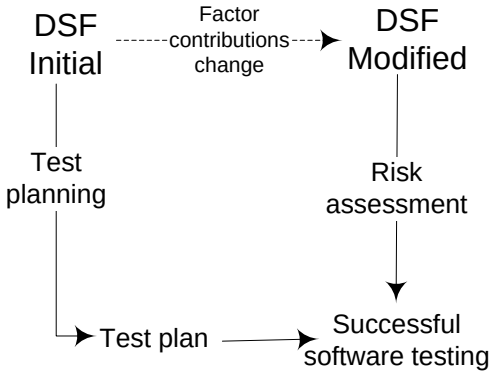


Figure 7.6: Decision support framework assessment basis

As noted above, the completed DSF Modified model needs to be evaluated. The software test manager selects one or more analytical techniques that they want to use, depending on the budget and resources available. The analytical techniques include:

- static techniques
  - cross review;
  - risk path analysis; and
  - event path analysis.
- dynamic techniques
  - fuzzy cognitive map (FCM) analysis; and
  - factor contribution analysis (FCA)

Assessment of risk is calculated, based on the DSF Modified model that results from the changed testing situation, with reporting to management. Furthermore, DSF Initial can be compared with DSF Modified for reporting purposes, as well as evaluation of the overall testing project for the type of software tested.

The preceding example illustrates clearly the role of risk management in the application of the DSF. Nonetheless, this example is built on an implicit assumption – the software testing situation only changes once. In the real world of software testing a single change to the testing situation is very unlikely. In software testing practice the length of testing is fluid:

testing length could be a couple of weeks or up to three months (which magnifies the potential for change). The number of changes to the testing situation could be many.

Continuing with our example, DSF Modified could have changes against it and would have to be adjusted. Thus, the software test manager needs a notation to distinguish between DSF Modified and models created to reflect any subsequent changes to the testing situation. A possible generalised notation is DSF Modified(i), where i = 1 to n. Using this notation, the model DSF Modified becomes DSF Modified(1). Where DSF Modified(1) has changes against it, and has to be adjusted, DSF Modified(1) becomes DSF Modified(2), and so on to accommodate any further changes.

7.5.2 Application Details

This section provides textual details about how the software test manager uses the decision support framework when they are testing a type of software.

The software test manager would use the DSF to model their particular software testing situation. This would involve using the DSF:

- to assist in identifying and understanding the relevant issues in the test planning needed for successful software testing, regardless of the type of software to be tested; and

  The DSF is used for test planning as follows. The software test manager reviews the software/system requirements and software/system design documents. Based on this review the software test manager creates a test plan, from which the test procedures will be developed. The software test manager brings up the DSF and focuses their attention on the elements and factors of the DSF. Next the software test manager goes through each element and its factors, to decide the degree to which each is relevant to the type of software to be tested. This information is used to ensure all applicable areas for software testing have been considered. The software test manager can use this information to help develop their test plan.

- to analyse the factors of each element that influences successful software testing;

  Analysis of factors of the DSF proceeds as follows. The software test manager analyses the factors of each element and assigns a contribution percentage to each factor. The software test manager provides estimates for each element's influence weight on achieving another element (as applicable), and on the goal of successful software testing (as applicable). These influence weights are assigned by the software test manager to the relationships of the DSF. The software test manager uses all the preceding details about factor contribution and influence weight percentages to complete the DSF model. At this point the DSF is ready to employ one or more of the analytical techniques.

The software test manager would evaluate the software to be tested. This would involve:

- conducting evaluation of the DSF model of the software testing situation, using analytical techniques to assist in test planning for, and risk management of the software to be tested; and

  Evaluation of the DSF model proceeds as follows. The software test manager can select which analytical techniques that they want to use, depending on the budget and resources available. The analytical techniques are static and dynamic. The static techniques include cross review, risk path analysis and event path analysis. The dynamic techniques include fuzzy cognitive map analysis and factor contribution analysis.

- interpreting the results of the evaluation of the DSF model;

  The evaluation results are interpreted as follows. Depending on the analytical technique selected, the software test manager has to interpret the results of the evaluation of the DSF model in terms of, how the analytical technique assists test planning, and how the analytical technique helps to improve understanding of the risk management issues, for the type of software to be tested, within the constraints of the organisation.

7.6 Additional Insights into the Decision Support Framework

This section offers a fresh perspective of some aspects of the DSF that have been discussed earlier, examines the foundation of the DSF, investigates how the DSF embodies the characteristics of decision support, explores how the DSF provides a basis for risk management, and presents a unified view of the application and advantages of the DSF.

7.6.1 Some Aspects of the Decision Support Framework Revisited

The decision support framework (DSF) has a single person scope, and is intended for a software test manager. It has been designed for the software test manager or someone with equivalent test managerial skills and experience (for example, the test lead or senior tester). The purpose of the DSF is twofold, in support of the goal of successful software testing, for any type of software application or software system. The DSF was developed to assist the software test manager in test planning, and to help improve their understanding of risk management issues, and the consequences of those issues. The DSF is a tool for the stand alone analysis of risk management, and this capability underlies its capacity for decision support. Mention of the DSF as a tool for software test managers has been made before; and the term tool in the context of the DSF needs to be clarified. The DSF tool is the computer software that captures the textual and graphical representations of the DSF, and provides the basic enabling technology to permit the software test manager to employ the analytical techniques which accompany the DSF.

Application of the decision support framework underscores access to and manipulation of the DSF model. Thus, the framework can be viewed via the model it creates. The framework is not data intensive. The software test manager supplies simple parameters (influence weighting and factor contribution percentages), that is, information about the specific software testing situation, and these parameters plug into the high level requirements of the DSF. This shifts the focus of the decision support framework from the general to the particular, and the interrelated elements captured in the framework are used to create a model, which represents the specific software testing situation that confronts the software test manager. The software test manager can use the elementary functionality of the framework (the static and dynamic analytical techniques), to manipulate the representation of the software testing situation provided by the DSF model, and analyse the software testing situation. Specifically, the representation is manipulated by the software test manager to

assess the risk management issues, and their consequences, and improve the software test manager's understanding of the risks inherent in the software testing situation.

7.6.2 Foundation of the Decision Support Framework

The decision support framework has a firm foundation, derived from three areas.

- *Conceptual basis of the DSF*. The DSF has a sound conceptual basis for its components (elements, the factors of the elements and relationships amongst the elements), which provide an integrated structure that is a reasonable representation of the software testing domain. The elements, the factors of the elements and relationships amongst the elements of the framework were carefully identified and defined using an eclectic approach, based on information that was synthesised, moulded and integrated, drawn from several diverse sources. The information sources were literature review, information derived from industry, informal discussions and testing experience.
- *Relationship of the DSF to test planning*. The DSF was designed to assist the software test manager in their planning for successful software testing. Underpinning this purpose is the close relationship of the DSF to the test planning phase of the software testing life cycle. Section 7.4.2 of this chapter demonstrated clearly how the elements of the DSF map to a large part of the test planning components, and how the DSF elements are semantically congruent with different aspects of the test planning components.
- *Literature support for characteristics of the DSF model*. The DSF model created during the application of the DSF is a representational model, the characteristics of which have been given prior attention in the literature. Alter (1977) developed a taxonomy to classify decision support systems (DSSs) according to their generic operations (Alter, 1977). One type of DSS Alter (1977) identified was the representational model, which estimates the consequences of actions (Alter, 1977). Some common features of this model are: used for planning tasks, aimed at managerial decision makers, and used for understanding (Alter, 1977). Although this type of DSS and the DSF are different in their complexity and functionality, their operations are both model driven. Application of the DSF and the type of DSS under discussion is predicated on models that represent the software testing situation, and problem space, respectively, coupled with the use of the functionality built into these different forms of decision support. The operations of the DSF model correspond

closely to the common features of the representational model described above: the DSF model assists test *planning*, targets the software test *manager* (decision maker), and improves *understanding* of risk management (by exploring the *consequences* of risk).

7.6.3 How the DSF Addresses the Characteristics of Decision Support

The research has described what the DSF is, how to use it, and some of its features. But an underlying and important question has thus far been overlooked: Is the decision support framework consistent with the characteristics of decision support? What makes something decision support was discussed in Chapter 2, Section 2.4.2, and a set of general characteristics was presented. These characteristics have to be satisfied before something can be considered as supporting decision making. The decision support framework is now largely deconstructed, and discussed in terms of the decision support characteristics that were previously identified. The decision support framework is a computerised tool (computerised means) that:

- supports the decision making of test managers (decision makers) but does not replace their judgement, which is based on their experience in and knowledge of the software testing domain (supports decision makers);
- helps to develop decisions in the complex test environment, which has some incomplete and imprecise information, accompanied by various forms and sources of uncertainty (Dick & Kandel, 2005), typical characteristics of unstructured situations (formulates decisions in unstructured situations);
- is focused on support for the decision making activities involved in the test planning phase of the software testing life cycle, an essential step in the testing process, where timing is crucial because test planning precedes development of the test plan (decision value at the time it is needed);
- helps improve the accuracy, timeliness and quality (in terms of process or outcomes and their consequences) of the decisions for test planning and risk management, using decision analysis to structure the testing space for the type of software to be tested, and provide information for more informed decision making (improve effectiveness of decisions); and
- offers the ability for software test managers to modify the factor contributions and adjust the framework, to allow for test situations that evolve before or during testing, and re-assess risk, based on the framework model that results from the changed testing situation (accommodate situations where information can change).

The discussion above clearly establishes that the decision support framework addresses the characteristics of decision support.

7.6.4 How the DSF Provides a Basis for Risk Management

The abstract claimed that the decision support framework establishes a basis for risk management in software testing. This claim has remained unexplored thus far: it will now be discussed and explained. The claim immediately raises a question: How does the framework establish a basis for risk management? Another question quickly follows: What aspect of risk management is being referred to?

The word basis is used here in the sense of underlying support or foundation. The DSF establishes a basis for risk management from two perspectives: a foundation and the use of that foundation by the software test manager. We will deal with the foundation perspective first. The DSF captures the essential characteristics from the dynamic software testing space that are required to achieve the goal of successful software testing. The DSF expresses these characteristics in a structured, integrated and generalised framework, which is accompanied by several analytical techniques. A reminder: the framework consists of a set of elements, element factors, relationships and high level requirements – for the influence weightings and factor contributions (see Chapter 5, Section 5.3 for details). The software test manager uses this foundation (the generalised DSF) in a two-step application procedure of the DSF for risk management. The software test manager provides input to the framework to create a representational model of their specific software testing situation. The model is then used by the software test manager to analyse the risk consequences of the frequently evolving conditions in that software testing situation (see Section 7.5).

Boehm (1991) describes the different steps in software risk management (Boehm, 1991). These steps are applicable to software testing. A primary step in risk management is risk assessment, which involves risk identification, risk analysis, and risk prioritisation (Boehm, 1991). The DSF is used specifically for risk analysis, and is used by the software test manager as a decision analysis tool to analyse risk for a particular software testing situation. The results from the risk analysis are used not only for decision making in the current software testing project, but can potentially be used for risk planning in future software testing projects. Risk analysis examines the factors that may jeopardise the success of a project, or achieving a goal ("Risk analysis (business)", 2011). The DSF achieves risk analysis by the

application of its static and dynamic analytical techniques to the DSF model. The analytical techniques focus on the element events, element links, element factor contributions and influence weightings of the framework (see Table 6.15). Decision analysis refers to analysis of important decisions, and includes evaluating the situation in which the decision is made, capturing the essence of that situation, "solving" the decision problem, and providing insight to the decision makers and implementers (The Stanford Decisions and Ethics Center, 2009).

7.6.5 Application and Advantages of the Decision Support Framework

In this chapter the application of the DSF and the advantages of the DSF have been addressed separately (see Section 7.5 and Section 7.3 respectively of this chapter). The following diagram provides a high level integrated view of the application and advantages that come with the DSF.

