Exploratory Study on an Innovative Use of COSMIC-FFP for Early Quality Assessment

Manar Abu Talib

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
at Concordia University
Montreal, Quebec, Canada

February 2007

©Manar Abu Talib, 2007

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Mrs. Manar Abu Talib

Entitled: Exploratory Study on an Innovative Use of COSMIC-FFP for Early Quality

Assessment

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science and Software Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

________________________ Chair

________________________ External Examiner

________________________ Examiner

________________________ Examiner

________________________ Examiner

________________________ Thesis Supervisor

________________________ Thesis Supervisor

Approved by ____
Department Chair or Graduate Program Director

_________________ 20__ __________________________________
Dr. Nabil Esmail, Dean
Faculty of Engineering and Computer Science

ABSTRACT

Exploratory Study on an Innovative Use of COSMIC-FFP for Early Quality Assessment

Manar Abu Talib, Ph.D.
Concordia University, 2007

The functional size measurement method, COSMIC-FFP, adopted as the ISO/IEC 19761 standard in 2003, was developed by the Common Software Measurement International Consortium (COSMIC). It focuses on the “user view” of functional requirements and is applicable throughout the development life cycle. As some of the software systems targeted by COSMIC-FFP are large-scale and inherently complex, feedback on their functional complexity would facilitate containment of that complexity throughout the software life cycle.

In this thesis, a new early quality assessment of COSMIC-FFP models is proposed. The benefits of this work include earlier prediction of the functional complexity of the behavior of software in the COSMIC-FFP context, right from the requirements phase, as well as a mechanism for generating black-box test cases from the COSMIC-FFP model, test case prioritization and test set adequacy monitoring and optimization within given budget constraints, and an early prediction of reliability based on Markov chains. We also present a study of the scales, units and scale types of both COSMIC-FFP and the Entropy-based Functional Complexity Measure that forms the basis of the testing assessment method we propose here. Previous studies have analyzed the scale types of many pieces of software, but not the concept of scale itself, nor how it is used in the design of a measurement method. Two well-known case studies are introduced to demonstrate the applicability of the proposed methods: the Hotel Accommodation System and the Railroad System.

We include a formalized COSMIC-FFP definition in the AS-TRM context (Autonomic Systems Timed Reactive Object Model), a language for the formal design of autonomic reactive systems developed at Concordia University. We introduce the Steam Boiler case study to demonstrate the applicability of formalizing COSMIC-FFP in the AS-TRM context. Future work based on this thesis can include the development of AS-TRM specifications for several benchmark case studies, and the collection of COSMIC-FFP measurement data for both the theoretical and empirical validation of the proposed measurement method.

The testing method proposed here has been adapted to a specific class of projects, namely Enterprise Resource Planning (ERP) projects, which are perceived to be mission-critical initiatives in many organizations. They can be found in business transformation programs and are instrumental in improving organizational performance.

Acknowledgments

A Ph.D. thesis should bring the writer feelings of fulfillment, accomplishment and joy, which is certainly the case for me. This achievement would have been impossible, however, without a great deal of help and support.

On an academic level, I would like to thank Dr. Olga Ormandjieva and Dr. Alain Abran for their astute and constructive criticisms which have helped me shape my thesis into its final form, and for their support and encouragement.

On a professional level, I would like to thank all my colleagues in the Computer Science and Software Engineering department, as well as the staff of the department for their commitment to further education.

On a personal note, I would like to thank my parents, my brothers and my sister for their support, and my husband Adel for his patience and understanding throughout the writing process. I would also like to thank my two kids for their cooperation and forbearance.

Table of Contents

TABLE OF CONTENTS .... **VI**

LIST OF FIGURES .... **VIII**

LIST OF TABLES .... **X**

CHAPTER I: INTRODUCTION **1**

1.1 MOTIVATION 3
1.1.1 Software Quality *4*
1.1.2 Software Size and COSMIC-FFP *5*
1.1.3 Software Complexity *8*
1.2 MAJOR CONTRIBUTIONS AND THESIS OUTLINE 9
1.2.1 Definition *11*
1.2.2 Planning and Operation *12*
1.2.3 Interpretations *14*

CHAPTER II: BACKGROUND **16**

2.1 FUNCTIONAL SIZE MEASUREMENT: STATE OF THE ART 16
2.1.1 Function Points *16*
2.1.2 ISO Standards *22*
2.1.3 COSMIC-FFP *25*
2.2 ENTROPY MEASUREMENT IN SOFTWARE ENGINEERING 29
2.3 SOFTWARE TESTING 30
2.4 SOFTWARE RELIABILITY 34
2.4.1 Markov Chains *35*
2.5 AS-TRM 38
2.6 SCALE TYPES 42

CHAPTER III: PROPOSED THESIS **46**

3.1 OBJECTIVE 46
3.2 EXPECTED RESULTS AND BENEFITS 48
3.3 METHODOLOGY 49

CHAPTER IV: RELATED WORK **52**

4.1 SOFTWARE TESTING 52
4.1.1 Scenario-based Testing *52*
4.1.2 Equivalence Classes *54*
4.2 SOFTWARE RELIABILITY 55
4.2.1 Markov Chains *55*
4.2.2 Uncertainty Analysis *56*
4.3 RELIABILITY ASSESSMENT IN THE AS-TRM 58

CHAPTER V: MAPPING COSMIC-FFP TO ENTROPY **61**

5.1 INTRODUCTION 61

5.2 ANALYSIS OF SIMILARITIES AND DIFFERENCES ACROSS COSMIC-FFP AND ENTROPY MEASURES 64
5.3 THEORETICAL VALIDATION 69

CHAPTER VI: TESTING AND RELIABILITY PREDICTION APPROACH 78

6.1 SCENARIO-BASED TESTING ASSESSMENT IN COSMIC-FFP 78
6.1.1 Test-Case Generation 79
6.1.2 Partitioning into Equivalence Classes 80
6.1.3 Priority of Test Cases 83
6.1.4 Test Selection Algorithm 84
6.1.5 Case Study: Hotel Accommodation System (Reservation) 86
6.1.6 Test-Case Execution (Update Reservation) 102
6.2 ENTROPY-BASED RELIABILITY ASSESSMENT IN COSMIC-FFP 106
6.2.1 Markov Model and State Machine Diagrams 107
6.2.2 COSMIC-FFP and Sequence Diagrams 110
6.2.3 Analysis of Linkages across Models 112
6.2.4 Reliability Model for a Component-based System 119
6.2.5 Case Study: Railroad System 123

CHAPTER VII: FORMALIZING COSMIC-FFP IN THE AS-TRM 131

7.1 MAPPING BETWEEN COSMIC-FFP AND THE AS-TRM 131
7.2 CASE STUDY: STEAM BOILER 133

CHAPTER VIII: CONCLUSIONS AND FUTURE WORK 140

8.1 SUMMARY OF SIGNIFICANT RESULTS 140
8.2 FUTURE WORK 147

REFERENCES 150

ABBREVIATIONS 157

List of Figures

FIGURE 1: ALBRECHT 83 MODEL (IFPUG 2005) 21
FIGURE 2: GENERIC FLOW OF DATA ATTRIBUTES THROUGH SOFTWARE FROM A FUNCTIONAL PERSPECTIVE (ABRAN, DESHARNAIS ET AL. 2001) 27
FIGURE 3: GENERAL PROCEDURE FOR MEASURING SOFTWARE SIZE WITH THE COSMIC-FFP METHOD – ISO 19761 (ABRAN, ORMANDJIEVA ET AL. 2004) 29
FIGURE 4: EXAMPLE OF A NON DETERMINISTIC SYSTEM 31
FIGURE 5: ORACLE METHODOLOGY 33
FIGURE 6: THESIS METHODOLOGY 49
FIGURE 7: TESTING WITH UML 53
FIGURE 8: EXAMPLE OF AN EQUIVALENCE CLASS 54
FIGURE 9: SCENARIO EXAMPLE 62
FIGURE 10: CREATE RESERVATION SEQUENCE DIAGRAM 66
FIGURE 11: ENTROPY-BASED FUNCTIONAL COMPLEXITY MEASURE 69
FIGURE 12: MEASUREMENT PROCESS – DETAILED TOPOLOGY OF SUBCONCEPTS (SELLAMI AND ABRAN 2003) 73
FIGURE 13: TEST SET PARTITIONING STRATEGY 81
FIGURE 14: METRIC-BASED TEST CASE PARTITIONING ALGORITHM 82
FIGURE 15: HOTEL RESERVATION SYSTEM – USE-CASE DIAGRAM 88
FIGURE 16: CREATE RESERVATION SEQUENCE DIAGRAM 89
FIGURE 17: UPDATE RESERVATION SEQUENCE DIAGRAM 89
FIGURE 18: CONFIRM RESERVATION SEQUENCE DIAGRAM 90
FIGURE 19: ACCEPT RESERVATION SEQUENCE DIAGRAM 90
FIGURE 20: CANCEL RESERVATION SEQUENCE DIAGRAM 91
FIGURE 21: SELECT RESERVATION SEQUENCE DIAGRAM 91
FIGURE 22: ROOM TYPE REPORT SEQUENCE DIAGRAM 92
FIGURE 23: HOTEL RESERVATION SYSTEM REQUIREMENTS MANAGEMENT 103
FIGURE 24: RATIONAL TESTMANAGER FOR THE HOTEL RESERVATION SYSTEM 104
FIGURE 25: RESULTS OF AUTOMATIC FUNCTION TESTING PRODUCED BY ROBOT 105
FIGURE 26: RESULTS PRODUCED BY THE PROPOSED TESTING PROCEDURE 106
FIGURE 27: TRAIN STATE MACHINE 109
FIGURE 28: TRAIN STATE DIAGRAM WITH ITS TRANSITION PROBABILITIES P_{IJ} 110
FIGURE 29: TRAIN ENTERS CROSSING SEQUENCE DIAGRAM 112
FIGURE 30: TRAIN LEAVES CROSSING SEQUENCE DIAGRAM 112
FIGURE 31: DEPENDENCY DIAGRAM 116
FIGURE 32: INITIAL STATE-MACHINE DIAGRAM FROM FIGURE 29 117
FIGURE 33: INITIAL STATE-MACHINE DIAGRAM FROM FIGURE 30 117
FIGURE 34: CONTROLLER STATE-MACHINE DIAGRAM 118
FIGURE 35: GATE STATE-MACHINE DIAGRAM 118
FIGURE 36: CONTROLLER STATE DIAGRAM WITH ITS TRANSITION PROBABILITIES P_{IJ} 124
FIGURE 37: GATE STATE DIAGRAM WITH ITS TRANSITION PROBABILITIES P_{IJ} 124
FIGURE 38: SYNCHRONOUS PRODUCT OF TRAIN, CONTROLLER AND GATE 125
FIGURE 39: SYNCHRONOUS PRODUCT OF TRAIN AND CONTROLLER (SECOND CONFIGURATION) 128

FIGURE 40: SYNCHRONOUS PRODUCT OF TRAIN, CONTROLLER AND GATE (SECOND CONFIGURATION) 128
FIGURE 41: STEAM BOILER CONTROLLER 135
FIGURE 42: STEAM BOILER CONTROLLER AND ITS INTERFACE 136
FIGURE 43: CONTROLLER REACTIVE TASK (1) 137
FIGURE 44: CONTROLLER REACTIVE TASK (2) 137

List of Tables

TABLE 1: BASILI ET AL. FRAMEWORK (BASILI, SELBY ET AL. 1986) 10
TABLE 2: ALBRECHT 79 WEIGHTS 17
TABLE 3: ALBRECHT 79 (GSC).. 18
TABLE 4: ALBRECHT 83 18
TABLE 5: ALBRECHT 83 (GSC).. 19
TABLE 6: SIMILARITY BETWEEN COSMIC-FFP & ENTROPY-BASED FUNCTIONAL COMPLEXITY MEASURE CONCEPTS 67
TABLE 7: COSMIC-FFP CONCEPTS AND THEIR UML EQUIVALENTS................................ 78
TABLE 8: TEST-CASE DESCRIPTIONS 93
TABLE 9: DISTANCE CALCULATED BETWEEN T_2 AND THE REMAINING TEST CASES............ 94
TABLE 10: HOW TO CALCULATE THE DISSIMILARITY BETWEEN TWO TEST CASES 95
TABLE 11: DISTANCE CALCULATED BETWEEN T_1 AND THE REMAINING TEST CASES........... 96
TABLE 12: HOW TO CALCULATE THE DISSIMILARITY BETWEEN TWO TEST CASES 97
TABLE 13: DISTANCE CALCULATED BETWEEN T_3 AND THE REMAINING TEST CASES 97
TABLE 14: HOW TO CALCULATE THE DISSIMILARITY BETWEEN TWO TEST CASES 98
TABLE 15: DISTANCE CALCULATED BETWEEN T_4 AND THE REMAINING TEST CASES 99
TABLE 16: HOW TO CALCULATE THE DISSIMILARITY BETWEEN TWO TEST CASES 99
TABLE 17: DISTANCE CALCULATED BETWEEN T_6 AND THE REMAINING TEST CASES 100
TABLE 18: HOW TO CALCULATE THE DISSIMILARITY BETWEEN TWO TEST CASES 100
TABLE 19: DISTANCE CALCULATED BETWEEN T_5 AND THE REMAINING TEST CASES 100
TABLE 20: HOW TO CALCULATE THE DISSIMILARITY BETWEEN TWO TEST CASES 101
TABLE 21: THE FUNCTIONAL COMPLEXITIES FOR THE TEST CASES. 101
TABLE 22: TRANSITION MATRIX *P* FOR TRAIN OBJECT.. 110
TABLE 23: SIMILARITY BETWEEN COSMIC-FFP AND THE STATE MACHINE DIAGRAM CONCEPTS 114
TABLE 24: TRANSITION MATRIX P FOR THE CONTROLLER OBJECT.................................. 125
TABLE 25: TRANSITION MATRIX P FOR THE GATE OBJECT 125
TABLE 26: SYNCHRONOUS PRODUCT OF TRAIN, CONTROLLER AND GATE...................... 126
TABLE 27: TRANSITION MATRIX P FOR A SYNCHRONOUS PRODUCT 129
TABLE 28: MAPPING COSMIC-FFP CONCEPTS TO AS-TRM NOTATIONS 131
TABLE 29: TOTAL FUNCTIONAL SIZE FOR THE STEAM BOILER USING AS-TRM TERMS .. 138
TABLE 30: SUMMARY OF SIGNIFICANT RESULTS 144

CHAPTER I: INTRODUCTION

Software complexity is related to the size of the software and to the unpredictability (uncertainty) of its behavior. In the early phases of software development, we can quantify the size of software functionality from the functional requirement specifications. A functional size measurement (FSM) method, COSMIC-FFP, adopted in 2003 as the ISO/IEC 19761 standard, was developed by the Common Software Measurement International Consortium (COSMIC). It focuses on the "user view" of functional requirements and is applicable throughout the development life cycle. Some of the software systems targeted by COSMIC-FFP are large-scale and inherently complex, and feedback on their functional complexity would facilitate containment of that complexity throughout the software life cycle.

In this thesis, a new early quality assessment of reliability and scenario-based test adequacy in complex COSMIC-FFP models is proposed. The benefits of this work include earlier prediction of the functional complexity of software behavior in the COSMIC-FFP context, right from the requirements phase, as well as a mechanism for generating black-box test cases from the COSMIC-FFP model, test case prioritization and test set optimization within given budget constraints, and early prediction of reliability. The optimization includes partitioning the set of test cases generated from COSMIC-FFP scenarios into equivalence classes based on a testing distance criterion and test case prioritization. Test case prioritization is based on a new entropy-based measurement in the COSMIC-FFP context proposed for quantifying functional complexity in terms of the

uncertainty of software behavior described in the scenarios of software system usage. The reliability assessment proposed in this thesis uses Markov chains for predicting the reliability of the software prior to its implementation. The use of Markov chains requires a modeling of software behaviors with state diagrams, and therefore we develop here a new method for mapping COSMIC-FFP scenarios to state diagrams. Also, we present a study of the scales, units and scale types of both COSMIC-FFP and the Entropy-based Functional Complexity Measure that forms the basis of the assessment method that we are proposing. Previous studies have analyzed the scale types of a great deal of software, but not the concept of scale, nor how it is used in the design of a measurement method. Two case studies are introduced to demonstrate the applicability of these proposed methods: the Hotel Accommodation System and the Railroad System.

The definition of COSMIC-FFP is general and can be applied to any specification language. We include here a formalization of that definition in the AS- TRM context (Autonomic Systems Timed Reactive Object Model), a language for the formal design of autonomic reactive systems developed at Concordia University. A formal definition of the COSMIC-FFP method for AS-TRM specifications would allow functional complexity and functional size to be formalized during the specification construction process. The Steam Boiler case study is introduced to demonstrate the applicability of FSM in terms of AS-TRM formalization. Future work based on this thesis can include the development of AS-TRM specifications for several benchmark case studies, and the collection of COSMIC-FFP measurement data for both the theoretical and empirical validation of the proposed measurement.

This section introduces some issues and concepts related to software measurement, software quality, software size and software complexity that justify the importance of the proposed research in the current research effort with a view to obtaining feedback on software quality.

1.1 Motivation

Software measurement is one of the key technologies for controlling and managing the software development process. Measurement also forms the foundation of both the science and engineering, and much more research in software is needed to ensure the recognition of software engineering as a true engineering discipline.

Fenton and Pfleeger (Fenton and Pfleeger 1998) defined software measurement as the process of quantifying the attributes of software in order to characterize them according to clearly defined rules. The essential goal of software measurement is to identify anomalies during the development phase in which they originated, as well as to measure development progress. Thus, every software development phase should contain measures to ensure that high project visibility and quality control are achieved (Ormandjieva 2002).

Evaluation and prediction are two main applications of software measurement (Fenton and Pfleeger 1998). Evaluation assesses an existing software entity by numerically characterizing one or more of its qualitative attributes. Prediction, by contrast, forecasts the attributes of a future software entity using a mathematical model and associated prediction procedures. Whitmire (Whitmire 1997), fleshes out the meaning of evaluation,

subdividing it into estimation, assessment, comparison and investigation. The prediction measurement may be applied in the early phases of software development to forecast future characteristics of software entities.

Note that, in the large body of literature on software quality, very little has been written on FSM that can be of use for predicting quality early on in the process, especially when the software entity is large and complex. FSM has been used mostly for productivity, benchmarking and estimation purposes, but it can also be used very early in the software development life cycle, such as when measuring functional user requirements (FURs), which are known prior to the design, architectural, coding and testing phases. There are some hints, both in the literature and in practice, that FSM could also be used for quality purposes (in addition to productivity, benchmarking and effort estimation). Therefore, the motivation underlying this thesis is to improve methods for early measurement and prediction of software quality, specifically in a COSMIC-FFP context.

1.1.1 Software Quality

ISO 8402 defines quality as follows: "The totality of features and characteristics of a product or a service that bear on its ability to satisfy stated or implied needs." The measurement of software quality is aimed at predicting the level of quality of the software entity or to monitor the improvement of that quality during the software development process, or both. Software quality is classified in two categories: internal quality, which is measured purely in terms of the process, project, product or resource itself; and external quality, which is measured only with respect to how the process, project, product or resource relates to its environment (Fenton and Pfleeger 1998). This

thesis targets two external quality factors: the adequacy and the reliability of the test set. The internal criteria for the adequacy of scenario-based test cases are the functional complexity of a scenario and the distance between test cases. The adequacy of the test in terms of coverage is achieved by partitioning the test suite into equivalence classes based on the distance criteria, and prioritization of the test cases based on their functional complexity. The internal criteria are based on the certainty of the Markov system (using the entropy of a Markov chain).

1.1.2 Software Size and COSMIC-FFP

The word “size” in software measurement has two meanings: project size and software size. The first refers to total effort, estimated or actual, in work-hours or staff-months, for example. The second refers to the size of either the requirements (functions) or the deliverables, such as modules or lines of code.

Fenton and Pfleeger (Fenton and Pfleeger 1998) have identified three attributes of software size: length, complexity and functionality.

Length represents the physical size of the product, and is meaningful to technical staff. It is useful to measure the length of specifications, designs and codes. For example, a specification length may help in predicting a design length, which, in turn, may help in predicting a code length. Now, software size can be measured by counting the number of lines of code (LOC). LOC is useful in deciding how big a file is when the code needs to be stored. By contrast, it does not reveal anything about the software's functionality or

the quality of coding itself. Another reason why LOC is an inadequate measure is illustrated in the following example:

Select * from employee

This means that one line of code is required in SQL to bring up the list of employees; however, around 20 lines are needed in Java or C++, and around 500 lines are needed in COBOL to execute such a requirement. Moreover, a standard definition is needed to count the length of the software; for instance, how the blank lines, comment lines, data declarations, multi-line instructions and multi-instructions should be handled. Different counting techniques could yield different LOC values.

Software complexity, which is the second attribute of software size, is an essential characteristic of the software process/product, and constitutes a multifaceted notion that depends on the context (Fenton and Pfleeger 1998), (Whitmire 1997), (Henderson-Sellers 1996), (Zuse 1991), (Davis and Leblanc 1988). Similar to the classification given by Whitmire (Whitmire 1997), the complexity of a software system is viewed in different dimensions, namely the computational, the representational, the structural and the functional. Computational complexity quantifies the time and resources required to complete the process, and these are covered in the study of algorithmic efficiency. Representational complexity considers the tradeoffs between graphical and textual notations for unambiguous representations of the system model, system interactions and system behavior. Structural complexity is viewed in terms of coupling and cohesion, without considering the individual complexity of the components. Functional complexity

characterizes the dynamic performance of the system seen as a sequence of events required to fulfill system functionality.

Finally, functionality, the last attribute of software size, consists of the functions that are supplied by a product. Many software engineers argue that length is misleading and complexity is highly subjective; we believe, however, that the amount of functionality a product provides gives a better picture of product size. It is meaningful to management and it must be independent of the effort, the method and the technology. Moreover, functionality conveys an intuitive notion of the number of functions contained in a delivered product. Several approaches have been proposed for measuring the functionality of software products, such as:

- Albrecht's function points (Abran and Robillard 1994).
- The application point proposed in COCOMO 2.0 (Boehm 2002).
- DeMarco's specification weight (DeMarco 1982).
- COSMIC-FFP (Abran, Desharnais, Oligny, St-Pierre and Symons 2001).

All four approaches measure the functional size of software specification documents, but each can also be applied to software products later in their life cycle. There are some hints in the literature and in practice that FSM could be used for quality purposes as well (in addition to productivity, benchmarking and effort estimation). In this work, it is the innovative use of the COSMIC-FFP method, the ISO/IEC 19761 standard, that is of interest. We examine them here and theoretically validate them for their potential to contribute to the early assessment of software complexity and quality.

1.1.3 Software Complexity

Again, functional complexity characterizes the dynamic performance of a system seen as a sequence of events required to fulfill system functionality.

Information theory-based software measurement (Khoshgoftaar and Allen; Martin and England 1981) is used to quantify functional complexity in terms of an amount of information based on some abstraction of the interactions among software components (Ormandjieva 2002). However, what does information mean in this context? Shannon, the father of information theory, has stated that information causes change, and, if it doesn't, it is not information (Shannon, E., Weaver and Warren 1969). In other words, we say that we have gained information when we know something now that we didn't know before, when what we know has changed. Under the assumption that the complexity of a software product is associated with the information content of that product, the quantification of the amount of information will be used to assess the functional complexity of the software system and the required quality improvement (Alagar, Ormandjieva and Zheng 2000). Now, the average amount of information is quantified by the entropy of a set of events occurring in one usage of the software (Alagar, Ormandjieva et al. 2000).

Entropy is a concept in information theory which was introduced by C. E. Shannon (Shannon, C. E. et al. 1969) as a quantitative measurement of the uncertainty associated with a random phenomenon. It is said that one phenomenon represents less uncertainty than a second one if we are more confident about the result of experimentation associated with the first than we are about the result of experimentation associated with the second.

A random phenomenon can be described as a mathematical model, referred to as a probability space, designed to use mathematical reasoning to investigate questions about that phenomenon. For example, in throwing a die, the probability of 1, 2, 3, 4, 5 or 6 appearing is 1/6 for each. A great deal of uncertainty is associated with throwing a die, since the expected outcome of the experiment is uncertain. Considering any set of n events and their probability distribution $\{p_1, \ldots, p_n\}$, the quantification of this uncertainty quantity is calculated using the following entropy formula:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i \quad \ldots\ldots (1)$$

Here, we propose a new entropy-based measurement in the COSMIC-FFP context for quantifying the uncertainty of software behavior described in terms of the scenarios of software system usage. The expected benefits of this work include earlier prediction of the functional complexity of software behavior, right from the requirements phase, a mechanism for generating black-box test cases and their prioritization, as well as an early prediction of reliability.

1.2 Major Contributions and Thesis Outline

The major contributions of this thesis have been published in the following papers and journals: (Abran, Ormandjieva and Abu Talib 2004), (Abu Talib, Ormandjieva, Abran and Buglione 2005), (Abu Talib, Ormandjieva, Abran, Khelifi and Buglione 2006), (Abu Talib, Abran and Ormandjieva 2006), (Abu Talib, Ormandjieva and Abran 2007) and (Abu Talib, Abran and Ormandjieva 2005). In order to facilitate the introduction of such a large body of work, we have used the Basili et al. framework (Basili, Selby and

Hutchens 1986) to help in outlining the thesis work process, as well as to provide a classification scheme for understanding and evaluating the thesis work that has already been completed and published. A schematic representation of this framework is presented in Table 1.

Table 1: Basili et al. framework (Basili, Selby et al. 1986)

<table>
<tr><th colspan="6">I Definition</th></tr>
<tr><td>Motivation</td><td>Object</td><td colspan="2">Purpose</td><td colspan="2">Perspective Domain Scope</td></tr>
<tr><th colspan="6">II Planning</th></tr>
<tr><td colspan="2">Design</td><td colspan="2">Criteria</td><td colspan="2">Measurement</td></tr>
<tr><th colspan="6">III Operation</th></tr>
<tr><td colspan="2">Preparation</td><td colspan="2">Execution</td><td colspan="2">Data Analysis</td></tr>
<tr><th colspan="6">IV Interpretation</th></tr>
<tr><td colspan="3">Context of Interpretation</td><td colspan="3">Extrapolation</td></tr>
</table>

The framework is defined in terms of six components, namely motivation, object, purpose, perspective, domain and range. During the definition phase, an intuitive understanding of a high-level problem is developed into a precise specification that could contribute to its solution. “Motivation” identifies the high-level problem to be tackled. “Object” defines the principal entity being studied. “Purpose” is the explicit problem to be resolved. “Perspective” specifies from what point of view the explicit problem will be addressed.

Usually, an experiment in software engineering has two domains: team and project. Teams (comprising one or more members) work on software projects which attempt to resolve an issue, in terms of a software deliverable (manual, program and specifications). Four combinations of domains are possible: one team working on one project, many teams working on one project, one team working on many projects and a combination of many teams and projects.

Our thesis work was planned in detail in the second phase of the framework. During the design step, the case studies were selected. The direct and indirect criteria or factors that are related to the thesis' purpose were identified. Then, the measures designed to quantify these direct and indirect criteria were determined.

The thesis work itself is actually carried out during the third phase of the framework: Training might be required for the team that will be taking the measurements. Data are collected and validated during the execution of the case studies. These data are then analyzed using techniques chosen during the design step.