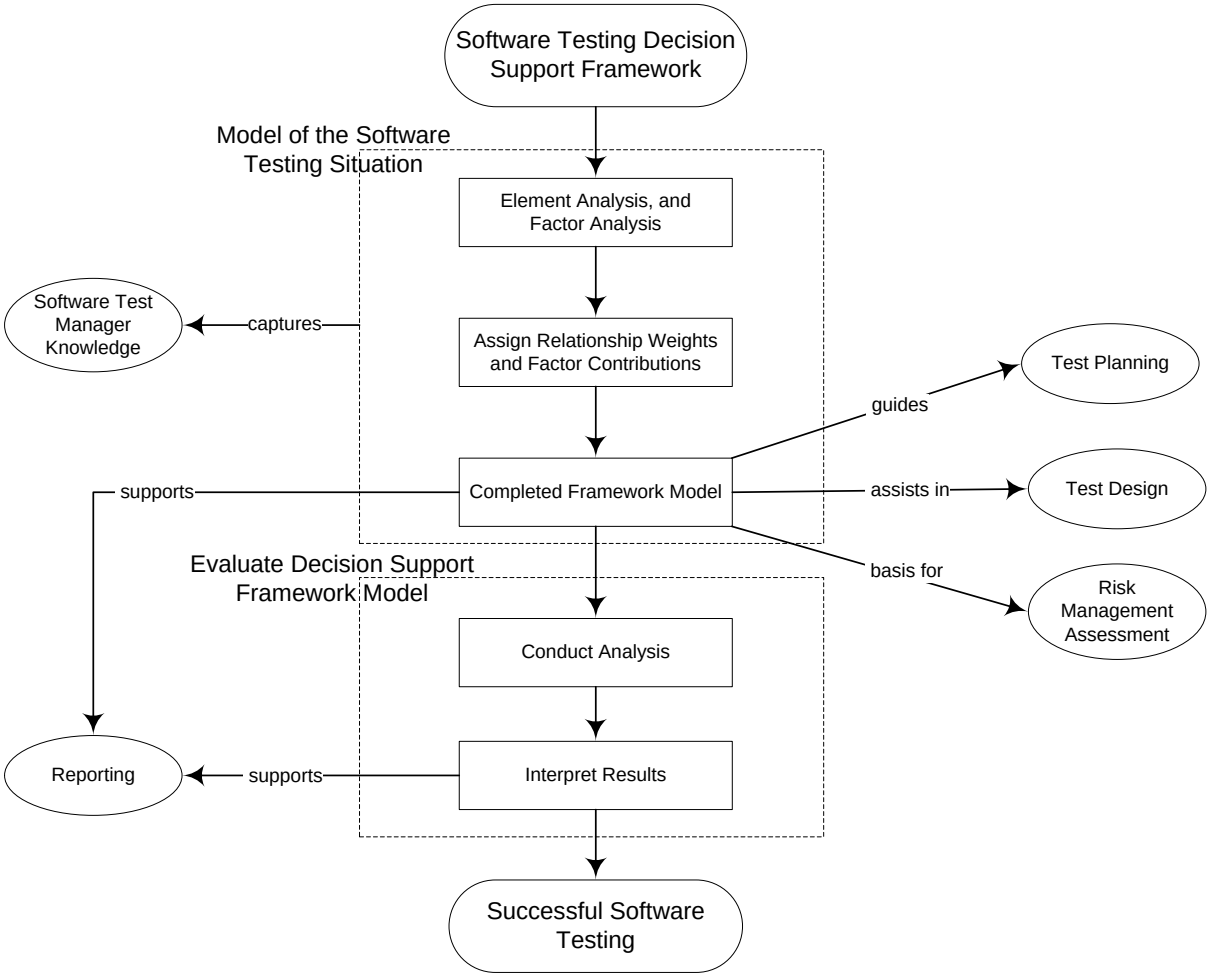


Figure 7.7: Consolidated overview: application and advantages of the DSF

7.7 Summary

This chapter has provided more insight into the DSF through the different perspectives that have been introduced. The seemingly straightforward notion of the goal of successful software testing was scrutinised, and shown to involve more than one view about what successful software testing means. The advantages that the DSF brings were detailed and briefly include engaging in test design, a guide to test planning, decision support for risk management (which involves support for what-if analysis), capture of software test manager knowledge, and the basis for reports to management and other stakeholders. How the DSF maps logically to a large part of the software test manager activity of test planning, was described and discussed. The application of the DSF was addressed from a holistic and practical stance, with the intent to combine and explain in greater depth the aspects of the DSF application dealt with separately, and at a broad level, in Chapter 5 and Chapter 6. Finally, the different insights that were previously implicit in the DSF were made explicit and explored in detail.

- ||| -

CHAPTER 8. CONCLUSIONS AND FUTURE RESEARCH

The following diagram provides an outline and overview of the conceptual structure of this chapter.

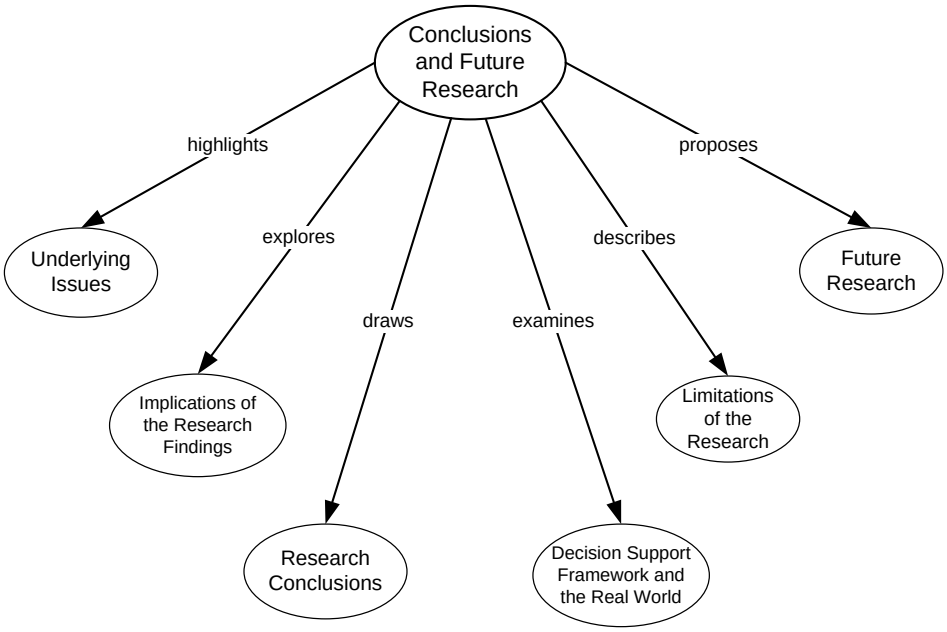


Figure 8.1: Conceptual structure of Chapter 8

8.1 Introduction

This chapter reviews and sums up the information about this research and the development of the decision support framework (DSF). The chapter starts by identifying and highlighting some underlying issues, which are based on a background review and research considerations. These issues are followed by an exploration of the implications of the research findings. Then the research findings are analysed and used as the basis to draw several conclusions. Issues surrounding the use of the decision support framework by software test managers in the real world are examined. Next the limitations of the research are investigated in detail, and suggestions for future research are discussed. In closing, the final remarks provide an overall perspective to complete this research.

8.2 Underlying Issues

This section highlights a set of issues that underlie this concluding chapter. A background review and research considerations encapsulate the different issues that are seen as important. The issues that are exposed establish a platform to promote understanding for the remainder of the chapter.

8.2.1 Background Review

Chapter 2 and 3 provide the basis upon which this research is founded. Chapter 4 extends the background information into the conceptual structure that underlies the development of the decision support framework.

Real world software testing can be complex and complicated. Research into software testing needs to be directed at a specific area of software testing. The research area needs to be well defined. As part of the definition, the research needs to have its terminology and perspective defined to promote an understanding for all of the concepts and terms used. The terminology surrounding the research must have explicit definitions. Those definitions clarify and confirm the author's research perspective, as well as the use of related terms within that research. Moreover, this helps to avoid circular definitions, and thereby facilitates the reader's understanding of the research and everything within the research boundary. This makes a more solid foundation upon which later discussions are held, and avoids repetitively defining the perspective that the author is taking.

The literature review has shown that there is little research into decision support frameworks for software testing management. More so, there have been few publications that are directed at assisting the software test manager in their test planning and risk management role.

The literature review has discussed what is meant by decision support to show that the development and application of the DSF fits into that environment. An important element of the literature review shows that the role of the software test managers is very encompassing, requires many skills, and that test managers need to apply their experience to software testing.

One important part of a software test manager's decision making role is the planning for successful software testing. But as in all such projects the factors that contribute to success do not always get fulfilled at the intended or planned rate. The software test manager needs to plan for and conduct risk management of successful software testing, before testing and while testing is continuing. Risk management is an area that is an indispensable part of the software test manager's decision making role.

Chapter 2 and Chapter 4 have shown that frameworks are not well understood and often used to mean a variety of things in different contexts. Furthermore, the term framework has been

loosely applied to research that addresses situations of related issues that do not constitute a framework, as defined for this research. Chapter 4 develops the understanding of what a framework is and the concepts of their use. This understanding provides a basis for a sound grasp of the concepts and the foundation upon which the decision support framework has been developed.

8.2.2 Research Considerations

The goal of testing is to uncover and report on issues. It is not possible to claim that any software product is issue free. Exhaustive testing is not possible. Even after spending thousands of hours in testing, there will be one or more combinations for execution of the software, which were not or could not be tested. One condition might surface in the customer's operational environment that was not known at the time of software development and testing of the software. Another condition may show itself due to unfamiliar hardware, specialised network configurations, or security settings that are not known to the software developers.

It is true for nearly all testing projects that time and resources are limited. But how many times issues are missed because of unavailability of resources, inadequate test planning, or unforeseen changes in resources during testing, is not known. Furthermore, assessing the risk of unavailable and changing resources has yet to be adequately researched or reported on.

Availability of time and resources is important, but it is more important to have solid test planning for when situations do not work according to the initial plan. Planning for successful software testing enables the test manager to assess the risks involved before and during testing, as well as planning for testing evaluation after testing. The test manager needs to employ a broad focus on software testing, while keeping in mind and planning for the risks associated with that testing, but in the context of the constraints imposed by their organisation.

The DSF provides a basis and perspective of the software to be tested, which gives the user of the DSF a variety of options in their analysis. This includes both static perspectives and dynamic perspectives. A set of analytical techniques that a software test manager can employ has been described and illustrated.

Using the DSF with the static perspective the software test manager can make a quick analysis. Often such an analysis is needed when schedules are tight. However, the static perspective does not provide the in-depth analysis that is often needed.

The dynamic perspective can be done using fuzzy cognitive map analysis or factor contribution analysis. Both of these analytical techniques offer in-depth analysis that can be used to assist the software test manager to plan for the risks associated with software testing, based on the obligatory constraints existing in their organisation.

8.3 Implications of the Research Findings

The research findings allow the research conclusions to be determined, provide answers to the research questions, and establish and describe the research contribution. Section 8.4 presents the research conclusions. Details of the answers to the research questions and a description of the research contribution are provided in this section.

8.3.1 Addressing the Research Questions

Chapter 1 introduced the five research questions to be addressed in this thesis. The research question: **How has decision support been used for software testing?**, has been addressed by the literature review. The literature review (Chapter 2) showed that decision support for software testing is limited. It was revealed that the approaches to decision support involve some type of test metric, or models developed with Bayesian networks (an artificial intelligence technique). Various aspects of software testing were considered by these approaches, and these aspects included test planning, test strategy design, the value of single test cases, test case failures, planning of test cases and writing of test cases.

The literature review revealed little to no evidence of decision support frameworks in software testing, particularly decision support frameworks designed for software test managers.

The research question: **What are the reasons for the areas of neglect in decision support for software testing?**, has been addressed by the literature review. The reasons identified were the testing complexities presented by the numerous types of software, the diverse categories applied to the software types from various perspectives, and the potentially large range of different requirements necessary to adequately test a particular type of software.