1.2.1 Definition

We present an exploratory study of related concepts across information theory-based measures and functional size measures, which was published in (Abran, Ormandjieva et al. 2004). Information theory-based software measurement has been used in the design of an entropy-based measure of functional complexity in terms of an amount of information based on some abstraction of the interactions among software components. As an FSM method, COSMIC-FFP, adopted in 2003 as the ISO/IEC 19761 standard, measures

software functionality in terms of the data movements across and within the software boundary. We explore some of the links between the two types of measures, and, in particular, the similarities (and differences) between their generic models of software functionality, their detailed model components taken into account in their respective measurement processes, and, finally, their measurement function. Also presented is an overview of some measurement concepts across COSMIC-FFP and our proposed Entropy-based Functional Complexity Measure. This overview validates three metrological properties (scale, unit and scale type) in both these measurement methods. The study for this work was published in (Abu Talib, Abran et al. 2005).

Investigations are also identified for extending the use of FSMs for scenario-based black-box testing and for reliability prediction purposes.

1.2.2 Planning and Operation

COSMIC-FFP focuses on the FURs of the software and is applicable throughout the development life cycle, from the requirements phase up to and including the implementation and maintenance phases. In this thesis, we extend the use of COSMIC-FFP for testing purposes by combining the functions measured by the COSMIC-FFP measurement method with the black box testing strategy. Our work here leverages the advantage of COSMIC-FFP, which is its applicability during the early development phase once the specifications have been documented, and also investigates the applicability of entropy measurement in terms of its use with COSMIC-FFP for assigning priorities to test cases. The criteria or factors related to such a study are the length of the test case, the length of the longest common prefix between two test cases, the test set, the

total number of events in a sequence, the number of occurrences of an event, the distance between two test cases, the similarity between two test cases, the dissimilarity between two test cases and, finally, the functional complexity. A case study of the Hotel Reservation System is applied in order to demonstrate the feasibility of the proposed testing strategy. Note that this testing strategy and the case study results have been also published in (Abu Talib, Ormandjieva et al. 2005), (Abu Talib, Ormandjieva et al. 2006).

Moreover, this thesis extends the architecture-based software reliability prediction model to the COSMIC-FFP context. The model is based on Markov chains and is applicable prior to implementation with the ability to build reliability models much earlier, at the requirements phase or based on the design specifications. In essence, each component of the system is modeled by a discrete time Markov chain. Then, a probabilistic analysis by Markov chains can be performed to evaluate the product's reliability in the early phases of software development and to improve the reliability process for large software systems. The criteria needed for this work are the transition matrix for each state diagram, the steady vector, the entropy for both the whole component and the object, and the reliability of both the component and the system. This approach of applying a Markov model in the COSMIC-FFP context is illustrated with the Railroad Crossing case study. The proposed reliability prediction approach and the results of the case study have been published in (Abu Talib, Abran et al. 2006; Abu Talib, Ormandjieva et al. 2007).

Finally, this thesis includes the starting points for the formalization of the COSMIC-FFP definition for the AS-TRM (Autonomic System Timed Reactive Object Model), a

language for the formal design of real-time reactive systems developed at Concordia University. A formal definition of the COSMIC-FFP method for the AS-TRM specifications would allow the formalization of functional complexity and functional size during the specification construction process. The mapping between COSMIC-FFP and AS-TRM terms has been explored, and the Steam Boiler case is the case study applied for estimating its functional size using AS-TRM terminology.

1.2.3 Interpretations

Both the proposed testing and the reliability prediction approaches have clearly stated steps and well-defined rules., The feasibility of the data obtained was demonstrated, in a spite of the difficulties usually encountered in attempting to do this. The proposed testing approach cannot test all the possibilities, but at least the maximum number of test cases that cover the most functionality of a system given budgetary constraints. The more test cases there are and the wider the variety of events, the greater the functional complexity. As for the reliability prediction approach, the higher the value of a reliability measure, the less uncertainty there is in the model, and thus the higher the level of software reliability. Theoretical validation is the proof of a valid scale, unit or scale type, and the case studies constitute the proof of concepts (i.e. demonstration of feasibility). By contrast, larger case studies can be taken into account in future work in order to see how scalable these approaches are for large amounts of data. In addition, more templates are required to capture information about the scenarios, which are sequences of events, and about the state diagrams, which visualize the Markov chain for each component in a system.

This thesis is organized as follows: Chapter 2 discusses software size, entropy measurement, software testing, software reliability, AS-TRM and the scale types in greater detail. Chapter 3 sets out the thesis objectives, results and benefits. It also introduces the methodology required as the starting point for this research. The related work that has been carried out in the testing and in the reliability prediction fields and AS-TRM are documented in chapter 4. In chapter 5, COSMIC-FFP is mapped to the measurement of complexity based on entropy, and their scale types, for COSMIC-FFP and functional complexity, are investigated. Chapter 6 contains more detail about the proposed testing and reliability approaches, as well as the case studies. The main keys to formalizing COSMIC-FFP in the AS-TRM with a case study are presented in chapter 7. We conclude and outline future work in chapter 8.

CHAPTER II: BACKGROUND

Software measurement is an essential activity of software development which allows for continuous feedback on the quality of products and processes during the software life cycle (Fenton and Pfleeger 1998).

Proposed here are new approaches for obtaining feedback on software functional complexity, and its applicability to testing and reliability assessments are investigated in the context of COSMIC-FFP, which was adopted in 2003 as the ISO/IEC 19761 standard (Abran, Desharnais et al. 2001), (ISO/IEC19761 2003), (ISO14143-1 1988). Before introducing the concepts related to COSMIC-FFP, it is important to present the state of the art of FSM. Entropy measurement in software engineering is also introduced in this chapter, as are the related concepts in software testing, software reliability, AS-TRM formalization and the scale types.

2.1 Functional Size Measurement: State of the Art

2.1.1 Function Points

The Function Point (FP) approach (Abran and Robillard 1994) is widely used in industry, specifically in Management Information System (MIS). It is considered better than LOC, since it reflects the requirements from the user's point of view and it measures software size from the specification stage (early in the life cycle). The FP approach provides a standardized method for measuring the various functions of a software application. Allan Albrecht of IBM developed Function Point Analysis (FPA) in 1979, and, in 1984, the

International Function Point Users Group (IFPUG) was set up to clarify FPA rules and set standards, and to promote their use and evolution.

The Albrecht 79 model (Abran and Robillard 1994) determines the Unadjusted Function Point Count (UFC) from the specifications, and involves four function types: files, inputs, outputs and inquiries, and one set of weight complexities. The UFC is the weighted sum of number of items of each type: UFC = Sum of (number of each item type * weight).

Table 2: Albrecht 79 weights

Function Types	Weights
Files	10
Inputs	4
Outputs	5
Inquiries	4

The FP count is, therefore, equal to UFC * [0.75 + 0.01 * TDI], where TDI is the Total Degree of Influence. As seen in Table 3, each system component is rated from 0 to 5, where 0 means irrelevant, 2 means moderate, 3 means average, 4 means significant and 5 means essential. The sum of previous influences constitutes the TDI.

Table 3: Albrecht 79 (GSC)

10 General System Characteristics
Backup
Data communications
Distributed processing
Performance issues
Heavily used configuration
Online data entry
Conversational data entry
Online update of master files
Complex functions
Internal processing complexity

Degree of Influence:
0. None
1. Incidental
2. Moderate
3. Average
4. Significant
5. Essential

In 1983, Albrecht and Gaffney expanded the model to five function types, three sets of weights (Table 4) and 14 GSCs (Table 5).

Table 4: Albrecht 83

Function Types	Weights		
	Low	Average	High
Internal logical files	7	10	15
External interfaces files	5	7	10
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6

Table 5: Albrecht 83 (GSC)

14 General System Characteristics	
1) Reusability	9) Complex Functions
2) Data communications	10) Internal processing complexity
3) Distributed processing	11) Installation ease
4) Performance issues	12) Operational ease
5) Heavily used configuration	13) Multiple sites
6) Online data entry	14) Facilitate change
7) Conversational data entry	
8) Online update of master files	

The formula now becomes: FP = UFC * [0.65 + 0.01 * TDI]. The detailed explanations of the five function types are taken from (IFPUG 2005), and they are as follows:

External Inputs (EI): EI is an elementary process in which data cross the boundary from outside to inside. The data come from a system "actor". The actor can add, change and delete information on an internal logical file. The data can be either control information or business information. If the data are control information, then the actor does not have to maintain an internal logical file. Examples of external inputs are file names and menu selection commands.

External Outputs (EO): EO is an elementary process in which derived data pass across the boundary from inside to outside. The data are sent to a system actor. In addition, the

system actor may update an ILF. The data create reports or output files which are sent to other actors. These reports and files are created from information contained in one or more internal logical files and/or external interface files. Reports and messages are examples of external outputs.

External Inquiry (EQ): EQ is an elementary process with both input and output components where an act or retrieves data from one or more internal logical files and/or external interface files. The input process does not update or maintain any FTRs (Internal Logical Files or External Interface Files) and the output side does not contain derived data.

Internal Logical Files (ILF): An ILF is a user-identifiable group of logically related data that resides entirely within the application boundary and is maintained through External Inputs.

External Interface Files (EIF): An EIF is a user-identifiable group of logically related data that is used for reference purposes only. The data reside entirely outside the application boundary and are maintained by other applications' external inputs. The EIF is an internal logical file for another application. The primary difference between an EIF and ILF is that an EIF is maintained by another application.

The complete picture of the Albrecht 83 model is shown in Figure 1.

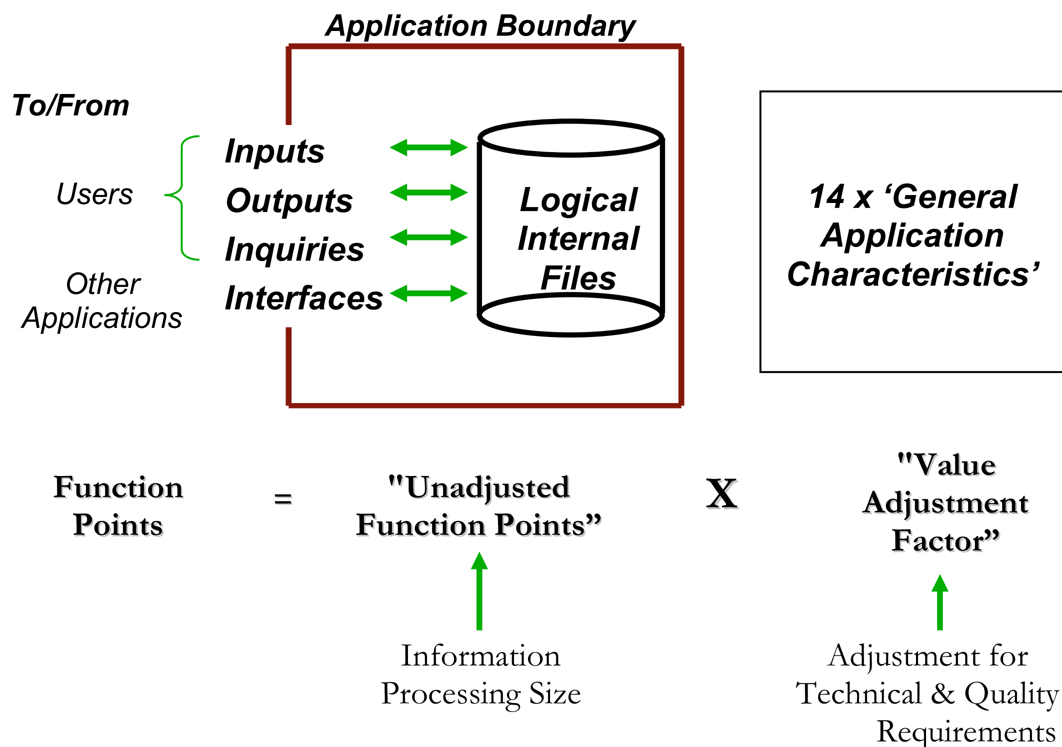

Figure 1: Albrecht 83 Model (IFPUG 2005)

The FP method has some limitations, such as subjectivity in the technology factor. As well, researchers (Abran and Robillard 1994) have shown that the value adjustment factor does not improve the accuracy relative to the UFC. Moreover, FP is suitable for measuring the functionality aspects of software size but not the complexity aspects. FP is also found to be effective in functionality-intensive applications (e.g. data processing), but not so effective in algorithmically complex applications (e.g. compilers). Another factor is the subjectivity in the assignment of weights. FP weights were determined subjectively from IBM experience, and may not necessarily be applicable to other environments. There is also the issue of double counting. Internal complexity is counted in assigning the UFC weight, as is the case in the TDI. Problems with the measurement

theory could constitute other limitations, in that measurements from different scales could be incorrectly combined. For example, weights and TDI ratings are expressed on the ordinal scale, while point counts are expressed on the absolute scale (or at least on a ratio scale). Linear combinations of the two are meaningless (Abran and Robillard 1994).

Even with its limitations, FP can be more useful than software length, if FP is used with care. There are also several variations of FP counting that have been proposed, such as Mark 2 FPA, among others.

2.1.2 ISO Standards

In this section, other ISO standards regarding the measurement of functionality are discussed. NESMA (Netherlands Software Metrics Users Association) and MK2 (Mark 2 FPA) have proposed almost the same concepts and terms, as well as the same rules and guidelines within FPA. There are some differences, however, which are mainly taken from (NESMA 2004), (Symons 1999) and (Committee 1998).

“Both the NESMA (NESMA CPM 2.0) and the IFPUG (IFPUG CPM 4.1) now use the same philosophy, the same concepts and terms, and the same rules and guidelines within FPA” (NESMA, 2003). More precisely, this statement means that both groups count the functions that users can identify, and these functions are of the same five types: external input, external output, external inquiry, internal logical file and external interface file. NESMA uses the same complexity matrices and unadjusted function point table to value the complexity of functions. It also uses the same 14 general system characteristics with almost the same valuation criteria, determining the unadjusted function point count, the

value adjustment factor, the adjusted function point count, the application function point count and the project function point count in the same way that IFPUG does.

By contrast, in 1996, at the request of NESMA members, the organization published operational guidelines on complex counting issues to help counters, where IFPUG has not provided specific guidelines on these issues. For example, an output generated by a non-unique identifying selection criterion is counted as an external output by NESMA, but as an external inquiry by IFPUG the point when no further data processing has to be done. This difference does not have a major influence on the number of function points in an application or project, because it does not influence the quantity of identifiable functions, only the type of function; i.e. an external inquiry or external output. Often the data to be changed or deleted in an application is first displayed before it is actually changed or deleted. The act of displaying this data is known as “an implicit inquiry”. IFPUG counts one extra external inquiry for this when the display is a distinct user requirement. NESMA counts an extra external inquiry for this when the user has required a specific query function for which the primary function is to transfer information to the user. In other cases, NESMA considers the display to be a part of the change and/or delete function, and counts the data presented only as additional data element types for that function. In practice, this difference does not have much influence on the number of function points in an application or project, because this same display will also often have been specified as an independent external inquiry (and will therefore have been counted already).

As a result, it can be said that there are a few differences between NESMA and IFPUG in terms of the number of function points in an application or project, and these have only a negligible effect on that number. Moreover, they have resolved almost all their differences in regard to how the complexity of functions should be established and how to determine the number of data element types and file types referenced.

All the requirements or user functionalities are introduced in terms of "Logical Transactions" (LT) in the MK2 method. An LT comprises an input component, some processing and an output component, and is defined as being triggered by an event in the real world of interest to the user, or a request for information. MK2 takes the size of the input and output components of an LT to be proportional to the number of DETs on the component. However, the size of the processing component is taken to be proportional to the number of entity types. On the basis of these two weighted counts, an MK2 FP size is given to each LT.

The MK2 FP size scale was designed to be more sensitive to small changes in functionality than the IFPUG scale, and to be more sensitive to variations in the internal processing complexity of the world of MIS. "The MK II method works at a much finer level of granularity than the IFPUG method and this leads to lower sizes for small enhancements" (Symons, 1999). The major difference is that MK2, with its granularity, is a continuous measure, whereas IFPUG limits component size once a threshold is reached. As the concepts on which the size measure is based are logical transactions and entities,

the MK2 functional size measure should be independent of the technology or methods used to develop the software.

The weightings introduced by Symons are designed to deliver a size scale of similar magnitude for the MK2 method as for the IFPUG method. On average, therefore, the methods give roughly the same software sizes up to around 400 function points. For larger sizes, MK2 FPA tends to produce increasingly larger sizes than the IFPUG method. For some purposes, portfolio management for instance, the methods may be regarded as equivalent. However, for the most common purposes of performance measurement and estimating, it is preferable to use one scale or the other consistently, only converting from one to another if needed, using a formula which shows the average relationship.

2.1.3 COSMIC-FFP

The FSM method developed by the Common Software Measurement International Consortium (COSMIC) has now been adopted as an international standard (ISO 19761 (ISO/IEC19761 2003)) and is referred to as the COSMIC-FFP method (Abran, Desharnais et al. 2001). This measurement method has been designed to measure the functional size of management information systems, real-time software and multi-layer systems. Its design conforms to all ISO requirements (ISO 14143-1 (ISO14143-1 1988)) for FSM methods, and was developed to address some of the major weaknesses of earlier methods, like FPA (Abran and Robillard 1994), the design of which dates back almost 30 years, to a time when software was much smaller and much less varied. COSMIC-FFP focuses on the "user view" of functional requirements and is applicable throughout the

development life cycle, right from the requirements phase to the implementation and maintenance phases. Before starting to measure using the COSMIC-FFP method, it is imperative to carefully define the purpose, the scope and the measurement viewpoint. This may be considered as the first step of the measurement process. The measurer defines why the measurement is being undertaken, and/or what the result will be, as well as the set of FURs to be included in a specific FSM exercise. Measurements taken using the COSMIC-FFP method with a different purpose and scope and a different measurement viewpoint may therefore give quite a different size.

In the measurement of software functional size using the COSMIC-FFP method, the software functional processes and their triggering events must be identified (Abran, Desharnais et al. 2001), (ISO/IEC19761 2003). In COSMIC-FFP, the unit of measurement is a data movement, which is a base functional component that moves one or more data attributes belonging to a single data group. Data movements can be of four types: Entry, Exit, Read or Write. The functional process is an elementary component of a set of user requirements triggered by one or more triggering events either directly or indirectly via an actor. It comprises at least two data movement types: an Entry plus at least either an Exit or a Write. The triggering event is an event occurring outside the boundary of the measured software and initiates one or more functional processes. The subprocesses of each functional process are sequences of events. An Entry moves a data group, which is a set of data attributes, from a user across the boundary into the functional process, while an Exit moves a data group from a functional process across the boundary to the user requiring it. A Write moves a data group lying inside the functional

process to persistent storage, and a Read moves a data group from persistent storage to the functional process. See Figure 2 for an illustration of the generic flow of data attributes through software from a functional perspective.

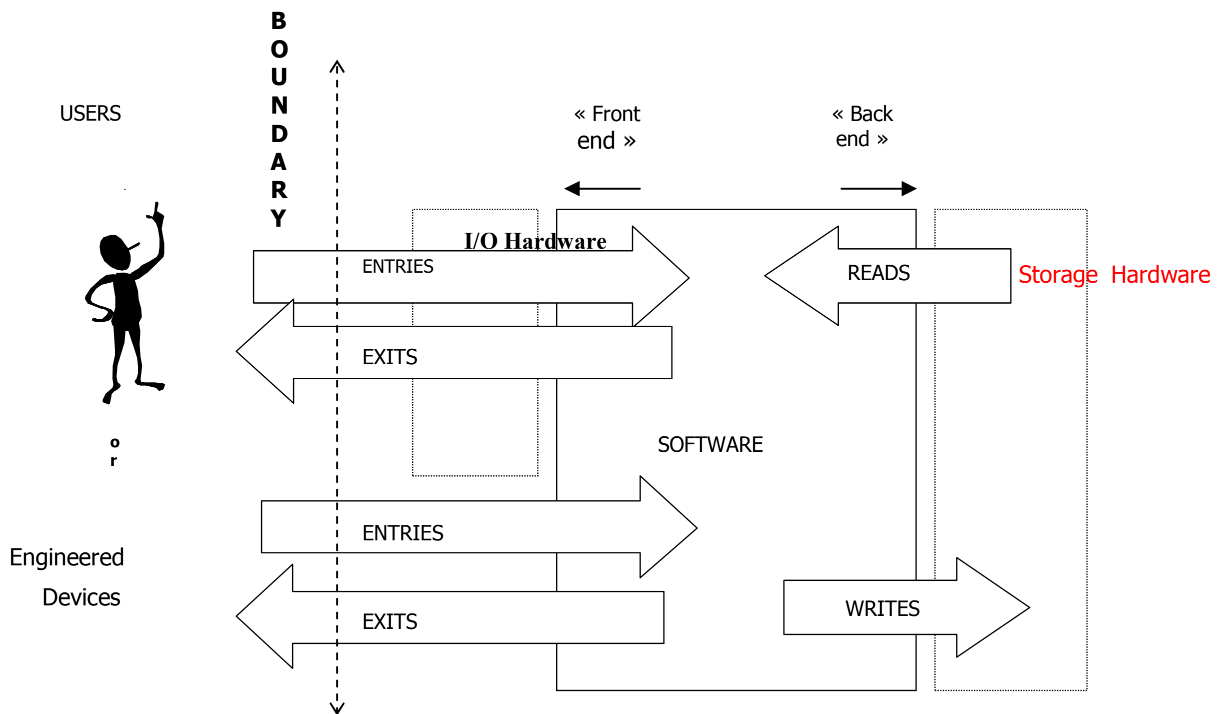

Figure 2: Generic flow of data attributes through software from a functional perspective (Abran, Desharnais et al. 2001)

A general procedure for measuring software functional size with the COSMIC-FFP method is proposed here, as in Figure 3. The measurement process is performed in five steps.

First, the boundary of the software to be measured is identified by the measurer based on the requirements and the specifications of the interaction between the hardware and the software. Second, the measurer identifies all possible functional processes, triggering events and data groups from the requirements. These are considered as candidate items at this stage. Third, the candidate items (i.e. functional processes, triggering events and data groups) are mapped into the COSMIC-FFP software context model (Figure 3) based on the COSMIC-FFP rules. In this mapping, each functional process must be associated with a triggering event and to the data group(s) manipulated by it. This mapping also allows the identification of layers. Fourth, the COSMIC-FFP subprocesses (i.e. data movements of the following types: Entry, Exit, Read and Write) are identified within each functional process. The COSMIC-FFP measurement function is applied to the subprocesses identified to determine their respective COSMIC-FFP Cfsu size measure. Finally, the measurer computes an aggregate of the measurement results to obtain the total functional size of the software being measured.

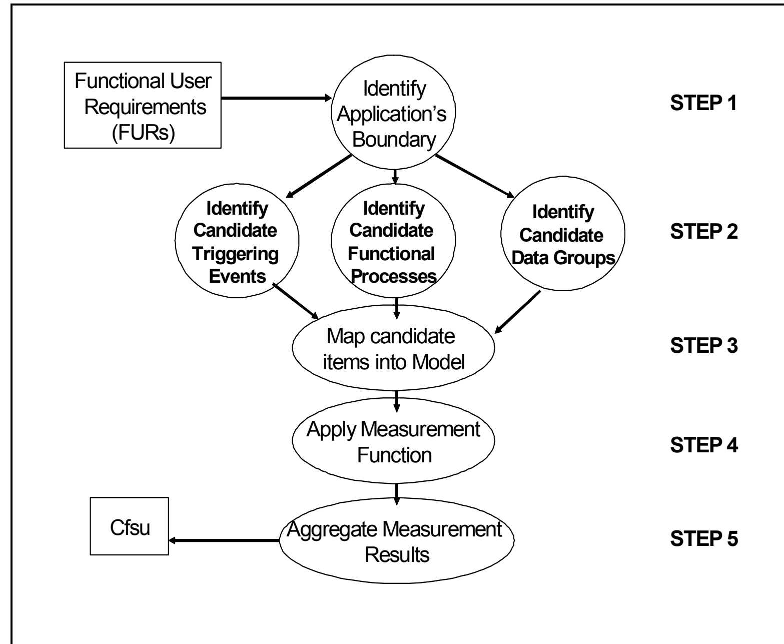

Figure 3: General procedure for measuring software size with the COSMIC-FFP method – ISO 19761 (Abran, Ormandjieva et al. 2004)

2.2 Entropy Measurement in Software Engineering

R. Hamming has introduced the concept of entropy into software engineering as a measure of the average rate at which information is conveyed in a message or language (Peters and Pedrycz 2000). A message means a string of symbols drawn from an alphabet of symbols $s_1, \ldots, s_q$ The field of information theory deals with the measure of the amount of information contained in a message (Hamming 1980). Information, in this context, is something that is not already known; that is, more information is gained when a symbol occurs where it is *not* expected than if it occurs where it *is* expected. Rate, in this context, means the frequency of occurrence of each symbol (Hamming 1980).

Thus, the amount of information conveyed by a single symbol in a message is related to its probability of occurring: $I_i = -\log_2 p_i$ …… (2).

Hartley (1928) was the first to propose the use of logarithms in this connection. The logarithm guarantees that the amount of information increases as the number of symbols increases.

The accumulation of information is additive (Hamming 1980); that is, the amount of information conveyed by two symbols is the sum of their individual information content. It follows, then, that an entire alphabet of symbols $s_1, \ldots, s_q$ would, on average, provide the amount of information calculated in formula (2), with the bit as the unit of information per symbol. It can be shown that the maximum amount of information per symbol is provided by an alphabet with symbols that all occur with equal probability. The average amount of information conveyed by each symbol in such an alphabet is for an alphabet having $\log_2 s_q$ symbols, each with an equal probability of occurring. The minimum amount of information is conveyed by an alphabet in which one symbol occurs with a probability of one, and all others occur with a probability of zero. Such an alphabet is said to have a language entropy of zero.

2.3 Software Testing

Testing represents a major effort within the whole of the software development life cycle. The Guide to the Software Engineering Body of Knowledge (SWEBOK) (Bertolino 2004) provides an overview, including references, of the basic and generally accepted notions underlying the Software Testing Knowledge Area. It describes testing as an activity performed to evaluate product quality, and to improve it by identifying defects and problems. The definition of testing provided in (Bertolino 2004) is as follows:

"Software testing consists of the dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the specified expected behavior."

The underlined terms are key concerns in software testing, and we must explain them briefly here prior to introducing COSMIC-FFP (Abran, Desharnais et al. 2001) into the testing context.

The term dynamic implies that, when we want to test a program, we can execute it with differently valued inputs. The valued input not only means the input value alone, but also the specified input state. The input value alone is not sufficient to determine the outcome of a test. For example, a nondeterministic system may react to the same input with different behaviors, depending on the system state. The nondeterministic system described in Figure 4 may go from state S_2 to either state S_3 or S_4 while reading *a* as an input value. The input state is also necessary in order for the system to decide where to go. However, this is a design issue and outside the scope of this thesis.

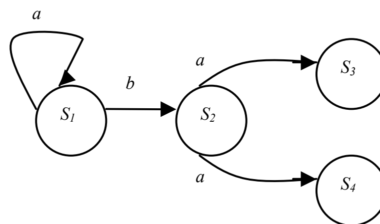

Figure 4: Example of a Non Deterministic System

Finite means having a test set (which includes test cases) while testing the system. In practice, an exhaustive test set can generally be considered infinite, even in simple

programs. For example, a small program comparing two integers and returning the smaller number may require that the set of integers constitute the infinite test set. This is what makes testing a long and expensive process. Testing implies a tradeoff between limited resources and schedules, and inherently unlimited test requirements. As a result, we need a finite test set with which enough testing is conducted to obtain reasonable assurance of acceptable behavior.