The research question: **Can a decision support framework be designed and developed to address the areas of neglect in decision support for software testing?**, was answered by Chapters 2, 3 and 5. Chapters 2 and 3, in combination, identified an area of neglect in decision support for software testing: the almost total absence of research on the development or use of decision support frameworks. This observation provided a clear focus for the decision support framework developed by this research. The literature review (Chapter 2) demonstrated that decision support for the planning of risk management before, during and after software testing is scarce. The research foundation (Chapter 3) highlighted the importance of risk management in software testing, and the essential role of planning as one of the software test manager's activities. Chapter 5 addressed the development and part of the application of the decision support framework. A generic decision support structure was built, which was the basis for the design of the decision support framework. The generic decision support structure was illustrated graphically, and accompanied by detailed descriptions of all of its components. The multi-step, iterative development procedure for the decision support framework was described. A comprehensive outline was provided of the framework components (elements, relationships and each element's factors). The architecture of the framework components was illustrated graphically. Illustrative software test manager inputs (influence weighting and factor contribution percentages) were assigned to the framework. This demonstrated how the software test manager can use the decision support framework to model their particular software testing situation.

The research question: **How can the decision support framework be applied, evaluated and interpreted by software test managers?**, was addressed by Chapter 5 (Decision Support Framework for Software Test Managers) and Chapter 6 (Using the Decision Support Framework). Chapter 5 described graphically and textually the two-step procedure to apply the framework. Chapter 5 introduced the first step in the framework application procedure. Software test managers analyse the framework elements and the factors of those elements to determine influence weighting percentages, and factor contribution percentages. These values are assigned to the framework by the software test manager, which transforms the framework to a model. The software test manager uses this decision support framework model to model their specific software testing situation. Chapter 6 described static and dynamic analytical techniques that the software test manager can use to evaluate and interpret the framework model, and establish a basis for analysing and interpreting the risk associated with testing a particular type of software. The static analytical techniques included cross review, risk path

analysis and event path analysis. The dynamic analytical techniques included fuzzy cognitive map analysis and factor contribution analysis.

The research question: **From what perspectives can the decision support framework be evaluated and interpreted by software test managers, and how can this be illustrated?**, was answered by Chapter 6. The way in which risk path analysis works was illustrated for two scenarios of different types of software. The results for risk path analysis for the two software scenarios were compared. The comparative approach applied the risk categories developed for this analytical technique to the results. Use of event path analysis was demonstrated for two software scenarios, and the results were also compared. Fuzzy cognitive map analysis was used to test the influence of the activation level of individual elements on the framework goal, through their interactions with other elements in the framework. This was achieved with a multi-step procedure. The influence of each element on the framework goal was tested and illustrated with worked examples for a type of software. Factor contribution analysis was used to determine the influence on the goal when an element's total factor contribution is less than 100%. The influence of each element on the framework goal, when that element's total factor contribution is less than 100%, was illustrated with worked examples for a type of software. Risk categories developed for this analytical technique were applied to the results. Scenarios with the factor contribution of several elements below 100% were examined, and illustrated with worked examples, including the application of risk categories.

8.3.2 Research Contribution

The major contribution of this research is the construction of a decision support framework. The research contribution is in two parts: the design and development of the decision support framework, and the application of the decision support framework.

The decision support framework was developed to assist the software test manager in the test planning and risk management required to help achieve the goal of successful software testing. This approach enables the test manager to analyse and interpret the risk associated with testing a particular type of software, before, during and after testing (for the evaluation activities). Moreover, the framework is sufficiently flexible to accommodate software from its initial state and when things change.

That design of the framework is based on a generic decision support structure, which allows a decision support framework to be built for any problem domain. The decision support framework was designed to:

- support the complex decision making of software test managers when they test software;
- be general, so that it is generally applicable, and suitable for different types of software and organisational settings; and
- have a simple topology, and thus a high level representation, which will not present the software test manager with too many concepts and relationships to get their head around, understand, and ultimately apply to their software testing situation.

The decision support framework consists of several interrelated and interactive components. These components are the elements, relationships, and the factors for each element.

The decision support framework has a clear context. The framework fits into the planning phase of the software testing life cycle (STLC). The STLC is located in the testing phase of the system development life cycle (SDLC), irrespective of the type of SDLC that is adopted for software development.

The application of the decision support framework is a two-step procedure. The software test manager assigns influence weighting percentages to the relationships of the framework, and factor contribution percentages to the factors of each element of the framework. Influence weights are determined by the software test manager based on their knowledge of the type of software to be tested, underpinned by their software testing experience. Factor contributions are determined by the software test manager based on analysis of the factors for each element of the framework.

The preceding inputs transform the framework, and allow the software test manager to model their testing situation. Static and dynamic analytical techniques enable the software test manager to evaluate the decision support framework model, which provides a basis for test planning and risk management. The static analytical techniques consist of cross review, risk path analysis and event path analysis. The dynamic analytical techniques consist of fuzzy cognitive map analysis and factor contribution analysis. The software test manager only needs a basic knowledge and understanding of mathematics to use these analytical techniques.

The decision support framework encapsulates several advantages. The decision support framework:

- provides a template that the software test manager can apply to any type of software that is planned to be tested (drives test design);
- serves as a guide for software test managers of those issues they need to consider when planning for software testings (guide to software test planning);
- can be used for assessment of risk management issues. When situations do not go 100% as planned, the decision support framework can be used to analyse the risk of not achieving successful software testing (decision support for risk management);
- captures the software test manager's knowledge and expertise in the assessment of the influence weightings and factor contributions (captures software test manager knowledge); and
- provides a basis for software test project review, and the review results are used for management reporting. Furthermore, the decision support framework can be used in comparing planned estimates against actual results, to assist in management and stakeholder reporting (supports reporting).

8.4 Research Conclusions

The literature review and research foundation have clearly demonstrated the complex and exacting role of the software test manager. The conclusion that follows is that any assistance that can be provided to test managers will not only make their job better, but could assist them in improving the planning of software testing and the analysis involved in that planning.

The design and development of the decision support framework (DSF) has been shown to provide a platform that software test managers can use for risk management. Various techniques have been presented and discussed. These techniques and methods provide the user of the DSF with different evaluative perspectives that they can use in their planning for successful software testing, and risk management when something does not go according to plan.

The design, development and evaluative use and interpretation of the DSF suggest some practical benefits for the software test manager. The DSF is a highly useable tool that has been developed to assist the software test manger in their planning and risk management of the type of software that is to be tested, and is being tested. The DSF tool has several

practical advantages: its conceptual structure is simple to understand, it can be used as an informal or formal approach, it requires only a basic understanding of mathematics, and it is quick and easy to apply.

8.5 The Decision Support Framework and the Real World

This section considers the DSF from a real world perspective. This perspective has two distinct parts. First: how to demonstrate the ability of the DSF to be applied to testing a software program in real world scenarios. Second: the practical areas to consider when the DSF is used in the real world.

8.5.1 How to Show that the DSF can be Applied in the Real World

Chapter 6 analysed the use of the DSF to test software, from static and dynamic perspectives, for two software scenarios. This analysis demonstrates general application of the framework, based on software scenarios designed for the analysis. However, to demonstrate the ability of the DSF to be applied, or validated, in real world software scenarios, requires empirical research. Such empirical research has to be based on a suitable research method, such as the case study. The case study method is a qualitative form of inquiry that focuses on a specific situation, and emphasises detailed, real life, contextual analysis (Yin, 2003). It has been pointed out previously that real world, concrete applications of the DSF to organisations in an industry environment, are outside the boundary of this research (see Chapter 6, Section 6.4.3.1). Notwithstanding this research boundary exclusion, the possibility of implementing case studies was investigated.

Specifically, the inclusion and application of organisation based case studies was explored. Several private and government organisations were contacted. Both government and private industry felt that the concepts presented in the DSF would be useful to their software testing manager. However, both government and private industry organisations provided several reasons for not being able to trial the DSF. These reasons included statements relating to: issues of time and resource constraints, security issues, organisation confidentiality, competitive and market privacy, intellectual property constraints, and client/customer confidentiality concerns.

8.5.2 Practical Issues in Real World Use of the DSF

Although the DSF is easy to use and understand, there are some areas that need to be considered when the framework is used in the real world. First, the experience and

background in testing of the software test manager needs to be considered. The DSF relies on the test manager to accurately, and to the best of their knowledge and experience, define and assign the influence weightings on relationships.

Next, the DSF requires the software test manager to understand the purpose and structure of the DSF elements, and factors of those elements. This will most likely require the software test managers to read over the information about elements and factors before they do their influence weightings.

Other documentation needs to be in place before the software test manager can begin test planning. For example, a complete software specification requirements document needs to be done. If the specification is not done or is constantly changing, it is very difficult, if not impossible, for the software test manager to develop any understanding of what the test requirements are, and what they mean. Therefore, the software test manager cannot determine what to test, or the degree of what they are to test. It is also difficult for the test manager to write a good test plan, with appropriate schedules and different options, if the requirements are not well defined.

A detailed design document should also be in place. This design document is needed, along with the software specification requirements document, to be able to create the test cases to ensure testing adheres to the intended design. This design document also provides a basis for describing issues that are identified during the testing process.

The development plan includes a schedule for the software development and milestones within the development. This schedule is needed by the software test manager so that he/she can begin their test planning. It provides a foundation upon which the software test manager can begin to plan for their resources and needs.

One other document that is needed is the test environment document. This document provides information to the software test manager about issues that are related to, and affected by the software being tested, but not issues related to the software itself. These test environment issues might include the hardware, network, source and destination of data, and system software.

8.6 Limitations of the Research

A limitation of the decision support framework is a potential inability to capture all the important elements from the software testing domain. Thus, the framework may be subject to external influences from elements potentially overlooked in the software testing domain. This is partly due to rapid changes in technologies and types of software to be tested. However, this limitation has been largely attenuated by the way the framework was designed and developed.

The framework has an integrated design, and integrative flaws identified during development helped reveal elements that had been poorly identified or omitted. The research, which exposed the ideas and issues used to identify and define the framework elements, is broadly based and extensive. This research base included literature review, information sourced from industry, informal discussions, and testing experience. Thus, the likelihood of accidentally excluding important elements was significantly diminished. Analysis of the elements to identify a set of factors is a strategy to ensure that issues consistent with the element being analysed were not disregarded. This element analysis was supplemented by suitable factor descriptions, which ensured that issues essential to each factor, were not left out either. Finally, the iterative approach to development of the framework provided the opportunity to not only refine the framework, but to reduce the potential to ignore important framework elements in the software testing domain, through repetition of the development procedure.

Determination of influence weight percentages by the software test manager is based on their software testing experience and expertise, but is not guided by any systematic support. Therefore, these determinations of influence weight percentages may only approximate the relationships that exist for a particular software testing situation. The limitations of expert knowledge have been well recognised in research on fuzzy cognitive maps and artificial intelligence. Some of the limitations are: experts, in common with other humans, are not always reliable (Groumpos, 2010); expert knowledge also includes the expert's ignorance and prejudice (Kosko, 1991); expert knowledge is never completely accurate and always contains some level of subjectivity (Salmeron, 2010); and expert judgements have the potential to be overly simplistic (Wooff et al., 2002). The framework does not provide a way to help software test managers systematically develop their expert opinions and judgments about influence weight percentages. Nor does the framework offer the potential to capture software test managers' motives and perspectives, and ultimately transform their opinions and

judgments about influence weight percentages to the crisp values used in the framework. This framework limitation was recognised, but is not addressed by this research, and is therefore a logical area for future research (see Section 8.7.1 of this chapter).

Software test managers, decision makers, or anyone else for that matter, may have difficulty in specifying point estimates for the magnitude of the influence weightings of relationships in the framework. Alternatives to point estimates exist (Grant & Osei-Bryson, 2005; Groumpos, 2010). Software test manager estimates of the magnitude of the influence weighting between elements C_i and C_j may be problematic. But software test managers may be able to specify that the magnitude of the influence weighting between elements C_i and C_j is greater, or less, than the magnitude of the influence weighting of the relationship between elements C_i and C_k (Grant & Osei-Bryson, 2005). Another option is for software test managers to provide interval estimates of the magnitude of the influence weightings. The intervals can be described by a qualitative term or a fuzzy linguistic variable (Grant & Osei-Bryson, 2005; Groumpos, 2010). This latter option presents a strong possibility for future research (see Section 8.7.1 of this chapter).