The term selected refers to the way in which the finite test set has been chosen. The most difficult problem in generating test cases is finding a test selection strategy that is both valid and reliable (Chow 1978). The power of a test case generation technique for detecting faults in an implementation is referred to as fault coverage (En-nouaary, Dssouli and Khendek 2002). Many different test methods exist (e.g. formal methods based on Finite-State Machines and Extended Finite State Machines (Beizer 1990)), which are all assumed to generate test suites containing test cases especially likely to reveal failures. These test methods can be compared according to their respective fault coverage. One method is considered more powerful than another if it has better fault coverage.

Finally, to make the testing process useful, it must be possible, even if not always easy, to decide whether or not the observed outcomes or observed outputs of program execution are acceptable. This describes the term expected in the testing definition. It must be possible to determine whether or not the observed behavior is in conformity with user expectations, specifications, anticipated behavior requirements or reasonable

expectations. The test pass/fail decision is, in the testing literature, commonly referred to as the “oracle problem” (Bertolino 2004), (Beizer 1990) (see Figure 5).

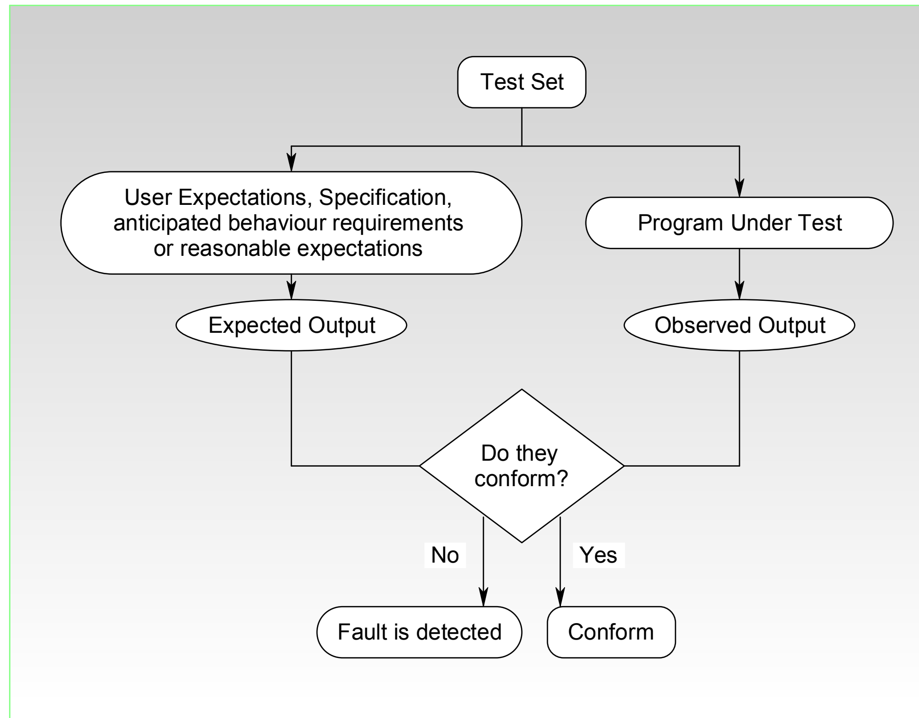

Figure 5: Oracle methodology

In our work here, the use of the COSMIC-FFP method has been investigated in the context of black-box use-case-driven testing using the Oracle methodology. Such testing is a testing methodology at the system level, where the scenarios depict the sequence of executions of the system, and the test cases can be derived from the use-case model and its corresponding UML diagrams. It is called “black-box testing” because the structure of the implementation is not known, and the test cases are generated and executed from the specification of the required functionality at defined interfaces (use-case model, in our case).

2.4 Software Reliability

Software reliability is the probability of failure-free software operation for a specified period of time in a specified environment (Engineers 1991). Reliability is expressed on a scale of 0 to 1. A system that is highly reliable will have a reliability measure close to 1, while an unreliable system will have a reliability measure close to 0.

Software reliability has many characteristics (Peters and Pedrycz 2000), and software failures are primarily due to design faults. Repairs are made by modifying the design to make it robust in conditions that can trigger a failure. There are no wear-out phenomena associated with software reliability, because software errors occur without warning. Also, so-called “old” code can exhibit an increasing failure rate as a function of errors induced during an upgrade. Moreover, while external environmental conditions do not affect software reliability, internal environmental conditions, such as insufficient memory or inappropriate clock speeds, do affect it. An important characteristic of software reliability is that it is not time-dependent. Failures occur when the logic path that contains an error is executed. Reliability growth is observed as errors are detected and corrected.

Software reliability measurement consists mainly of two activities. One is reliability estimation, an activity in which current software reliability is evaluated by applying statistical inference techniques to failure data obtained during system test or system operation. Also determined is whether or not a reliability model is, in retrospect, a good fit. The other is reliability prediction, an activity in which future software reliability is predicted based upon available software measurement data.

Traditional software reliability engineering assumes usage-based statistical testing under the guidance of operational profiles that characterize usage patterns. Current software reliability models are applicable when the code is generated and is being tested, and apply statistical inference procedures to failure data taken from software testing and operation to determine whether or not a reliability model is, in retrospect, a good fit (Fenton and Pfleeger 1998).

2.4.1 Markov Chains

Reliability modeling has two classification schemes: the Musa, and Okumoto classification scheme (Musa and Okumoto 1982) and the Hoang Pham classification scheme (Pham 1999). The latter has two reliability models, a deterministic model and a probabilistic model. The deterministic model is used to study the number of distinct operators and operands in a program, as well as the number of errors and the number of machine instructions in the program. The probabilistic model represents the failure occurrences and the fault removals as probabilistic events, which are subdivided into six groups: error-seeding, failure rate, curve-fitting, reliability growth, Markov Chains and the Non-homogeneous Poisson Process.

The Markov model (Ormandjieva 2002, Strook 2005 and Trvedi 1975) is a very powerful tool with which scientists and engineers can analyze and predict the behaviors of a complex system.

A Markov model analysis can yield a variety of useful performance measures describing the operation of the system, including the following:

- System reliability;
- Availability;
- Mean time to failure (MTTF);
- Mean time between failures (MTBF);
- The probability of being in a given state at a given time;
- The probability of repairing the system within a given time period (maintainability);
- The average number of visits to a given state within a given time period.

There is interest in using Markov models for software reliability prediction purposes as well, since:

- Environmental laws are considered as random and not controlled by system laws;
- Being in a particular state, a system may choose to execute any of the transitions available in that state in order to move to another state.

A Markov process is a stochastic one, with two main characteristics:

- It can take on a finite number of possible states, which we will index by the non-negative integers: 0, 1 ….
- It has what is known as the "Markovian" property: the probability distribution of future states of the process depends only on the current state, and is conditionally independent of past states (the path of the process).

In other words, a Markov system can be in one of several mutually exclusive states, and can move from one state to another according to the fixed probabilities. For example, if a Markov system is in state S_i, there is a fixed probability p_{ij} of it moving into state S_j at the

next time step. Therefore, the transition matrix is defined as matrix P, the ij^{th} entry of which is p_{ij}, and the entries in each row add up to 1.

Markov processes have many applications in the environmental sciences and in management. A simple example of a Markov model is the weather model (Wikipedia Encyclopedia). Given the weather state on the preceding day, we can represent the weather probabilities for today by a transition matrix:

$$P = \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix}$$

The weather on day 0 is known to be sunny. This is represented by a vector in which the "sunny" entry is 100%, and the "rainy" entry is 0%:

$$\mathbf{x}^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

The weather on day 1 can be predicted by:

$$\mathbf{x}^{(1)} = P\mathbf{x}^{(0)} = \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$$

Thus, there is a 90% chance that day 1 will also be sunny. The weather on day 2 can be predicted in the same way:

$$\mathbf{x}^{(2)} = P\mathbf{x}^{(1)} = P^2\mathbf{x}^{(0)} = \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix}^2 \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.86 \\ 0.14 \end{bmatrix}$$

Now, we need to calculate the steady vector that represents the probabilities of sunny and rainy weather on all days, and is independent of the initial weather. The steady-state vector is defined as:

$$\mathbf{q} = \lim_{n\to\infty} \mathbf{x}^{(n)}$$

Since **q** is independent of initial conditions, it must remain unchanged when transformed by *P*. This makes it an eigenvector (with eigenvalue 1), and means it can be derived from *P* as follows:

$$\begin{aligned}
P &= \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \\
P\mathbf{q} &= \mathbf{q} && (\mathbf{q} \text{ is unchanged by } P.) \\
&= I\mathbf{q} \\
(I - P)\mathbf{q} &= \mathbf{0} \\
&= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} 0.9 & 0.5 \\ 0.1 & 0.5 \end{bmatrix} \right) \mathbf{q} \\
&= \begin{bmatrix} 0.1 & -0.5 \\ -0.1 & 0.5 \end{bmatrix} \mathbf{q}
\end{aligned}$$

- Set $s = q2$, so $5\ s = q1$.
- We want $s + 5\ s = 1$, therefore $s = 0.167$.
- The steady-state vector is:

$$\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} 5s \\ s \end{bmatrix} = \begin{bmatrix} 0.833 \\ 0.167 \end{bmatrix}$$

In our work here, we are extending the use of the COSMIC-FFP method for reliability prediction purposes in the context of the Markov Model. This can be done once the similarity between COSMIC-FFP and Markov Model concepts has been explored.

2.5 AS-TRM

Another objective of this thesis is to contribute to the success of autonomic systems research through a rigorous investigation of an architectural approach to developing and evolving autonomic (self-managing) systems, and for the continuous monitoring of their quality. The automation of system self-management requires a solid formal foundation

for system development, including integration of the non-functional requirements into the development process, as described in our work here.

The Timed Reactive Object Model (TROM) formalism created at Concordia University has the required expressive power for specifying autonomic elements. The formalism has to be extended to include specification of the autonomic system architecture, system configuration and self-monitoring within a single formal framework, and to integrate the non-functional requirements (NFRs) for quality through the usage and structure specification of their mapping to system. Both the TROM and the AS-TRM are briefly described below.

The TROM formalism is a three-tier formal model (Alagar, Achuthan and Muthiayen 1996). As a layered model, each tier communicates only with the tier immediately above it. The independence of the tiers makes modularity, reuse, encapsulation and hierarchical decomposition possible. The three-tier structure describes the system configuration, AS components and their interaction, reactive classes and the relative Abstract Data Type. The uppermost tier is the system architecture specification. The third tier is the AS-Component configuration specification. It specifies the configuration of reactive objects and their collaboration, as well as the port links, which regulate the communication tunnels between objects. The second tier constitutes the TROM class, which is a Generic Reactive Class and is included in the subsystem. The TROM class is also a hierarchical finite-state machine augmented with ports, attributes, logical assertions on the attributes and time constraints. The lowest tier is the Larch Shared Language (LSL) trait, which represents the Abstract Data Type used in the TROM classes.

The TROMLAB (Timed Reactive Object Model Lab) is a framework that provides an environment for rigorous development of real-time reactive systems (Alagar, Achuthan et al. 1996). It is based on the TROM, which is a formalism based on the object-oriented and real-time technologies.

The TROMLAB contains a number of tools to assist in the development of real-time reactive systems. These tools provide an automatic mechanism for collecting and analyzing quality measurement data. The current architecture of the TROMLAB consists of the following components:

- Rose-GRC Translator (Popistas 1999): a module which maps the graphical model in Rational Rose to formal specifications based on the TROM;

- Interpreter (Tao 1996): a module which syntactically performs verification on specifications and generates an internal representation of them;

- Simulator (Muthiayen 1996): a module which animates a subsystem based on an internal representation generated by the interpreter;

- Browser for Reuse (Muthiayen 2000): an interface to a library which helps the user navigate, query and access system components during development;

- Graphical User Interface (Srinivasan 1999): an interface for the system developer to interact with the TROMLAB environment;

- Reasoning System (Haidar 1999): a debugging facility that allows the user to query system behavior based on interactive queries;

- Verification Assistant (Pompeo 1999): an automated tool which extracts mechanized axioms from real-time reactive systems;

- Test Case Generator (Zheng 2002 and Chen 2002): an automated tool for generating test cases from specifications.

The process model in TROMLAB (Alagar, Achuthan et al. 1996) supports the iterative development approach, which provides the following benefits:

- Reduces risks by exposing them early in the development process.
- Gives importance to the architecture of the system's configuration.
- Designs modules for large-scale software reuses.

AS-TRM extends the TROM formalism (Vassev, Kuang, Ormandjieva and Paquet 2006) by adding more tiers and including the following specifications:

- A timed reactive autonomic component (AC);
- A group of synchronously interacting ACs (ACG);
- An autonomic system (AS), consisting of asynchronously communicating ACGs.

The autonomic (self-monitoring) behavior is implemented in all upper tiers within the formal framework, as described below.

The AC is a newly added tier that encapsulates TROM objects (the TROM formalism's second tier) into an AS-TRM autonomic component (AC). The synchronous interaction among the ACs allows a reactive task to be performed. The communication between an AC and its upper-tier ACG is realized through an interface and is asynchronous.

An AS-TRM Component Group is a set of synchronously communicating ACs cooperating in the completion of a group task. Each ACG can independently accomplish

a complete real-time reactive task. The self-monitoring behavior in the ACG tier and the asynchronous interaction between an ACG and the ACs is realized by an ACG Manager (AGM).

The Autonomic System (AS) tier abstracts a set of asynchronously communicating ACGs. The self-monitoring behavior and the asynchronous interaction between an AS and the ACGs is realized by the Global Manager (GM).

2.6 Scale Types

In measurement theory, the meaning of numbers is characterized by scale types (Fenton and Pfleeger 1998, Whitmire 1997, Zuse 1991 and ISO 2002), but the theory does not directly address the concept of scale, as typically defined in metrology. A scale is defined as a set of ordered values, continuous or discrete, or a set of categories to which the attribute is mapped (ISO 2002), whereas a scale type depends on the nature of the relationship between the values on the scale (ISO 1993).