Applying fuzzy cognitive map analysis to evaluate the framework model has a limitation because of the high level nature of the framework. Fuzzy cognitive map (FCM) analysis results in this research are dependent on the threshold value assigned by the software test manager, and can yield results that provide no decision. This makes it difficult for the software test manager to interpret the results of FCM analysis, and determine the impact of those results on the potential decision. The decision support framework is represented at a high level, despite encapsulating many concepts and relationships in elements C1 to C3.1 inclusive. This high level representation was an explicit design choice, made so that the framework does not provide too many concepts and relationships to confront the software test manager with, and therefore invoke inflexibility. Too many concepts and relationships would make it difficult for the software test manager to interpret the decision support framework results. Thus, FCM analysis to evaluate the framework model has to be used judiciously for the current representation level of the framework. Judicious use of FCM analysis to evaluate the framework model would be based on meaningful results for the potential decision.

8.7 Future Research

This research forms the basis for looking at a specific phenomenon and building a framework around this phenomenon. The decision support framework (DSF) was developed to work with and be applied to software testing, and to be used by the software test manager.

8.7.1 Research Extensions

This research can be extended. Possible extensions include the following.

1. A look at the factor contribution analysis and how it could be extended to include a temporal perspective. The temporal perspective, as defined here, refers to time delays in achieving the goal, not to any time delays in the effect between a causal element and its affected element. Thus, when an element's factors do not achieve 100%, the question asked is: What affect will that have on how much more time it will take to achieve the goal? The initial temporal information would be derived from the test plan and other software development or test management documentation.

2. Develop a set of questions to assist and guide the software test manager to determine point estimates of the magnitude of influence weightings, for the relationships in the decision support framework.

3. Investigate the possibility of developing interval influence weightings for the relationships in the decision support framework. Such interval influence weightings could be described by qualitative terms or fuzzy linguistic variables, and translated to a range of numeric estimates of the influence weightings. It would have to be determined if such interval influence weightings are useful for software test managers, and can be used successfully with the analytical techniques discussed in Chapter 6.

4. A spreadsheet application was designed to perform what-if analysis for the factor contribution analysis. This application could be extended and made more user friendly. It could also be developed whereby DSF parameters could be entered, and the user could select the particular analysis technique (based on those discussed in Chapter 6). This would make it even easier for the software test manager to plan, analyse and perform risk management associated with the software being tested.

5. Examine the variations in FCM inference calculations, with the aim of making FCM analysis a more discriminating analytical technique for risk assessment when planning software testing. Possible sources of variation in the FCM inference calculations are the update equation and the transformation function of the update equation. Recently, Groumpos (2010) has proposed a new formulation of the update equation to yield smoother variation in the values of concepts after each calculation (Groumpos, 2010). It is known that appropriate use of certain transformation functions (such as the sigmoid function), can transform concept values of FCM state vectors to any point in the real unit fuzzy interval [0,1] (Grant & Osei-Bryson, 2005; Tsadiras, 2008): rather than transform concept values to 0 or 1, as the step function does (Grant & Osei-Bryson, 2005; Kosko, 1997; Tsadiras, 2008) – the step function was used in this research.

6. The Decision Support Framework (DSF) use of fuzzy cognitive map (FCM) analysis could be extended to include a temporal perspective. The temporal perspective refers to time delays in achieving the goal, not to any time delays in the effect between a causal element and its affected element. Thus, at equilibrium not only the attainment of the goal would be analysed, but also the temporal consequence at that point. Research of the literature has not found any FCM analysis that deals with the temporal aspects of the problem in the sense that temporal is used here.

8.7.2 Peripheral Research Directions

Peripheral research directions are those areas that will take extensive research and development to address. Some of the peripheral research is due to the lateral consequences brought on by issues and avenues of research discovered during this research. Here are some of the possibilities.

Develop an algorithm that not only takes into account the effects of the factors on the influence weightings from the primary elements to the goal, but also the effects of the factors on the influence weightings to other elements.

Currently the total factor contribution (100%) for an element is considered. Percentages can be assigned to each factor within an element. Additional work can be done to examine an element's individual factors, and how they interact. For example, if the Technical support

factor End-user environment requires that all personnel involved in the testing have a security clearance: How does that affect the Test management factor Organising test staff?

Classify and refine the type of software to be tested into categories. Then use the categories in test planning and risk management within the DSF. Develop a matrix for the software test manager based on the categorisation, and thereby provide an additional tool to enhance their test planning efforts. This tool could also be developed as a simple application.

Identify suitable real world software development projects and work through the entire STLC, with the software test manager, using the DSF. Use each analytical technique discussed in Chapter 6 and analyse the results.

One other possible area of research is to extend the DSF approach, its analytical techniques and considerations into other phenomena, such as evaluation and risk management associated with disaster recovery or with information technology (IT) business management. Another possible area is the application of the DSF in IT management and risk analysis.

8.8 Concluding Remarks

Addressing the research questions paved the way to satisfy the research objectives, and to achieve the goal for this research "***To provide a practical, integrated and conceptually sound decision support framework, which can be used to assist test managers achieve successful software testing.***". The research goal directed this research to examine relevant issues from the domains of software testing, the software test manager and decision support. These issues were synthesised, integrated and systematically combined into the design, development and use of the decision support framework, thereby adding to the research knowledge on decision support in software testing.

Decision support for software testing is an area that has seen little to no research. It is an area that needs to be researched and utilised to provide tools that can assist the software test manager in their decision making. Software testing is a difficult area that is becoming more complex to manage. The different types of software to be tested, and the issues within those types of software, make software testing a very complex area to address. Much needs to be done to provide frameworks that can be used to assist the software test manager with the intricacies of decision making. The development of the decision support framework provides

the software test manager with a tool that can assist them in their test planning and risk management of successful software testing.

The decision support framework is of great benefit because it can be used in risk management planning before testing, during testing and after testing. It provides an integrated structure of components for the software test manager to explore the possible risk consequences for the type of software application they plan to test, or are testing. These risk consequences are based on the availability and change of the numerous variables that affect the capacity of the software test manager to achieve the goal of successful software testing.

Figure 8.2 was first introduced, with a different caption, in Chapter 2, as a signpost to indicate the direction that development of this research would take. This figure is reproduced here to signify the culmination of this research, and to emphasise the major contribution of the research, and the links of that research contribution to three contextual domains – software testing, the software test manager and decision support.

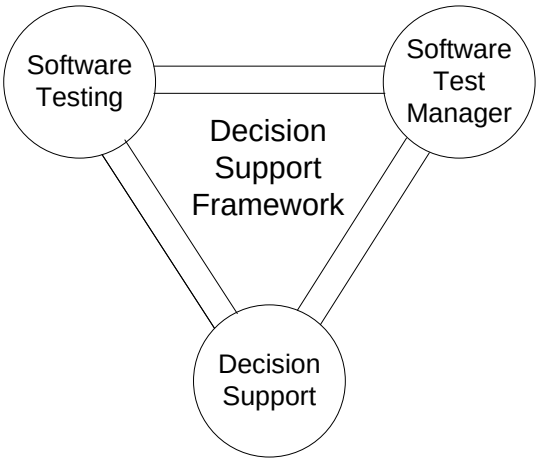


Figure 8.2: High level view of the research contribution

- ||| -

REFERENCES

Ad hoc testing. (2011). Retrieved from the Wikipedia website: http://en.wikipedia.org/wiki/Ad_hoc_testing

Afzal, W. & Torkar, R. (2008). Incorporating metrics in an organizational test strategy. *Proceedings of the IEEE International Conference on Software Testing Verification and Validation Workshop, 2008, Lillehammer, Norway*, (pp. 304-315). doi: 10.1109/ICSTW.2008.23

Alter, S. (1977). A taxonomy of decision support systems. *Sloan Management Review*, *19*(1), 39-56. Retrieved from http://sloanreview.mit.edu/

Alter, S. (1980). *Decision support systems: Current practice and continuing challenges*. Reading, MA: Addison-Wesley.

Alter, S. (2001). Which life cycle-work system, information system, or software? *Communications of the Association for Information Systems*, *7*(17), 1-54. Retrieved from http://aisel.aisnet.org/cgi/viewcontent.cgi?article=2830&context=cais

Alter, S. (2002). *Information systems: Foundation of e-business* (4th ed.). Upper Saddle River, NJ: Prentice Hall.

Alter, S. (2004). A work system view of DSS in its fourth decade. *Decision Support Systems*, *38*(3), 319-327. doi: 10.1016/j.dss.2003.04.001

Amitysoft Technologies. (2006). *Vibrant trends in software testing* [Press release]. Retrieved from http://www.amitysoft.com/press27.aspx

Ammann, P. & Offutt, J. (2008). *Introduction to software testing*. New York, NY: Cambridge University Press.

André, V. (2008). Testing risk assessment [Web log post]. Retrieved from http://knolstuff.com/profiles/blog/show?id=1781665:BlogPost:31659

Application. (2011). Retrieved from the TechTarget website: http://searchsoftwarequality.techtarget.com/definition/application

Application program. (2011). Retrieved from the TechTarget website: http://searchsoftwarequality.techtarget.com/definition/application-program

Application software. (2011). Retrieved from the Wikipedia website: http://en.wikipedia.org/wiki/Application_program

Applied Testing and Technology. (2011). *Software testing glossary*. Retrieved from http://www.aptest.com/glossary.html

Australian Bureau of Statistics. (2010). *Experimental estimates of industry multifactor productivity, 2009-10* (Cat. no. 5260.0.55.002). Retrieved 10 April, 2011, from http://www.abs.gov.au/AUSSTATS/abs@.nsf/ProductsbyCatalogue/A834424832179D03CA25739B00158F99?OpenDocument

Australian Bureau of Statistics. (2011). *Australian national accounts: National income, expenditure and product, Mar 2011* (Cat. no. 5206.0). Retrieved 10 April, 2011, from http://www.abs.gov.au/AUSSTATS/abs@.nsf/ProductsbyCatalogue/52AFA5FD696482CACA25768D0021E2C7?OpenDocument

Avižienis, A., Laprie, J.-C., Randell, B. & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing, 1*(1), 11-33. doi: 10.1109/TDSC.2004.2

Baltzan, P., Phillips, A., Lynch, K. & Blakey, P. (2010). *Business driven information systems*. North Ryde, NSW: McGraw-Hill.

Bangia, R. (2007). *Multimedia and web technology* (2nd ed.). New Delhi: Laxmi.

Bannerman, P. L. (2008). Risk and risk management in software projects: A reassessment. *Journal of Systems and Software, 81*(12), 2118-2133. doi: 10.1016/j.jss.2008.03.059

Beizer, B. (1983). *Software testing techniques*. New York, NY: Van Nostrand Reinhold.

Beizer, B. (1990). *Software testing techniques* (2nd ed.). London: International Thomson Computer Press.

Bertolino, A. (2007). Software testing research: Achievements, challenges, dreams. *Proceedings of the Future of Software Engineering, 2007, Minneapolis, Minnesota*, (pp. 85-103). doi: 10.1109/FOSE.2007.25

Black, R. (2002). *Managing the testing process: Practical tools and techniques for managing hardware and software testing* (2nd ed.). New York, NY: Wiley.

Blanchard, K. H. & Peale, N. V. (1988). *The power of ethical management*. New York, NY: Morrow.

Boehm, B. W. (1981). *Software engineering economics*. Englewood Cliffs, NJ: Prentice Hall.