In a mathematical representation, a scale is defined by $<E, N, \Phi>$, where E is the empirical structure, N is the numerical structure and Φ is the mapping between them (Zuse 1991). By contrast, a scale type is always defined by admissible transformations, and relationships between mappings are described in terms of transformations (Whitmire 1997). There are five major scale types: nominal, ordinal, interval, ratio and absolute, and they can be seen as describing certain empirical knowledge behind the numbers (Whitmire 1997). Knowing the characteristics of each type helps in interpreting the

measures (Fenton and Pfleeger 1998). In the following subsections, the scale types are summarized.

Nominal Scale Type:

Nominal-scale measurement places elements in a classification scheme (Fenton and Pfleeger 1998; Whitmire 1997). The classification partitions the set of empirical entities into equivalence classes with respect to a certain attribute. Two entities are considered to be equivalent, and therefore to belong to the same equivalence class, if and only if the same attribute of both is being measured. Empirical classes are exhaustive and mutually exclusive. The classes are not ordered because of a lack of empirical knowledge about relationships among the classes. In nominal-scale measurement, each empirical class might be represented by a unique number or symbol, and the only mathematical operation allowed in the nominal scale type is "=". The only admissible transformation is one-to-one mapping that preserves the partitioning. An example of a nominal scale type could be the assignment of numbers to the football team players where you cannot say that the player with number one is, for example, the best or the shortest on his team, since assigning the number one in such a case serves only to distinguish that player from the others on the team.

Ordinal Scale Type:

The ordinal scale forms the basis of software measurement, and all other extended measurement structures are based on it (Zuse 1991). The ordinal scale assigns numbers or symbols to objects so that they can be ranked and ordered with respect to an attribute

(Whitmire 1997). The main characteristic of the ordinal scale is that the numbers represent a ranking only, and so addition, subtraction and other arithmetic operations have no meaning. Also, any mapping that preserves the ordering is acceptable as an ordinal scale (Fenton and Pfleeger 1998). For example, data loss failures are more critical than incorrect output failures, and therefore 10 could be assigned for the data loss failures and 5 for the incorrect output failures. In this case, such failures can be ordered according to how critical they are where assigning these numbers will preserve the same meaning.

Interval Scale Type:

The interval scale is useful for augmenting the ordinal scale with information about the size of the intervals that separate the classes. That is, the difference in units between any two of the ordered classes in the range of the mapping is known, but computing the ratio of two classes in the range does not make sense. This scale type preserves order, as with an ordinal scale; however, in an interval scale, addition and subtraction are acceptable operations. However, multiplication and division are not (Fenton and Pfleeger 1998). For instance, the Celsius and Fahrenheit scales for temperature have no meaning for ratios.

Ratio Scale Type:

A ratio scale is an interval scale on which there exists an absolute zero. This zero element represents the smallest scale value, where an object has a null amount of the attribute. Therefore, the measurement mapping in a ratio scale must start at zero and increase by equal intervals, known as units. All arithmetic in a ratio scale can be meaningfully

applied to the classes in the range of the mapping (Fenton and Pfleeger 1998). Time interval and length are good examples of ratio scales.

Absolute Scale Type:

The absolute scale represents counts of objects in a specific class. There is only one possible measurement mapping, namely the actual count, and it has a unique unit. As with the ratio scale, all arithmetic analysis of the resulting count is meaningful (Fenton and Pfleeger 1998).

More detail about these scale types can be found in (Fenton and Pfleeger 1998), (Whitmire 1997) and (Zuse 1991).

Once the main concepts and definitions of software size, entropy measurement, software testing and reliability, AS-TRM and scale types had been introduced in the software engineering literature, chapter 3 discusses the proposed thesis and its methodology in terms of how to investigate the candidate linkages, and candidate contributions, across the two fields, COSMIC-FFP and FC, in order to extend COSMIC-FFP for purposes such as testing and reliability assessment, and for formalizing it in the AS-TRM context.

CHAPTER III: PROPOSED THESIS

In section 3.1, our thesis objective is introduced. Section 3.2 presents the expected results and the potential benefits of this thesis. Section 3.3 describes our methodology and is the starting point for achieving our research objective.

3.1 Objective

This thesis proposes new approaches for obtaining feedback on software functional complexity and quality, and investigates their applicability to testing and reliability assessments in the context of the COSMIC-FFP measurement method adopted in 2003 as the ISO/IEC 19761 standard (Abran, Desharnais et al. 2001), (ISO/IEC19761 2003), (ISO14143-1 1988).

Functional complexity is quantified in terms of the entropy of an amount of information exchanged between the environment and the software system, and within the system, in one usage of the system. As an FSM, COSMIC-FFP measures software functionality in terms of the data movements across and within the software boundary. There is almost no cross-referencing in the software engineering literature between these two fields of knowledge, that is, FSM and the measurement of entropy. With the recent publication of work on the measurement of functional complexity based on entropy concepts (Abran, Ormandjieva et al. 2004), there is now an opportunity to investigate the candidate linkages, and candidate contributions, across the two fields. On the one hand, COSMIC-FFP measurement concepts and procedures are well documented and, through the method's international acceptance as an ISO standard, it has achieved international

recognition supported by the international community specializing in software measurement. However, the field of software functional size by itself has very limited depth in terms of research and theoretical support on which to draw. Its use is therefore fairly limited, extending only to productivity studies and estimation, with almost no reported use in complexity, quality and reliability analysis.

On the other hand, entropy has been used extensively in many fields, including entropy-based measurement, to measure performance for instance (V. S. Alagar, O. Ormandjieva and J. Shen 2004), and for reliability estimation (Weiss and Weyuker 1988), both with very strong theoretical and empirical support.

The thesis directions include the use of entropy measurement for managing the test adequacy of the scenario-based black-box test cases generated from COMSIC-FFP models. COSMIC-FFP scenarios are further transformed into state diagrams for reliability prediction purposes within the COSMIC-FFP context.

Reliability prediction in our work here uses the mapping of the state diagrams to discrete-time Markov chains to analyze the reliability of a system.

The thesis also includes a formalization of COSMIC-FFP in the AS-TRM (Autonomic System Timed Reactive Object Model) context (Achthan 1995) – an extension of the Timed Reactive Object Model (TROM) formalism for real-time reactive systems created at Concordia University. This formalization targets the modeling of autonomic systems,

the architecture, system configuration and continuous quality self-monitoring of which are to be specified within a single formal framework, and would allow the automatic measurement of the size and functional complexity of AS-TRM specifications from several case studies.

The thesis objectives are summarized as follows:

1- Use of COSMIC-FFP in the measurement of complexity.
2- Use of COSMIC-FFP for black-box testing strategies.
3- Use of COSMIC-FFP for the early prediction of reliability.
4- Use of COSMIC-FFP for the early assessment of complexity from formal specifications, i.e. AS-TRM.

3.2 Expected Results and Benefits

To the best of the researchers' knowledge, this work is the first extension of COSMIC-FFP to testing and reliability assessment, and, as such, it would be of benefit to industry insofar as it is concerned with compliance with the internationally recognized ISO standards. Early feedback on software quality would allow for effective quality control earlier in the development process, which would increase the reliability of the final product and reduce the development and testing cost. The benefits of this work will also include formalization of the COSMIC-FFP measurement in the AS-TRM formalism for autonomic system development. The testing quality assessment method introduced here has been adapted successfully to the ERP methodology (Daneva, Abran, Ormandjieva and Abu Talib 2006).

3.3 Methodology

The following figure summarizes the complete thesis methodology, which consists of 5 phases:

1. Mapping COSMIC-FFP to the measurement of complexity based on entropy (chapter 5).
2. Validation (chapter 5).
3. Combining COSMIC-FFP procedures with a black-box testing strategy (chapter 6).
4. Extending COSMIC-FFP for reliability assessment purposes (chapter 6).
5. Mapping between COSMIC-FFP and the AS-TRM (chapter 7).

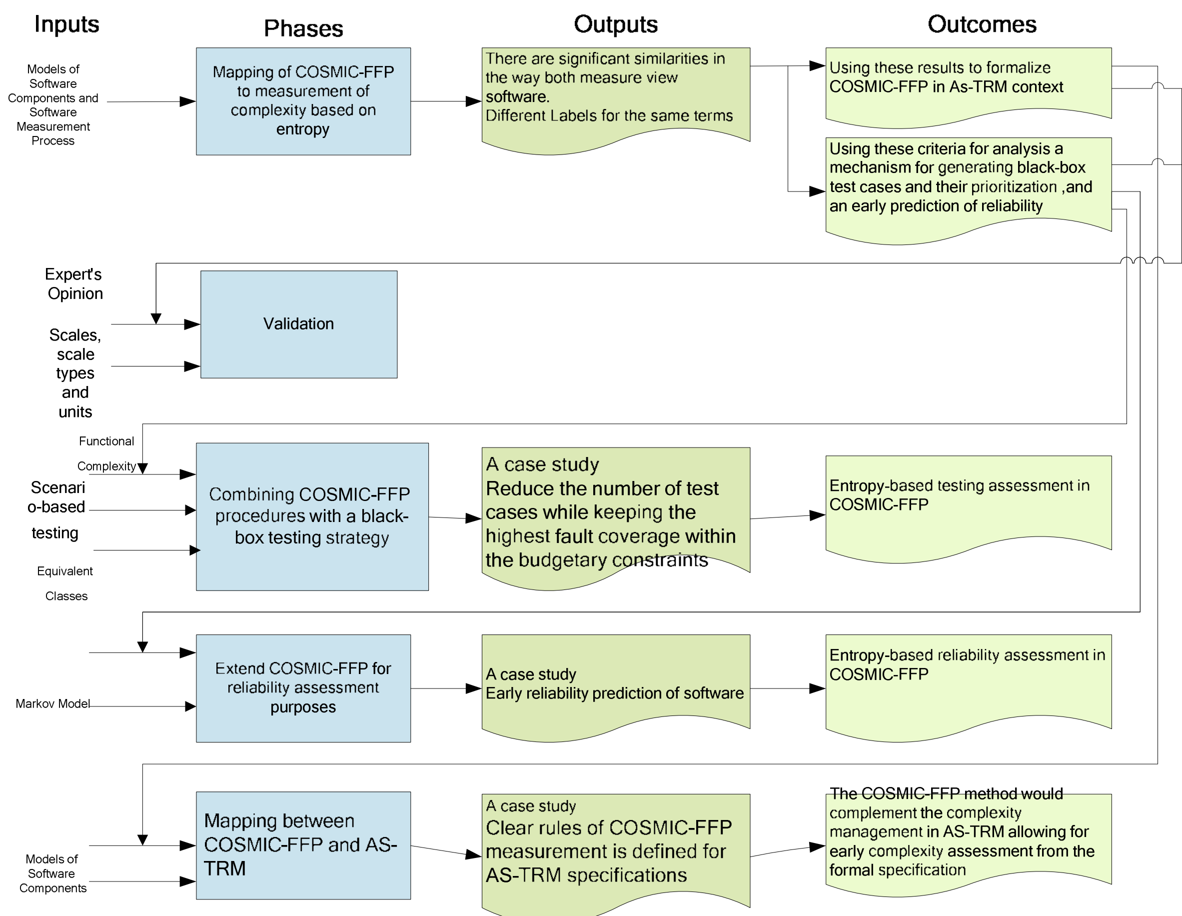

Figure 6: Thesis methodology

In the first phase, a study is conducted with the objective of mapping COSMIC-FFP to the measurement of complexity based on entropy. The results of this study, as will be detailed in chapter 5, show that there are significant similarities across these two measures in the way that they view software. This helps in analyzing a mechanism for generating black-box test cases and prioritizing them, and for obtaining an early reliability prediction for software. It also helps in formalizing COSMIC-FFP in the AS-TRM context.

In the second phase, based on expert opinion and related concepts in scale types, three metrological properties in particular (scale, unit and scale type) are validated in both these measurement methods. This study is presented in chapter 5.

The third phase of the methodology combines COSMIC-FFP procedures with a black-box testing strategy based on the results obtained in the first phase and the related work that has been performed in the area of scenario-based testing and equivalence classes. As a result, entropy-based testing assessment is achieved in COSMIC-FFP by reducing the number of test cases, while keeping the highest fault coverage within the budgetary constraints. The feasibility of this concept is demonstrated by applying the proposed testing approach in the Hotel Reservation System case study.

Extending COSMIC-FFP for reliability assessment purposes is achieved in the fourth phase, where early software reliability prediction is illustrated by the Railroad case study

based on the Markov model of software components. The third and fourth phases are presented in greater detail in chapter 6.

Finally, the fifth phase is detailed in chapter 7, where COSMIC-FFP is mapped to the AS-TRM, and, therefore, clear rules for COSMIC-FFP measurement are defined for use with AS-TRM specifications. This will complement complexity management in the AS-TRM, in turn allowing for early complexity assessment from the formal specifications. The feasibility of this concept is demonstrated by applying these rules in the Steam Boiler case study.

Before detailing each phase in the following chapters, related work in the fields of software testing, software reliability and AS-TRM formalization is introduced in the next chapter.

CHAPTER IV: RELATED WORK

This chapter provides a brief overview of the work that has been done so far in related areas related that is relevant to the proposed research. Section 4.1 presents the related work that has been done in scenario-based testing and in equivalence classes. Section 4.2 discusses the related work that has been used for software reliability prediction and the corresponding tools that support it. The work related to AS-TRM formalism is introduced in section 4.3.

4.1 Software Testing

In the following sections, the work related to scenario-based black-box testing and equivalence classes is reviewed.

4.1.1 Scenario-based Testing

There are three main testing strategies available: white-box testing, black-box testing and grey-box testing. In white-box testing, the test suite is generated from the implemented structures. In black-box testing, the structure of the implementation is not known, and the test cases are generated and executed from the specification of the required functionality at defined interfaces. In grey-box testing, the modular structure of the implementation is known, but the details of the programs within each component are not.

Scenario-based testing is a typical black box testing methodology at the system level, in which the scenarios depict the sequence of executions of the system, and the test cases can be derived from the use-case model and its corresponding UML diagrams (Bai, Peng and Li 2002; Bai, Tsai, Feng and L. Yu 2002). UML is the de facto industrial standard for modeling object-oriented software systems (Bai, Peng et al. 2002). There are 12 kinds of

diagrams in 3 categories in UML: structural diagrams (including class diagrams), behavioral diagrams (including use-case diagrams, interaction diagrams (e.g. Figure 9), activity diagrams, collaboration diagrams and state-chart diagrams) and model management diagrams.

In (Bai, Peng et al. 2002), a use case is defined as a collection of related scenarios describing the actors and operations in a system, and these use cases can be organized hierarchically. Specifically, the root of the tree is the main use-case diagram, the middle branches are the low-level use-case diagrams and the leaves are the sequence diagrams for each use case in the low-level use-case diagram. However, the main use cases may be at too high a level to derive test cases. The authors propose transforming a use case into scenarios, then from a scenario into thin threads, and, finally, from thin threads into test cases – Figure 7.

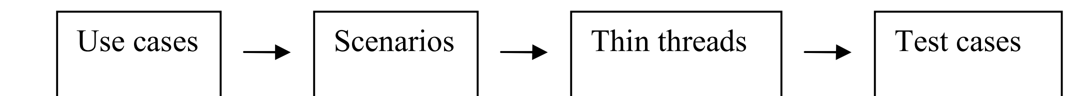

Figure 7: Testing with UML

A scenario is a specific sequence of actions and interactions between actors and the system (Bai, Peng et al. 2002). A thin thread is a minimum-usage scenario in a software system. It is a complete data/message trace using a minimally representative sample of external input data transformed through an interconnected set of the system (architecture) to produce a minimally representative sample of external output data. The execution of a thin thread demonstrates that a method performs a specified function (Bai, Peng et al. 2002).

The proposed scenario-based testing approach follows the related work in the same way; however, it is aimed at creating an optimal set of test cases in the context of the COSMIC-FFP method based on a specified budget, as will be seen in section 5.1.

4.1.2 Equivalence Classes

Weiss and Weyuker (Weiss and Weyuker 1988) partitioned the input domain into some equivalence classes with respect to the behavior of the system under test. This approach mainly reduces the number of test cases with respect to the input domain.

Davis and LeBlanc (Davis and Leblanc 1988) discussed the entropy-based software measures at a higher level by considering chunks of code. For example, a single statement, a block of code or a module itself can be considered as a chunk of code. Groups of chunks may form an equivalence class if they have the same number of in-degrees and out-degrees. In Figure 8, A and C belong to the same equivalence class. The entropy is therefore computed with respect to the chunks that are in the same equivalence class, as follows (FC representing the Entropy-based Functional Complexity Measure): $FC = FC_1 + FC_2 + \ldots + FC_p$.

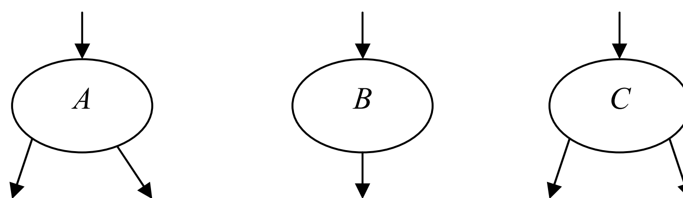

Figure 8: Example of an Equivalence Class

One of the problems in identifying equivalence classes in COSMIC-FFP is how to organize the scenarios into equivalence classes where each equivalence class is characterized by a distinct functionality. Determination of the equivalences classes themselves can be a tedious and rather time-consuming process. We propose an automatic extraction of equivalence classes from the scenario descriptions. The following research problems have been targeted and addressed by the proposed approach:

1. Defining criteria for partitioning the set of scenarios into equivalence classes of test cases based on their similarity;
2. Determining the partitioning of the input domain into equivalence classes by the sets of input events corresponding to the set of scenarios in the equivalence classes;
3. Prioritizing, within each equivalence class, the corresponding test cases based on their functional complexity values (in descending order) – see section 3.3 to calculate the functional complexity measurement.

The work we present in this thesis differs from related work in that we target the optimization of the testing process in the software environment prior to the actual implementation of the system. The proposed approach would allow the achievement of better testing results (through including test cases that cover most of the functionality in software) within a given budget. The approach is easy to implement as an enhancement to current testing processes.

4.2 Software Reliability

4.2.1 Markov Chains

The industrial applicability of Markov chains derived from state machine diagrams for reliability purposes, as described in (Relex), (Item-software) and (ISO-graph), is worth

mentioning. For instance, the RELEX Markov (Relex) provides fast, accurate reliability analyses for complex systems with common-cause failures, degradation, induced or dependent failures, multi-operational state components and other sequence-dependent events. Once a state transition diagram has been completed, the Markov model incorporates optimized algorithms to perform calculations accurately and supports both transient and steady state analysis results. In addition to calculating overall system results, the RELEX Markov also calculates parameters for each state.

4.2.2 Uncertainty Analysis

K. Goseva-Popstojanova and Sunil Kamavaram explore the notion of entropy in Markov models in a recent paper (Goˇseva-Popstojanova and Kamavaram 2004). They propose a methodology for the uncertainty analysis of architecture–based software reliability models suitable for large, complex, component–based applications and applicable throughout the software life cycle (Goˇseva–Popstojanova and Kamavaram 2003). Within this methodology, two methods for uncertainty analysis have been developed: the method of moments and Monte Carlo simulation. In (Goˇseva–Popstojanova and Kamavaram 2003), the method of moments is used to quantify the uncertainty in software reliability due to uncertainty in component reliabilities. The expressions derived in (Goˇseva–Popstojanova and Kamavaram 2003) are valid for independent random variables, but did not allow the uncertainty in software reliability to be studied due to uncertainty in the operational profile. Generalizing earlier research work on the method of moments, these authors then derived expressions for the mean and the variance of system reliability, which consider both sources of uncertainty in software reliability (the way software is

used, i.e. the operational profile) and the component's failure behavior (i.e. component reliabilities).

This was illustrated through case studies in which the estimated values of the system reliability moments provide more information than the traditional point estimate. Thus, these authors have a higher level of confidence in the reliability predictions for systems with a reliability having smaller variance. This information is especially useful in making predictions early in the life cycle, in keeping track of software evolution and in certifying the reliability of component-based systems.

The reliability assessment method discussed in this work shares the related work methods in important ways:

1. It focuses on the early reliability prediction of software.
2. It is based on the architecture model of software and the state machine description of its components.
3. It models the software as a Markov system.
4. The reliability prediction is derived from the steady state of the Markov system.

However, the proposed method enhances the use of COSMIC-FFP, not only for estimating the functional size of software, but also for testing and reliability assessments, as this thesis introduces COSMIC-FFP as a package to be used for many purposes to control and manage the software development process.

4.3 Reliability Assessment in the AS-TRM

The TROMLAB reliability assessment is based purely on the architecture model of the autonomic reactive group and the state machine descriptions of the reactive components. It is supposed that the transition that will be triggered from one state is the same as a random walk; based on it, we can use Markov model to analyze the reliability of the state machine of an autonomic reactive system. Therefore, the reliability prediction is derived from the steady state of the Markov system.

The work steps in the reliability assessment (Lee 2003) are as follows:

Initialization: Calculation of the transition matrices for the participating objects.

Step 1. Mapping an object to a Markov system

Step 2. Mapping two synchronously interacting objects to a Markov system

Step 3. Mapping a subsystem to a Markov system

For each subsystem, there is a corresponding output file containing all the intermediate computational results; these include:

- Transition matrix for each GRC specification in the subsystem;
- Transition matrix for synchronous product machines generated from the GRC specifications;
- Steady vector of each GRC specification;
- Steady vector of the synchronous product machine specification;

- Various debugging information on the contents of the internal data structure;
- The reliability of the subsystem.

It will also produce an output file containing the reliability of each subsystem and the reliability of the overall system.

Now, the reliability measure for a system in TROM-SRMS is the lowest reliability measure of its *m* subsystems:

$$Reliability(System) = min\{Reliability(Subsystem_i)\}_i^m$$

The minimum value is chosen due to the safety-critical character of the real-time reactive system. The higher the value of the reliability measure, the less uncertainty there is in the model, and thus the higher the level of software reliability.

H is calculated as a level of uncertainty of the Markov system corresponding to a subsystem:

$$H = -\sum_i \pi_i \sum_j p_{ij} \log p_{ij}$$

where π_i is a steady state distribution vector. *Hi* is a level of uncertainty in a Markov system corresponding to an object. We also write it as:

$$Reliability(Subsystem) = \sum_{i=1}^{n} H_i - H$$

H is related exponentially to the number of paths that are "statistically typical" of the Markov system. The higher the entropy value, the more sequences must be generated in order to accurately describe the asymptotic behavior of the Markov system.

Our work differs from the TROMLAB reliability assessment method in the following ways:

- the number of steps in the reliability calculation is reduced to two;
- the reliability of a system is calculated from the configuration type of the components in the system.

Now that the related work on testing and reliability fields has been introduced in this chapter, and having defined the proposed methodology in chapter 3, COSMIC-FFP is mapped to entropy measurement in chapter 5.

CHAPTER V: MAPPING COSMIC-FFP TO ENTROPY

5.1 Introduction

In our work here, a new method is proposed for quantifying functional complexity from the description of the software's behavior. The proposed functional complexity measure characterizes the performance of the system as specified in the scenarios. Functional complexity is quantified in terms of the entropy of an amount of information based on an abstraction of the interactions among software components. Assuming that each message represents an event, therefore, entropy-based software measurement is used to quantify the complexity of interactions between the software and its environment and within the software (between software classes) in terms of the information content of the interactions, based on some abstraction of the interactions (Davis and Leblanc 1988; Shannon, E. et al. 1969; Harrison 1992).

In OO development, the only vehicle for information interchange between the software and the environment, and within software modules (classes or packages), is the event, also called the "message". The environment is communicating with the OO software via external messages: input (to communicate a request from the environment for a service/usage) and output (to communicate the answer from the software to the environment). In order to fulfill the requested functionality, the objects collaborate via message exchange. In the Rational Unified Process (RUP) (Kruchten and Philippe 2000) for OO software development, the interactions (exchange of messages, that is, events, between the environment and the modules, and between the modules themselves) are described as scenarios – written stories of a system's functionality related to one specific

usage. In UML, the interchanging of events in one scenario is represented graphically using standard interaction diagrams.

For the purpose of measuring functional complexity, each scenario is mapped to a timed sequence of events, where each event is considered as a unit of information. The scenario described in Figure 9 has the sequence of events (messages) $e_1.e_2.e_3.e_2.e_5$. To quantify the amount of information contained in this scenario, we can apply the entropy formula (1). Therefore, the set of events and the set of their probabilities are $\{e_1, e_2, e_3, e_5\}$ and {1/5, 2/5, 1/5, 1/5} respectively.

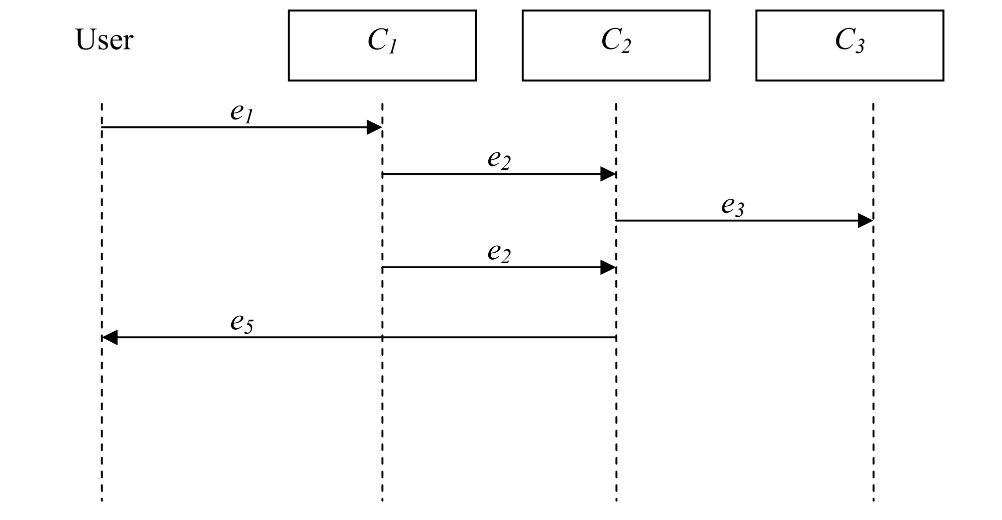

Figure 9: Scenario example

The functional complexity for system implementation (Alagar, Ormandjieva et al. 2000) is defined as an amount of work output performed in a time slice by the system. The amount of work performed, in this context, means the quantity of information processed in that period of time, and the number of functions necessary to perform the work. The

events represent the functions necessary to perform the work in one usage of the system, i.e. in one scenario.

The concepts of information theory (Alagar, Ormandjieva et al. 2000) are applied to measure the amount of work output performed in a time slice by the system in terms of the amount of information in the event sequence. That measure is based on an empirical distribution of events within a sequence.

The probability of the ith most frequently occurring event is equal to the percentage of the total number of event occurrences it contributes, and is calculated as $p_i = f_i / NE$, where f_i is the number of occurrences of the ith event and *NE* is the total number of events in the sequence. The classical entropy calculation quantifies the average amount of information contributed by each event. Therefore, the functional complexity in a time slice is defined in (Alagar, Ormandjieva et al. 2000) as an average amount of information in the corresponding sequence of events, and is computed as follows:

$$FC = -\sum_{i=1}^{n} (f_i / NE) \log_2 (f_i / NE) \quad \ldots\ldots\ldots (3)$$

FC represents the quantification of the amount of information exchanged in a given interaction (scenario). The functional complexity in a period of time with a higher average information content should, on the whole, be more complex than that of another with a lower average information content. That is, the FC measure is intended to order the usages of system in a time period in relation to their functional complexity.