Boehm, B. W. (1991). Software risk management: Principles and practices. *IEEE Software, 8*(1), 32-41. doi: 10.1109/52.62930

Bogue, R. (2005a). *Anatomy of a software development role: Development manager*. Retrieved from the QuinStreet website: http://www.developer.com/mgmt/article.php/3529081

Bogue, R. (2005b). *Anatomy of a software development role: Project manager*. Retrieved from the QuinStreet website: http://www.developer.com/mgmt/article.php/3526491

Bohanec, M. (2001). What is Decision Support? In C. Bavec, M. Bernik, M. Bohanec, T. Domanjko, S. Dragan, M. Gams, . . . T. Urbančič (Eds.), *Proceedings of the 4th International Multi-conference Information Society, 2001, Ljubljana, Slovenia, A*, (pp.86-89). Ljubljana, Slovenia: Institut Jožef Stefan. Retrieved from http://kt.ijs.si/MarkoBohanec/pub/WhatDS.pdf

Borysowich, C. (2005). Testing: Sample technical risk assessment questionnaire [Web log post]. Retrieved from http://it.toolbox.com/blogs/enterprise-solutions/testing-sample-technical-risk-assessment-questionnaire-3180

Borysowich, C. (2007). Establishing a test environment [Web log post]. Retrieved from http://it.toolbox.com/blogs/enterprise-solutions/establishing-a-test-environment-17420

Bradshaw, S. (2004). *Test metrics: A practical approach to tracking & interpretation*. Paper presented at the International Conference on Software Testing Analysis & Review (STAREAST 2004), Orlando, Florida, (pp. 1-10). Retrieved from http://www.stickyminds.com/getfile.asp?ot=XML&id=11480&fn=XDD11480filelistfilename1%2Edoc

Briand, L. C., El Emam, K., Freimut, B. G. & Laitenberger, O. (2000). A comprehensive evaluation of capture-recapture models for estimating software defect content. *IEEE Transactions on Software Engineering*, *26*(6), 518-540. doi: 10.1109/32.852741

Briand, L. C., Feng, J. & Labiche, Y. (2002). *Experimenting with genetic algorithms and coupling measures to devise optimal integration test orders* (Technical Report SCE-02-03, version 3, Carleton University). Retrieved from http://squall.sce.carleton.ca/pubs/tech_report/TR_SCE-02-03.pdf

Brooks, F. P. (1995). *The mythical man-month: Essays on software engineering* (Anniversary ed.). Reading, MA: Addison-Wesley.

Bryant, A. & Charmaz, K. (Eds.). (2010). *The SAGE handbook of grounded theory*. Thousand Oaks, CA: SAGE.

Bunke, H., Kandel, A. & Last, M. (Eds.). (2004). *Series in machine perception and artificial intelligence: Vol 56. Artificial intelligence methods in software testing*. Singapore: World Scientific.

Burns, N. & Grove, S. K. (2009). *The practice of nursing research: Appraisal, synthesis, and generation of evidence* (6th ed.). St. Louis, MO: Elsevier Saunders.

Butcher, M., Munro, H. & Kratschmer, T. (2002). Improving software testing via ODC: Three case studies. *IBM Systems Journal*, *41*(1), 31-44. doi: 10.1147/sj.411.0031

Charette, R. N. (2005). Why software fails. *IEEE Spectrum*, *42*(9), 42-49. doi: 10.1109/MSPEC.2005.1502528

Charmaz, K. (2006). *Constructing grounded theory: A practical guide through qualitative analysis*. London: SAGE.

Chen, Y., Probert, R. L. & Robeson, K. (2004). Effective test metrics for test strategy evolution. In H. Lutfiyya, J. Singer & D. A. Stewart (Eds.), *Proceedings of the 2004 Conference of the Centre for Advanced Studies on Collaborative Research, Markham, Ontario, Canada*, (pp. 111-123). Retrieved from http://www.site.uottawa.ca/~ychen/Paper_CASCON04.pdf

Client. (2011). In *BusinessDictionary.com dictionary*. Retrieved from the WebFinance website: http://www.businessdictionary.com/definition/client.html

Consumer. (2011). In *BusinessDictionary.com dictionary*. Retrieved from the WebFinance website: http://www.businessdictionary.com/definition/consumer.html

Copeland, L. (2004). *A practitioner's guide to software test design*. Norwood, MA: Artech House.

Craig, R. D. & Jaskiel, S. P. (2002). *Systematic software testing*. Boston, MA: STQE.

Customer. (2011). In *BusinessDictionary.com dictionary*. Retrieved from the WebFinance website: http://www.businessdictionary.com/definition/customer.html

D'Angelo, E. (2008). *Why is test planning so hard?* [Powerpoint slides]. Paper presented at the Quality Engineered Software and Testing Conference, Toronto, British Columbia, Canada, (pp. 1-32). Retrieved from http://www.google.com.au/url?sa=t&rct=j&q=why%20is%20test%20planning%20so%20hard%3F&source=web&cd=1&ved=0CCQQFjAA&url=http%3A%2F%2Fwww.swosqg.org%2Fpresentations%2FNov2008-TestPlanning.ppt&ei=NK1ET6naJsmoiAey1u38Ag&usg=AFQjCNGK7T0asPTYbFPVzTHLRHestbbTqQ&cad=rja

Daintith, J. (2004a). Software quality assurance. In *A dictionary of computing*. Retrieved from the HighBeam Research website: http://www.encyclopedia.com/doc/1O11-softwarequalityassurance.html

Daintith, J. (2004b). Software reliability. In *A dictionary of computing*. Retrieved from the HighBeam Research website: http://www.encyclopedia.com/doc/1O11-softwarereliability.html

Dedolph, F. M. (2003). The neglected management activity: Software risk management. *Bell Labs Technical Journal*, *8*(3), 91-95. doi: 10.1002/bltj.10077

Dennis, A., Wixom, B. H. & Roth, M. R. (2006). *System analysis and design* (3rd ed.). Hoboken, NJ: Wiley.

Dhunna, M. & Dixit, J. B. (2010). *Information technology in business management*. New Delhi: University Science Press.

Dick, S. & Kandel, A. (2005). *Series in machine perception and artificial intelligence: Vol 63. Computational intelligence in software quality assurance*. Hackensack, NJ: World Scientific.

Dustin, E. (2003). *Effective software testing: 50 specific ways to improve your testing*. Boston, MA: Addison-Wesley.

Dustin, E., Garrett, T. & Gauf, B. (2009). *Implementing automated software testing: How to save time and lower costs while raising quality*. Upper Saddle River, NJ: Addison-Wesley.

Dustin, E., Rashka, J. & Paul, J. (1999). *Automated software testing: Introduction, management, and performance*. Reading, MA: Addison-Wesley.

Editorial. (n.d.). Software testing life cycle [Web log post]. Retrieved 18 June, 2010, from http://editorial.co.in/software/software-testing-life-cycle.php

End user. (2010). In *Tech Terms computer dictionary*. Retrieved from the Techterms.com website: http://www.techterms.com/definition/enduser

Engström, E. & Runeson, P. (2011). *Decision support for test management and scope selection in a software product line context* [Abstract]. Paper presented at the 1st International Workshop on Variability-Intensive Systems Testing Validation & Verification Co-located with the Fourth International Conference on Software Testing, Verification and Validation, Berlin, Germany, (pp. 1-4). Retrieved from http://www.lunduniversity.lu.se/o.o.i.s?id=12683&postid=1890800

Everett, G. D. & McLeod, R., Jr. (2007). *Software testing: Testing across the entire software development life cycle*. Hoboken, NJ: Wiley Interscience.

Fan, A. (2010). Software testing life cycle [Web log post]. Retrieved from http://www.angelinefan.com/2010/05/05/software-testing-life-cycle/

Farrell-Vinay, P. (2008). *Manage software testing*. Boca Raton, FL: Auerbach.

Framework. (2008). Retrieved from the TechTarget website: http://whatis.techtarget.com/definition/0,,sid9_gci1103696,00.html

Frankl, P. G., Hamlet, R. G., Littlewood, B. & Strigini, L. (1998). Evaluating testing methods by delivered reliability. *IEEE Transactions on Software Engineering*, *24*(8), 586-601. doi: 10.1109/32.707695

Friedman, M. A. & Voas, J. M. (1995). *Software assessment: Reliability, safety, testability*. New York, NY: Wiley.

Garvin, D. A. (1984). What does "product quality" really mean? *Sloan Management Review (pre-1986), 26*(1), 25-43. Retrieved from http://sloanreview.mit.edu/

Ghezzi, C., Jazayeri, M. & Mandrioli, D. (1991). *Fundamentals of software engineering*. Englewood Cliffs, NJ: Prentice Hall.

Glaser, B. G. & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. Chicago, IL: Aldine.

Grant, D. & Osei-Bryson, K.-M. (2005). Using fuzzy cognitive maps to assess MIS organizational change impact. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences, Waikoloa, Hawaii, 8*, 1-11. Washington, DC: IEEE Computer Society. doi: 10.1109/HICSS.2005.658

Groumpos, P. P. (2010). Fuzzy cognitive maps: Basic theories and their application to complex systems. In M. Glykas (Ed.), *Studies in fuzziness and soft computing*: *Vol 247. Fuzzy cognitive maps: Advances in theory, methodologies, tools and applications*, (pp. 1-22). Berlin: Springer.

Hailpern, B. & Santhanam, P. (2002). Software debugging, testing, and verification. *IBM Systems Journal*, *41*(1), 4-12. doi: 10.1147/sj.411.0004

Hass, A. M. J. (2008). *Guide to advanced software testing*. Norwood, MA: Artech House.

Hayen, R. L. (2006). Investigating decision support systems frameworks. *Issues in Information Systems*, *VII*(2), 9-13. Retrieved from http://iacis.org/iis/2006/Hayen.pdf

Hermadi, I. (2004). *Genetic algorithm based test data generator* (Masters thesis, King Fahd University of Petroleum & Minerals, Saudi Arabia). Retrieved from http://eprints.kfupm.edu.sa/10546/1/10546.pdf

Hoffer, J. A., George, J. F. & Valacich, J. S. (2011). *Modern systems analysis and design* (6th ed.). Upper Saddle River, NJ: Prentice Hall.

Holtsnider, B., Wheeler, T., Stragand, G. & Gee, J. (2010). *Agile development & business goals: The six week solution*. Burlington, MA: Morgan Kaufmann.

Institute of Electrical and Electronics Engineers. (1990). *IEEE standard glossary of software engineering terminology* (IEEE Std 610.12-1990). doi: 10.1109/IEEESTD.1990.101064

Institute of Electrical and Electronics Engineers. (2008). *IEEE standard for software and system test documentation* (IEEE Std 829-2008). doi: 10.1109/IEEESTD.2008.4578383

Institute of Electrical and Electronics Engineers. (2010). *IEEE standard classification for software anomalies* (IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)). doi: 10.1109/IEEESTD.2010.5399061

Ireland, A. (2005). *Software engineering 4, Lecture 1: The software testing life-cycle* [Lecture PDF slides]. Retrieved from http://www.macs.hw.ac.uk/~air/se4/pdflec/lec-1-life-cycle-a5.pdf

Jaideep (2008a). 5 essentials while building test environment for software testing [Web log post]. Retrieved from http://itknowledgeexchange.techtarget.com/quality-assurance/5-essentials-while-building-test-environment-for-software-testing/

Jaideep (2008b). Twelve essential steps of software testing life cycle (STLC) [Web log post]. Retrieved from http://itknowledgeexchange.techtarget.com/quality-assurance/twelve-essential-steps-of-software-testing-life-cycle-stlc/

Jaideep (2008c). What is a testing environment for software testing? [Web log post]. Retrieved from http://itknowledgeexchange.techtarget.com/quality-assurance/what-is-a-testing-environment-for-software-testing/

Jaideep (2008d). Why build a test environment for software testing? [Web log post]. Retrieved from http://itknowledgeexchange.techtarget.com/quality-assurance/why-build-a-test-environment-for-software-testing/

James, K. L. (2009). *Software development process*. New Delhi: PHI Learning.

Jentzsch, R. (2008). *Model and modules - Training example*. Unpublished manuscript.

Johnson, D. W. (2011). *The role of a software test manager*. Retrieved from the TechTarget website: http://searchsoftwarequality.techtarget.com/tip/The-role-of-a-software-test-manager

Jones, C. (2007). *Estimating software costs: Bringing realism to estimating* (2nd ed.). New York, NY: McGraw-Hill.

Juristo, N., Moreno, A. M. & Strigel, W. (2006). Guest editors' introduction: Software testing practices in industry. *IEEE Software, 23*(4), 19-21. doi: 10.1109/MS.2006.104

Kahai, P., Namuduri, K. R. & Thompson, H. (2005). A decision support framework for automated screening of diabetic retinopathy. *International Journal of Biomedical Imaging, 2006*, Article ID 45806, 1-8. doi: 10.1155/IJBI/2006/45806

Kalogirou, S. A. (Ed.). (2007). *Artificial intelligence in energy and renewable energy systems*. New York, NY: Nova Science.

Kaner, C. & Bach, J. (2001). *Rapid test planning* [PDF slides]. Paper presented at the Software Testing, Analysis & Review Conference (STARWEST 2001), San José, California, (pp. 1-163). Retrieved from http://www.kaner.com/pdfs/RapidTestPlanning2001.pdf

Kaner, C., Bach, J. & Pettichord, B. (2002). *Lessons learned in software testing: A context-driven approach*. New York, NY: Wiley.

Kaner, C., Falk, J. L. & Nguyen, H. Q. (1999). *Testing computer software* (2nd ed.). New York, NY: Wiley.

Kay, R. (2002). *Quickstudy: System development life cycle*. Retrieved from the ComputerWorld website: http://www.computerworld.com/s/article/71151/System_Development_Life_Cycle

Kayne, R. (2011). *What are the different types of software?* Retrieved from the Conjecture Corporation website: http://www.wisegeek.com/what-are-the-different-types-of-software.htm

Keen, P. G. W. & Scott Morton, M. S. (1978). *Decision support systems: An organizational perspective*. Reading, MA: Addison-Wesley.

Kendall, K. E. & Kendall, J. E. (1998). *System analysis and design* (4th ed.). Upper Saddle River, NJ: Prentice Hall.

Kerkhoff, W. (2002). *Software release management*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=23AFD7B5DF4A021F7491780394C44B3C?doi=10.1.1.97.6645&rep=rep1&type=pdf

Khosrowpour, M. (Ed.). (2000). *Challenges of information technology in the 21st century*. Hershey, PA: Idea Group.

Kitchenham, B. & Pfleeger, S. L. (1996). Software quality: The elusive target. *IEEE Software*, *13*(1), 12-21. doi: 10.1109/52.476281

Kosko, B. (1991). *Neural networks and fuzzy systems: A dynamical systems approach to machine intelligence*. Englewood Ciffs, NJ: Prentice Hall.

Kosko, B. (1997). *Fuzzy engineering*. Upper Saddle River, NJ: Prentice Hall.

Krishnamoorthy, C. S. & Raveev, S. (1996). *Artificial intelligence and expert systems for engineers*. Boca Raton, FL: CRC Press.

Larkman, D., Jentzsch, R. & Mohammadian, M. (2011). Software testing - Factor contribution analysis in a decision support framework. In J. Watada, G. Phillips-Wren, L. C. Jain & R. J. Howlett (Eds.), *Smart innovation, systems and technologies*: *Vol 10. Intelligent decision technologies: Proceedings of the 3rd International Conference on Intelligent Decision Technologies (IDT' 2011), Piraeus, Greece, July 20-22, 2011*, (pp. 897-905). Berlin: Springer-Verlag. doi: 10.1007/978-3-642-22194-1_88

Larkman, D., Mohammadian, M., Balachandran, B. & Jentzsch, R. (2010a). Fuzzy cognitive map for software testing using artificial intelligence techniques. In H. Papadopoulos, A. S. Andreou & M. Bramer (Eds.), *IFIP advances in information and communication technology*: *Vol 339. Artificial intelligence applications and innovations: Proceedings of the 6th IFIP WG 12.5 International Conference, AIAI 2010, Larnaca, Cyprus, October 6-7, 2010*, (pp. 328-335). Berlin: Springer. doi: 10.1007/978-3-642-16239-8_43

Larkman, D., Mohammadian, M., Balachandran, B. & Jentzsch, R. (2010b). General application of a decision support framework for software testing using artificial intelligence techniques. In G. Phillips-Wren, L. C. Jain, K. Nakamatsu & R. J. Howlett (Eds.), *Smart innovation, systems and technologies*: *Vol 4. Advances in intelligent decision technologies: Proceedings of the Second KES International Symposium IDT 2010, Baltimore, Maryland, July 28-30, 2010*, (pp. 53-63). Berlin: Springer-Verlag. doi: 10.1007/978-3-642-14616-9_5

Leffingwell, D. & Widrig, D. (2000). *Managing software requirements: A unified approach*. Boston, MA: Addison-Wesley.

Légaré, F., O'Connor, A. M., Graham, I. D. & Wells, G. A. (2006). Impact of the Ottawa decision support framework on the agreement and the difference between patients' and physicians' decisional conflict. *Medical Decision Making*, *26*(4), 373-390. doi: 10.1177/0272989X06290492

Lendermann, P., Low, M. Y. H., Gan, B. P., Julka, N., Chan, L. P., Lee, L. H., . . . Buckley, S. (2005). An integrated and adaptive decision-support framework for high-tech manufacturing and service networks. *Proceedings of the 37th Conference on Winter Simulation, Orlando, Florida*, (pp. 2052-2062). doi: 10.1109/WSC.2005.1574487

Levi9 Global Sourcing. (2009). *Testing glossary*. Retrieved from http://www.software-testing-outsourcing.com/levi9.html

Limaye, M. G. (2009). *Software testing: Principles, techniques, and tools*. New Delhi: Tata McGraw Hill.

Liu, Z.-Q. & Satur, R. (1999). Contextual fuzzy cognitive map for decision support in geographic information systems. *IEEE Transactions on Fuzzy Systems*, *7*(5), 495-507. doi: 10.1109/91.797975

Maidasani, D. (2007). *Software testing*. Daryaganj, Delhi: Firewall Media.

Mantere, T. (2003). *Automatic software testing by genetic algorithms* (Doctoral thesis, University of Vaasa, Finland). Retrieved from http://www.uwasa.fi/materiaali/pdf/isbn_952-476-003-7.pdf

Marick, B. (1995). *The craft of software testing: Subsystem testing including object-based and object-oriented testing*. Englewood Cliffs, NJ: Prentice Hall.

Martin, P. Y. & Turner, B. A. (1986). Grounded theory and organizational research. *Journal of Applied Behavioral Science*, *22*(2), 141-157. doi: 10.1177/002188638602200207

Mateou, N. H., Moiseos, M. & Andreou, A. S. (2005). Multi-objective evolutionary fuzzy cognitive maps for decision support. *Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Edinburgh, Scotland*, *1*, 824-830. doi: 10.1109/CEC.2005.1554768

Mathur, A. P. (2008). *Foundations of software testing*. Delhi: Pearson.

McMinn, P. (2004). Search-based software test data generation: A survey. *Software Testing, Verification and Reliability*, *14*(2), 105-156. doi: 10.1002/stvr.294

Meyer, B. (2008). Seven principles of software testing. *Computer*, *41*(8), 99-101. doi: 10.1109/MC.2008.306

Miller, J., Macdonald, F. & Ferguson, J. (2002). ASSISTing management decisions in the software inspection process. *Information Technology and Management*, *3*(1), 67-83. doi: 10.1023/a:1013112826330

Mosley, D. J. & Posey, B. A. (2002). *Just enough software test automation*. Upper Saddle River, NJ: Prentice Hall.

Mu, C.-P., Huang, H.-K. & Tian, S.-F. (2004). Fuzzy cognitive maps for decision support in an automatic intrusion response mechanism. *Proceedings of the International Conference on Machine Learning and Cybernetics, 2004, Shanghai, China*, *3*, 1789-1794. doi: 10.1109/ICMLC.2004.1382066

Myers, G. J. (2004). *The art of software testing* (T. Badgett, T. Thomas & C. Sandler, revised and updated). Hoboken, NJ: Wiley. (Original work published 1979)

Naik, K. & Tripathy, P. (2008). *Software testing and quality assurance: Theory and practice*. Hoboken, NJ: Wiley.

National Institute of Standards and Technology. (2002). *The economic impacts of inadequate infrastructure for software testing* (Planning report 02-3). Retrieved from http://www.nist.gov/director/planning/upload/report02-3.pdf

Novak, J. D. & Cañas, A. J. (2008). *The theory underlying concept maps and how to construct and use them* (Technical Report IHMC CmapTools 2006-01, Rev 01-2008). Retrieved from http://cmap.ihmc.us/publications/researchpapers/theoryunderlyingconceptmapshq.pdf

Oak, M. (2011). *Major types of software*. Retrieved from the Buzzle.com website: http://www.buzzle.com/articles/major-types-of-software.html

Operations management. (2011). Retrieved from the Wikipedia website: http://en.wikipedia.org/wiki/Operations_manager

Papageorgiou, E. I., Markinos, A. T. & Gemtos, T. A. (2010). Soft computing technique of fuzzy cognitive maps to connect yield defining parameters with yield in cotton crop production in central Greece as a basis for a decision support system for precision agriculture application. In M. Glykas (Ed.), *Studies in fuzziness and soft computing*: *Vol 247. Fuzzy cognitive maps: Advances in theory, methodologies, tools and applications*, (pp. 325-362). Berlin: Springer.

Papageorgiou, E. I., Stylios, C. D. & Groumpos, P. P. (2003). An integrated two-level hierarchical system for decision making in radiation therapy based on fuzzy cognitive maps. *IEEE Transactions on Biomedical Engineering*, *50*(12), 1326-1339. doi: 10.1109/TBME.2003.819845

Patton, R. (2006). *Software testing* (2nd ed.). Indianapolis, IN: Sams

Pezzè, M. & Young, M. (2008). *Software testing and analysis: Process, principles, and techniques*. Hoboken, NJ: Wiley.

PhD projects. (n.d.). Retrieved 10 December, 2011, from the Software Competence Center Hagenberg website: http://www.scch.at/en/forschung/forschungsprofil/promotionsvorhaben-am-scch

PhysOrg.com. (2009). *Software testing market resilient despite crisis: Report*. Retrieved from http://www.physorg.com/news155992141.html

Power, D. J. (2001). Supporting decision-makers: An expanded framework. *Proceedings of the 2001 Informing Science Conference, Krakow, Poland*, (pp. 431-436). Retrieved from http://proceedings.informingscience.org/IS2001Proceedings/pdf/PowerEBKSupp.pdf

Project manager. (2011). Retrieved from the Wikipedia website: http://en.wikipedia.org/wiki/Project_manager

Ramler, R. (2004). Decision support for test management in iterative and evolutionary development. *Proceedings of the 19th IEEE International Conference on Automated Software Engineering, Linz, Austria*, (pp. 406-409). doi: 10.1109/ASE.2004.31

Ramler, R. (n.d.). *Decision support for test management in iterative and evolutionary development*. Retrieved 10 December, 2011, from the Software Competence Center Hagenberg website: http://www.scch.at/en/ramler-rudolf-diss

Raza, F. N. (2009). Artifical intelligence techniques in software engineering (AITSE). In S. I. Ao, O. Castillo, C. Douglas, D. D. Feng & J.-A. Lee (Eds.), *Proceedings of the International MultiConference of Engineers and Computer Scientists, 2009, Hong Kong, China, I*, 1086-1088. Retrieved from http://www.iaeng.org/publication/IMECS2009/IMECS2009_pp1086-1088.pdf

Redmill, F. (1999). Why systems go up in smoke. *The Computer Bulletin*, *41*(5), 26-28. doi: 10.1093/combul/41.5.26

Rees, K., Coolen, F., Goldstein, M. & Wooff, D. (2001). Managing the uncertainties of software testing: A Bayesian approach. *Quality and Reliability Engineering International*, *17*(3), 191-203. doi: 10.1002/qre.411

Richardson, D. J., O'Malley, T. O., Moore, C. T. & Aha, S. L. (1992). Developing and integrating ProDAG in the Arcadia environment. *SIGSOFT Software Engineering Notes*, *17*(5), 109-119. doi: 10.1145/142882.143759

Risk analysis (business). (2011). Retrieved from the Wikipedia website: http://en.wikipedia.org/wiki/Risk_analysis_%28business%29

Rivers, A. T. & Vouk, M. A. (1999). Guiding resource constrained software testing. *Proceedings of the 10th International Symposium on Software Reliability Engineering, 1999, Boca Raton, Florida*, (pp. 180-188). doi: 10.1109/ISSRE.1999.809323

Ruhe, G. (2003a). Software engineering decision support - A new paradigm for learning software organizations. In S. Henninger & F. Maurer (Eds.), *Lecture Notes in Computer Science*: *Vol 2640*. *Advances in learning software organizations: 4th International Workshop, LSO 2002, Chicago, Illinois, August 6, 2002, Revised Papers*, (pp. 104-113). Berlin: Springer-Verlag. doi: 10.1007/978-3-540-40052-3_10

Ruhe, G. (2003b). Software engineering decision support: Methodology and applications. In G. Tonfoni & L. Jain (Eds.), *International series on advanced intelligence*: *Vol 3*. *Innovations in decision support systems*, (pp. 143-174). Magill, SA: Advanced Knowledge International. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.197.9019&rep=rep1&type=pdf

Russell, S. J. & Norvig, P. (2003). *Artificial intelligence: A modern approach*. Upper Saddle River, NJ: Prentice Hall.

Sadiq, R., Kleiner, Y. & Rajani, B. B. (2004). Fuzzy cognitive maps for decision support to maintain water quality in ageing water mains. *Proceedings of the 4th International Conference on Decision Making in Urban and Civil Engineering, Porto, Portugal*, (pp. 1-10). Retrieved from http://www.nrc-cnrc.gc.ca/obj/irc/doc/pubs/nrcc47305/nrcc47305.pdf

Salmeron, J. L. (2009). Supporting decision makers with fuzzy cognitive maps. *Research Technology Management*, *52*(3), 53-59. Retrieved from http://www.iriweb.org/Main/Library/RTM_Journal/Public_Site/Navigation/Publications/Research-Technology_Management/index.aspx

Salmeron, J. L. (2010). Fuzzy cognitive maps-based IT projects risks scenarios. In M. Glykas (Ed.), *Studies in fuzziness and soft computing*: *Vol 247*. *Fuzzy cognitive maps: Advances in theory, methodologies, tools and applications*, (pp. 201-215). Berlin: Springer.

Schuster, A., Sterritt, R., Adamson, K., Curran, E. P. & Shapcott, C. M. (2000). Towards a decision support system for automated testing of complex telecommunication networks. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, 2000, Nashville, Tennessee*, *3*, 2002-2006. doi: 10.1109/ICSMC.2000.886408

Senior management. (2011). Retrieved from the Wikipedia website: http://en.wikipedia.org/wiki/Senior_management

Shelly, G. B. & Rosenblatt, H. J. (2010). *System analysis and design* (8th ed.). Boston, MA: Course Technology.

Silva, C. (2008). What is software testing? [Web log post]. Retrieved from http://chamaras.blogspot.com/2008/09/what-is-software-testing.html

Siraj, A., Bridges, S. M. & Vaughn, R. B. (2001). Fuzzy cognitive maps for decision support in an intelligent intrusion detection system. *Proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference, 2001, Vancouver, British Columbia, Canada*, *4*, 2165-2170. doi: 10.1109/NAFIPS.2001.944405

Skyttner, L. (2005). *General systems theory* (2nd ed.). Singapore: World Scientific.

Słowiński, R. (Ed.). (1992). *Theory and decision library series D: System theory, knowledge engineering, and problem solving: Vol 11*. *Intelligent decision support: Handbook of applications and advances of the rough sets theory*. Dordrecht, Netherlands: Academic.

Smith, E. & Eloff, J. (2000). Cognitive fuzzy modeling for enhanced risk assessment in a health care institution. *IEEE Intelligent Systems and their Applications*, *15*(2), 69-75. doi: 10.1109/5254.850830

SMS Management & Technology. (2011). *Test managers* [Job advertisement]. Retrieved 7 July, 2011, from http://www.smsmt.com/Join-SMS/Job-Posting.aspx?job=test-managers-adelaide-200016155

Software product line. (2011). Retrieved from the Wikipedia website: http://en.wikipedia.org/wiki/Software_product_line

Sprague, R. M., Jr. & Carlson, E. D. (1982). *Building effective decision support systems*. Englewood Cliffs, NJ: Prentice-Hall.

Sree, U. (2008). *Software testing life cycle* [Flash Player slides]. Retrieved from the SlideShare website: http://www.slideshare.net/UdayaSree/software-testing-life-cycle-presentation

Strauss, C., Stummer, C., Bauer, C. & Trieb, A. (2009). A networked ubiquitous computing environment for damage prevention: A decision support framework for the insurance sector. *Proceedings of the International Conference on Parallel Processing Workshops, 2009, Vienna, Austria*, (pp. 276-281). doi: 10.1109/ICPPW.2009.79

Stylios, C. D., Georgoulas, G. & Groumpos, P. P. (2001). The challenge of using soft computing for decision support during labour. *Proceedings of the 23rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2001, Istanbul, Turkey*, *4*, 3835-3838. doi: 10.1109/IEMBS.2001.1019676

Stylios, C. S. & Georgopoulos, V. C. (2010). Fuzzy cognitive maps for medical decision support - A paradigm from obstetrics. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2010, Buenos Aires, Argentina*, (pp. 1174-1177). doi: 10.1109/IEMBS.2010.5626239

Swan, G. (2011). *IT skills shortage could delay software development: Report*. Retrieved from the International Data Group Communications website: http://www.techworld.com.au/article/376523/it_skills_shortage_could_delay_software_development_report/

Tamres, L. & Mills, S. (2002). *Introducing software testing*. Harlow, Essex: Addison-Wesley.

Test. (2008). In *Reverso English cobuild dictionary*. Retrieved from the Softissimo website: http://dictionary.reverso.net/english-cobuild/test

Test. (2009). In *WordWeb Online dictionary and thesaurus*. Retrieved from the WordWeb Software website: http://www.wordwebonline.com/search.pl?w=test

Test. (2011a). In *Cambridge dictionaries online: Learner's dictionary*. Retrieved from the Cambridge University Press website: http://dictionary.cambridge.org/dictionary/british/test_1?q=test

Test. (2011b). In *Dictionary.com dictionary*. Retrieved from http://dictionary.reference.com/browse/test

Test. (2011c). In *Merriam-Webster dictionary*. Retrieved from http://www.merriam-webster.com/dictionary/test

Test. (2011d). In *Oxford dictionaries*. Retrieved from the Oxford University Press website: http://oxforddictionaries.com/definition/test?rskey=FlReFG&result=1

Test. (2011e). In *The free dictionary by Farlex*. Retrieved from http://www.tfd.com/test

The Stanford Decisions and Ethics Center. (2009). *What is decision analysis?* Retrieved from http://decision.stanford.edu/library/the-principles-and-applications-of-decision-analysis-1/what-is-decision-analysis

ThinkExist.com. (2011). *Journey quotes*. Retrieved from http://thinkexist.com/quotations/journey/

Tian, J. (2005). *Software quality engineering: Testing, quality assurance, and quantifiable improvement*. Hoboken, NJ: Wiley.

Tsadiras, A. K. (2008). Comparing the inference capabilities of binary, trivalent and sigmoid fuzzy cognitive maps. *Information Sciences, 178*(20), 3880-3894. doi: 10.1016/j.ins.2008.05.015

Turban, E., Sharda, R. & Delen, D. (2011). *Decision support and business intelligence systems* (9th ed.). Upper Saddle River, NJ: Prentice Hall.

Turing, A. M. (1949). *Checking a large routine* (CategoryAMT/B, document 8). Retrieved from the Turing Digital Archive website: http://www.turingarchive.org/browse.php/B/8

van Gelder, T. J. (2010). The wise delinquency of decision makers. *Quadrant, LIV*(3), 40-43. Retrieved from http://sites.google.com/site/timvangelder/publications-1/wise-delinquency-of-decision-makers/WiseDelinquency.pdf?attredirects=0&d=1

Vijay (2007). Types of risks in software projects [Web log post]. Retrieved from http://www.softwaretestinghelp.com/types-of-risks-in-software-projects/#

Wang, K., Hjlmervik, O. R. & Bremdal, B. (2001). *Introduction to knowledge management: Principles and practice*. Trondheim, Norway: Tapir Academic Press.

Wasson, C. S. (2006). *System analysis, design, and development concepts, principles, and practices*. Hoboken, NJ: Wiley Interscience.

Wegener, J., Baresel, A. & Sthamer, H. (2002). Suitability of evolutionary algorithms for evolutionary testing. *Proceedings of the 26th Annual International Computer Software and Applications Conference, 2002, Oxford, England*, (pp. 287-289). doi: 10.1109/CMPSAC.2002.1044566

Weiss, G., Pomberger, G., Beer, W., Buchgeher, G., Dorninger, B., Pichler, J., . . . Weinreich, R. (2009). Software engineering - Processes and tools. In B. Buchberger, M. Affenzeller, A. Ferscha, M. Haller, T. Jebelean, E. P. Klement, . . . W. Windsteiger (Eds.), *Hagenberg research*, (pp. 157-236). Berlin: Springer-Verlag. doi: 10.1007/978-3-642-02127-5

White, D. (2002). *Knowledge mapping & management*. Hershey, PA: IRM Press.

Whittaker, J. A. (2000). What is software testing? And why is it so hard? *IEEE Software, 17*(1), 70-79. doi: 10.1109/52.819971

Wooff, D. A., Goldstein, M. & Coolen, F. P. A. (2002). Bayesian graphical models for software testing. *IEEE Transactions on Software Engineering, 28*(5), 510-525. doi: 10.1109/TSE.2002.1000453

Xirogiannis, G. & Glykas, M. (2007). Intelligent modeling of e-business maturity. *Expert Systems with Applications, 32*(2), 687-702. doi: 10.1016/j.eswa.2006.01.042

Yin, R. K. (2003). *Applied social research methods series: Vol 5. Case study research: Design and methods* (3rd ed.). Thousand Oaks, CA: SAGE.

APPENDICES

Appendix A: Software Testing Approaches

This list is not meant to be comprehensive; only to provide a sample of testing approaches considered in the literature. Some of the following approaches have been discussed in Chapter 3.

Ad Hoc Testing. A testing phase where the tester tries to "break" the system by randomly trying the system's functionality. Negative testing can be included as well (Applied Testing and Technology, 2011).

Basis Path Testing. A white box test case design technique that uses the algorithmic flow of the program to design tests (Levi9 Global Sourcing, 2009).

Black Box Testing. Testing based on an analysis of the specification of a piece of software without reference to its internal workings. The goal is to test how well the component conforms to the published requirements for the component (Applied Testing and Technology, 2011).

Bottom Up Testing. An approach to integration testing where the lowest level components are tested first; then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested (Levi9 Global Sourcing, 2009).

Boundary Testing. Test which focuses on the boundary or limit conditions of the software being tested. (Some of these tests are stress tests.) (Applied Testing and Technology, 2011)

Branch Testing. Testing in which all branches in the program source code are tested at least once (Levi9 Global Sourcing, 2009).

Compatibility Testing. Testing whether software is compatible with other elements of a system with which it should operate, e.g. browsers, operating systems, or hardware (Applied Testing and Technology, 2011).

Concurrency Testing. Multi-user testing geared towards determining the effects of accessing the same application code, module or database records. Identifies and measures the level of locking, deadlocking, use of single-threaded code and locking semaphores (Applied Testing and Technology, 2011).

Dynamic Testing. Testing software through executing it (Applied Testing and Technology, 2011). See also Static Testing.

Endurance Testing. Checks for memory leaks or other problems that may occur with prolonged execution (Applied Testing and Technology, 2011).

End-to-End Testing. Testing a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate (Levi9 Global Sourcing, 2009).

Functional Testing. Testing the features and operational behaviour of a product to ensure it corresponds with its specifications. Testing that ignores the internal mechanism of a system or component, and focuses solely on the outputs generated in response to selected inputs and execution conditions (Levi9 Global Sourcing, 2009).

Gray Box Testing. A combination of Black Box and White Box testing methodologies: testing a piece of software against its specification but using some knowledge of its internal workings (Levi9 Global Sourcing, 2009).

High Order Tests. Black-box tests conducted once the software has been integrated (Applied Testing and Technology, 2011).

Integration Testing. Testing of combined parts of an application to determine if they function together correctly. Usually this type of testing is performed after unit and functional testing. This type of testing is especially relevant to client/server and distributed systems (Applied Testing and Technology, 2011).

Installation Testing. Confirms that the application under test recovers from expected or unexpected events without loss of data or functionality. Events can include shortage of disk space, unexpected loss of communication, or power out conditions (Applied Testing and Technology, 2011).

Localization Testing. Tests how well software has been designed (localized) for a specific locality (Levi9 Global Sourcing, 2009).

Loop Testing. A white box testing technique that exercises program loops (Levi9 Global Sourcing, 2009).

Path Testing. Testing in which all paths in the program source code are tested at least once (Levi9 Global Sourcing, 2009).

Performance Testing. Testing conducted to evaluate the compliance of a system or component with specified performance requirements. Often this testing is performed using an automated test tool to simulate a large number of users (Applied Testing and Technology, 2011).

Quality Assurance. All those planned or systematic actions necessary to provide adequate confidence that a product, or service, is of the type and quality needed and expected by the customer (Applied Testing and Technology, 2011).

Regression Testing. Retesting a previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes made (Applied Testing and Technology, 2011).

Sanity Testing. Brief test of major functional elements of a piece of software to determine if it is basically operational (Applied Testing and Technology, 2011).

Smoke Testing. A quick-and-dirty test of the major functions of a piece of software to determine if they work. Smoke testing originated in the hardware testing practice of turning on a new piece of hardware for the first time, and considering it a success if it does not catch on fire (Applied Testing and Technology, 2011).

Static Testing. Analysis of a program carried out without executing the program (Levi9 Global Sourcing, 2009).

Structural Testing. Testing based on an analysis of the internal workings and structure of a piece of software (Levi9 Global Sourcing, 2009).

System Testing. Testing that attempts to discover if defects are properties of the entire system rather than of its individual components (Levi9 Global Sourcing, 2009).

Testing. The process of exercising software to verify that it satisfies specified requirements and to detect errors. The process of analysing a software item to detect the differences between existing and required conditions (that is, bugs), and to evaluate the features of the software item (Applied Testing and Technology, 2011). The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component (IEE, 1990).

Thread Testing. A variation of top-down testing where the progressive integration of components follows the implementation of subsets of the requirements, as opposed to the integration of components by successively lower levels (Applied Testing and Technology, 2011).

Top Down Testing. An approach to integration testing, where the component at the top of the component hierarchy is tested first, with lower level components being simulated by stubs. Tested components are then used to test lower level components (Applied Testing and Technology, 2011).

Unit Testing. Testing of individual software components (Applied Testing and Technology, 2011).

Validation. The process of evaluating software at the end of the software development process to ensure compliance with software requirements. The techniques for validation are testing, inspection and reviewing (Applied Testing and Technology, 2011).

Verification. The process of determining whether or not the products of a given phase of the software development cycle meet the implementation steps, and can be traced to the incoming objectives established during the previous phase. The techniques for verification are testing, inspection and reviewing (Applied Testing and Technology, 2011).

White Box Testing. Testing based on an analysis of internal workings and structure of a piece of software. Includes techniques such as Branch Testing and Path Testing. Also known as Structural Testing and Glass Box Testing (Applied Testing and Technology, 2011).

Definitions of Test

This research defines test as it relates to testing software and software systems. It is instructive to consider the various perspectives adopted for the definition of test represented in a selection of different test definitions sourced from various dictionaries.

Reverso English Cobuild Dictionary:
A Test is a deliberate action or experiment to find out how well something works ("Test", 2008).

Cambridge Learner's Dictionary:
A Test is an act of using something to find out whether it is working correctly or how effective it is ("Test", 2011a).

The Free Dictionary by Farlex:
A Test is a procedure for critical evaluation; a means of determining the presence, quality, or truth of something ("Test", 2011e).

Merriam-Webster:
A Test is a critical examination, observation, or evaluation ("Test", 2011c).

Dictionary.com:
A Test is the means by which the presence, quality, or genuineness of anything is determined ("Test", 2011b).

WordWeb Online:
A Test is a trial of something to see if or how it works ("Test", 2009).

Oxford Dictionaries:
A Test is a procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use ("Test", 2011d).

Appendix B: Terminology Used in this Research

Table B.1 lists the terminology used in this research, including the terminology developed for the decision support framework.

Table B.1: Terminology used in this research

Term	Definition
Application Software	A computer program designed for a specific purpose, to do work for end-users, or sometimes to do work for other application software ("Application", 2011; "Application software", 2011; IEEE, 1990).
Bug	A software bug occurs when: the software does not do something that the product specification says it should do; the software does something that the product specification says it should not do; the software does something that the product specification does not mention; the software does not do something that the product specification does not mention but should; and the software is difficult to understand, hard to use, slow, or – in the software tester's eyes – will be viewed by the end user as not right (Patton, 2006).
Concept	A perceived regularity in a phenomenon. A concept has a separate meaning, is described abstractly, and is given a label to identify it (Burns & Grove, 2009; Novak & Cañas, 2008).
Decision	A decision is the choice of the preferred alternative, the solution to a problem, a requirement or an opportunity (Bohanec, 2001; Turban et al., 2011).
Decision Framework	A way to organise a thought process about a phenomenon to provide a guide or support to achieve an overall goal or objective (Alter, 2002; Jentzsch, 2008).
Decision Making	Is the process of developing arguments for alternative courses of action and choosing amongst alternatives, with the aim of achieving a goal or goals (Bohanec, 2001; Turban et al., 2011).
Decision Support	Any computerised or non-computerised means, to assist an organisation's decision makers to improve their decision making within structured, semi-structured, or unstructured situations/issues/problems/requirements/opportunities (Alter, 2002; Salmeron, 2010; Turban et al., 2011).
Decision Support Framework	A guide whereby the user can take what they know about the type of phenomenon, and the environment in which the decision support framework will be used, and enhance one or more relevant aspects of decision making information to achieve an overall goal (Alter, 2002; Jentzsch, 2008).
Decision Support Framework Tool	The computer software that captures the textual and graphical representations of the decision support framework (DSF), and provides the basic enabling technology to permit the software test manager to employ the analytical techniques which accompany the DSF.
Element	An element represents a specific concept that has a relationship to the achievement of a defined goal.

End-user	The person or persons for which ultimately something is intended to be used by. The software end-user is the person who uses or operates the software ("End user", 2010).
Error	The amount by which the result is incorrect or deviates from expectation. The difference between a computed, observed or measured result and the correct result (IEEE, 1990).
Factor Contribution	The percentage for a factor that indicates how much the factor contributes to the element's ability to meet its influence weighting percentage(s).
Factors	Describe the details of an element's characteristics and supporting things that are needed to ensure that an element can meet its purpose.
Failure	The inability of a system to perform its required functions, thereby giving an incorrect or unexpected result. The result of a fault (IEEE, 1990).
Fault	An incorrect step, process, or data definition in a computer program. The manifestation of a mistake (IEEE, 1990).
Framework	A set of ideas and assumptions to organise a thought process about a phenomenon, such as a type of thing, issue, or situation (Alter, 2002).
Goal	There is only one goal in a decision support framework. The purpose of the framework is to support the achievement of that goal.
Influence Weighting	Indicates the strength of an element's influence on another element or achieving the goal, determined by user's knowledge and experience, with the values of influence weightings expressed as percentages.
Knowledge	Is the decision makers' practical understanding of or experience within a particular domain.
Mistake	A human unintentional action that produces an incorrect or unexpected result that the human is unaware of (Avižienis et al., 2004; IEEE, 1990).
Model	A simplified representation of a specific event, thing or situation in the real world. Models help inform the user, emphasise some features of reality, and downplay or ignore others (Alter, 2002; Turban et al., 2011). A graphical illustration or some type of mathematical illustration where multiple situations can be applied to a specific, concrete situation, as a way to inform the user of that situation (Turban et al., 2011).
Problem Solving	Problem solving deals with systems which do not meet their established goals, do not yield the predicted results, or do not work as planned (Turban et al., 2011).
Program Path	The sequence of instructions performed to execute the program (IEEE, 1990).

Program State	A program state is a snapshot into the state of an executing program on some input (Friedman & Voas, 1995). The program state is all of the information needed to restart an execution of the program (Friedman & Voas, 1995). Program state information includes the values of all programmer-defined variables, the current value of the program counter, the current scope, and information such as the contents of cache memory, main memory, and the program registers (Friedman & Voas, 1995).
Quality Assurance	The process of ensuring that software or a software system and its associated documentation are in all respects of sufficient quality for their purpose (Daintith, 2004a).
Relationships	Exist between elements and the goal, or with other elements, and are indicated by a connecting one way arc between elements.
Risk	The possibility of harm or loss. Typically risks are future uncertain events with a probability of occurrence, a consequence that follows if the event occurs, and the consequence has a potential for loss and an associated cost (Boehm, 1991; Dedolph, 2003; Vijay, 2007).
Risk Management	The identification, assessment, and prioritisation of a possible event or circumstance that can have negative influences on the meeting of an objective in the testing of software (Boehm, 1991; Dedolph, 2003).
Software	Computer programs, associated documentation, and data used to perform some task or tasks by a computer, or by other electronic devices (Brooks, 1995; IEEE, 1990).
Software Reliability	The probability of failure-free software operation for a specified period of time in a specified environment (Daintith, 2004b).
Software System	A set of coordinated, interacting software programs required to accomplish a specific function, or set of functions to achieve a specific purpose (Brooks, 1995; IEEE, 1990).
Software Test Manager	Has to evolve and successfully respond to the diverse requirements of rapidly changing software technologies, and of complex software testing cycles, while they also have to effectively lead the test team, and manage, implement and maintain an effective testing process. The software test manager must be mindful of the planning, scheduling and infrastructure needs to support the test process, and be able to communicate clearly with all stakeholders at all levels (http://www.testinginstitute.com; Johnson, 2011; SMS Management & Technology, 2011).
Software Testing	A set of activities performed on software and software systems with the purpose of finding errors in that software and software system. The process is directed at quality assurance, verification, validation, and reliability of the software and the software system (IEEE, 1990, 2008).

Software Testing Life Cycle	A set of steps or phases that identifies what test activities to carry out and when to accomplish those test activities (Editorial, n.d.; Jaideep, 2008b).
Successful Software Testing	The result of a set of activities that produces a product with a minimum number of issues, on time, within a given budget, and of sufficient quality to meet the requirements and expectations of the customer.
System Development Life Cycle	A series of steps that form a process in creating or altering systems into existence that includes models and methodologies that people use to develop systems (Baltzan et al., 2010; Dennis et al., 2006; Hoffer et al., 2011; Kendall & Kendall, 1998; Shelly & Rosenblatt, 2010).
System Software	Facilitates the operation and maintenance of a computer system, examples of system software include operating systems and utilities (IEEE, 1990).
Test Case	A set of conditions or variables linked to requirements under which a tester will determine whether software or a software system is working as intended and providing expected results (Ammann & Offutt, 2008; Hermadi, 2004; IEEE, 1990).
Test Planning	The activity of identifying and gathering the information that pertains to the particular software to be tested (Ammann & Offutt, 2008; Copeland, 2004; Mosley & Posey, 2002).
Validation	The process to determine if a software product, at the end of its development, meets its intended use: where intended use is defined by requirements and expectations (Ammann & Offutt, 2008; IEEE, 1990; Naik & Tripathy, 2008).
Verification	The process to determine if the products of a given software development phase satisfy the conditions established before the start of that phase (Ammann & Offutt, 2008; IEEE, 1990; Naik & Tripathy, 2008).

Appendix C: Abbreviations Used in this Research

Table C.1 lists the abbreviations used in this research.

Table C.1: Abbreviations used in this research

Abbreviation	Description
ABS	Australian Bureau of Statistics
AI	Artificial Intelligence
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BGM	Bayesian Graphical Models
CPA	Critical Path Analysis
CPU	Central Processing Unit
DLL	Dynamic Link Library
DSF	Decision Support Framework
DSR	Directional Signed Relationship
DSS	Decision Support System
EPA	Event Path Analysis
ERP	Enterprise Resource Planning
FCA	Factor Contribution Analysis
FCM	Fuzzy Cognitive Map
GDP	Gross Domestic Product
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IDT	Innovative Defence Technologies
IEEE	Institute of Electrical and Electronics Engineers
IIOP	Internet Inter-Orb Protocol
IPO	Input-Processing-Output
IT	Information Technology
NIST	National Institute of Standards and Technology
ODC	Orthogonal Defect Classification
OEM	Original Equipment Manufacturer
PDF	Portable Document Format
QA	Quality Assurance
RAM	Random Access Memory
RCT	Resource Constrained Testing
RPA	Risk Path Analysis
SDLC	System Development Life Cycle
STLC	Software Testing Life Cycle
TPWA	Total Path Weight Analysis
UC	University of Canberra
US	United States of America