5.2 Analysis of Similarities and Differences across COSMIC-FFP and Entropy Measures

The method used to analyze the compatibility between the Entropy-based Functional Complexity Measure and COSMIC-FFP consists of comparing the generic software models, their software model components and their software measurement processes (Oligny and Abran 1999). For such a comparison, it is necessary to identify the concepts behind the terms used in the two measures: in fields that are not yet mature and where the terminology is not fully standardized, such as software engineering, different terms will sometimes refer to the same concepts, and, at other times, the same term will refer to different concepts (Oligny and Abran 1999).

1) Models of Software Comparison

In their generic view of software from a functional perspective, the two measures share a similar generic modeling of how to recognize the functionality of software.

Information theory-based software measurement quantifies functional complexity in terms of an amount of information based on some abstraction of the interactions between software modules, and, more specifically, the complexity of interactions between the software and its environment and within the software itself in terms of the information content of the interactions.

COSMIC-FFP has a similar generic model of software functionality, which is defined as the interactions between the software and its environment and within the software itself, as illustrated in Figure 2. In COSMIC-FFP, the environment is represented by the users

interacting with the software, such users being either humans, engineering devices or other software applications. Within the software, the interactions deal with the data read from or sent to persistent data storage.

2) Models of Software Components Comparisons

The second step consists in identifying and comparing the software model components used by each method required for the instantiation of the generic software model of functionality. In the RUP context, the functional processes used in COSMIC-FFP can represent the set of scenarios for the software. For example, in the Hotel Reservation System, the user can create a reservation. This process of allowing the user to add a new reservation is considered as a functional process, and is triggered by selecting the user for this option. Similarly, creating the reservation is a scenario containing a sequence of events between the user and the system, and this scenario contains a sequence of events within the system. Therefore, for each functional process, its subprocesses and its triggering events are sequences of events (events).

Within the same RUP context for functional complexity measurement, the entropy formula can likewise be calculated on one process (scenario).

See Figure 10 for the functional process for creating a reservation that can be the scenario of a set of events interchanged between software components. The set of alphabets (events) comprises {1: select "create", 2: display, 3: type required information, 4: store

information, 5: display error message if it occurs} and their probabilities: {1/5, 1/5, 1/5, 1/5, 1/5}.

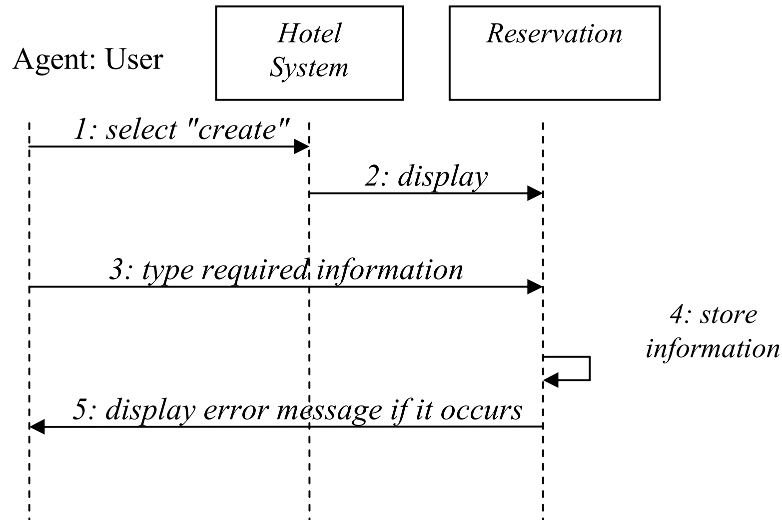

Figure 10: Create Reservation Sequence diagram

Note next that the measurement unit defined in COSMIC-FFP is a data movement, that is, a base functional component which moves one or more data attributes belonging to a single data group. It can now be observed that an event as the unit of the Entropy-based Functional Complexity Measure has the same meaning as the data movement, which is the unit of COSMIC-FFP.

A summary of the terms used in both COSMIC-FFP and the Entropy-based Functional Complexity Measure having similar meanings is presented in Table 6. This table shows the same conceptual level for both COSMIC-FFP and the Entropy-based Functional Complexity Measure; however, the terms used in the data movements of COSMIC-FFP and in the interactions of the Entropy-based Functional Complexity Measure have different labels. For example, in COSMIC-FFP, data movements are classified in four

categories. The term corresponding to the data movements and its categories that is used in Entropy-based Functional Complexity Measure is the event, but without classification. In addition, other terms used in both measurement methods, such as those interacting with the software, the software boundary and the set of user requirements, have the same meaning even though some have different labels. For example, software users are actors, while at the same time they are interacting with the software. As can be seen in Table 6, the models of software of these measures are compatible. The measurement unit and the measurement unit symbol for Entropy-based Functional Complexity Measure is analyzed more extensively in chapter 5 (section 5.3), which is dealing with the scale type issue, and that work has been published in (Abu Talib, Abran et al. 2005).

Table 6: Similarity between COSMIC-FFP & Entropy-based Functional Complexity Measure concepts

Concepts	COSMIC-FFP (Data Movement) terms	Entropy-based Functional Complexity Measure (Interaction) terms
Humans or things interacting with the software	Software users	Actors
Interactions between the environment and the software	Software boundary	Software boundary
Set of User Requirements	Functional Process	Scenario (Sequence of Events)
Data which are part of the interaction	Data groups	Set of data attributes
External Input (from the environment)	Triggering event	Event
External Input (from the environment)	Entry data movement	Event
Output (to the environment)	Exit data movement	Event
Entity being taken into account in the measurement	Data movement	Event
Measurement Unit	1 data movement	Bit
Measurement unit symbol	Cfsu	Bit

3) Software Measurement Process Comparison

Even though each type of measure takes into account similar concepts (with different terms) for the model of the software to be measured, each type of measure, when its measurement processes are compared, defines different measurement functions (e.g. formulas) to combine the information into a 'measure' for purposes which are obviously different. For example, the COSMIC-FFP formula is used to measure the functional size of software, that is, the amount of functionality the software has through the addition of data movements. COSMIC-FFP recognizes only data movement type subprocesses, and it contains an approximation assumption that each data movement is associated with an average amount of data transformation.

By contrast, functional complexity measurement based on entropy is used to measure the amount of information in the interactions between the software and the environment, and within the software modules, a formula which associates the functional complexity of software with the frequency of occurrence of a function through a logarithmic function of probability distribution of the events (see formula 3). It can be easily observed that the Entropy-based Functional Complexity Measure extracts more information than does COSMIC-FFP about events, their frequencies and their probabilities of occurrence.

For illustrative purposes, let us consider two events: e_1 and e_2. In Figure 11, if either $e_1$or e_2 is certain (p_1 =1 or p_2 = 1), then FC is zero. The same thing will happen when p = 0 for either event. However, when p_1 = 0.5 or p_2= 0.5, the two events are just as probable and FC is 1, which is the maximum. Therefore, FC is maximum if all probabilities are equal, and it is minimum if one event has a probability equal to 1. However, in COSMIC-FFP,

when two different data movement types are required to perform the functional process, then the number of Cfsu is 2. That number is also 2 when one data movement is required twice and the same data group is accessed in order to execute the functional process. Note that two functional processes may in the end have the same functional size regardless of the type of data movement.

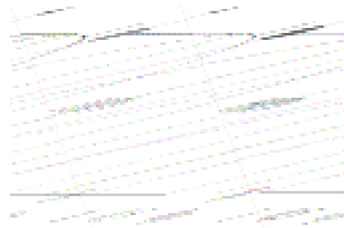

Figure 11: Entropy-based Functional Complexity Measure

5.3 Theoretical Validation

This section presents a study of the scales, units and scale types of both COSMIC-FFP and the Entropy-based Functional Complexity Measure. The work for this study has been published in (Abu Talib, Abran et al. 2005). Previous studies have analyzed the scale types of many software measures (such as Zuse (Zuse 1991), Fenton (Fenton and Pfleeger 1998) and Whitemire (Whitmire 1997), but not the concept of scale nor how it is used in the design of a measurement method.

Well-designed and well-defined measures in the sciences and in engineering should have most of the many characteristics described in metrology (ISO 1993), including 'scales', 'units' and 'etalons' to which measuring instruments should refer to ensure meaningfulness of the numbers obtained from measurement. However, some of these concepts, such as units, scales and etalons, have not yet been addressed and discussed by

researchers in terms of empirical validation approaches of software measures (Fenton and Pfleeger 1998): for instance, researchers in software measurement have, to date, focused on scale types rather than on the scale embedded within the definitions of these measures. This could lead to less than optimally designed software measures. Moreover, when these software measures are analyzed without taking into account metrological concepts, the result can be improperly stated conclusions about their strengths.

Measurement with COSMIC-FFP is more than counting and adding up the data movements. To identify the types of scales and analyze their uses in the COSMIC-FFP measurement process, the measurement process procedure must be broken down into substeps, and each substep further analyzed in order to understand the transformation occurring between the steps (Abran and Robillard 1994). As mentioned previously, two phases (mapping and measurement) are required to measure the functional size of software in COSMIC-FFP. Basically, the mapping phase is the process of abstracting a set of FURs, whatever methodology is used to describe them, as a COSMIC-FFP generic model of the software. This is similar to mapping a distance traveled into meters or mapping time into a mechanical clock. Only then will the measurer be able to read the distance on the scale or the position of the hands on the clock face. Therefore, the mapping phase, mapping the FUR set into a measurement scale, is an important step. This takes the measurer to the next phase, which is the measurement phase.

More specifically, the mapping phase involves identifying software layers (MAP 1), and then, for each layer, carrying out the following steps:

MAP 1.1: Identify software boundaries.
MAP 1.2: Identify functional processes.
MAP 1.3: Identify data groups.
MAP 1.3.1: Identify data attributes.

For MAP 1, the concept of software layers in COSMIC-FFP is meant as a tool which identifies both the individual components that have to be sized and their boundaries (Abran, Desharnais et al. 2001). In a specific measurement exercise, layers can be distinguished and have an order where, for instance, software in any one layer can be subordinate to a higher layer for which it provides services. Also, the measurement method defines a "peer-to-peer" exchange, as two items of software in the same layer exchanging data (Abran, Desharnais et al. 2001). From this point on, it can be said that a layer at level 2 is above a layer at level 1, which is itself used by the layer above that, or we can say that two pieces of software are at the same level or in the same layer.

The next step, MAP 1.1, involves identifying the boundaries between each pair of layers where one layer is the user of another, and the latter is to be measured. In the same layer, there is also a boundary between any two distinct pieces of software if they exchange data in "peer to peer" communications (Abran, Desharnais et al. 2001).

In MAP 1.2, the set of functional processes of the software to be measured is identified. In each layer, software delivers functionality to its own users. A functional process, which comprises at least two data movements, an entry and either an exit or a write (Abran, Desharnais et al. 2001), can be derived from at least one identifiable FUR.

Next, in (MAP 1.3), the data groups are identified. A data group is the object of interest that may or may not survive the functional process using it (Abran, Desharnais et al. 2001). Each data group has a set of data attributes which is non-empty and non-ordered.

MAP 1.3.1 is the last step in the mapping phase. In it, the data attributes for each data group are identified.

Note that, the steps taken in the mapping phase having been analyzed, steps MAP 1 to MAP 1.3.1 by themselves are not taken into account in the measurement of COSMIC-FFP functional size: only ‘data movements’ are considered directly in the measurement with units of 1 Cfsu, as will be seen later in the measurement phase.

Figure 12 explains the contribution of the mapping phase to the measurement process (Sellami and Abran 2003). The measurand is basically the textual description of the text within which the FURs are embedded in any kind of format. Then, the measurement signal would be basically the elements within the text that are related to the FURs.

Finally, the mapping from the format, whatever it is, into the generic COSMIC model of software could be the ‘transformed value’. It is to this transformed value (e.g. the identification of all Functional Processes recognized by COSMIC-FFP) that the measurement function would be applied with the corresponding measurement unit.

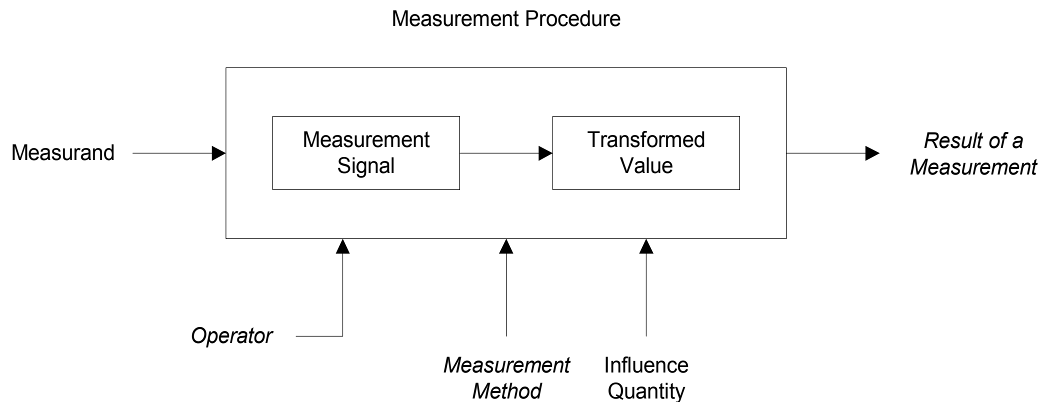

Figure 12: Measurement process – detailed topology of subconcepts (Sellami and Abran 2003)

The second phase is the measurement phase, where the measurer applies the measurement to the required elements of the model produced in the mapping phase. Each functional process included within the software boundary identified in the mapping phase (MAP 1.2) undergoes the measurement phase, which is subdivided into three steps:

MSP1: Identify data movements.
MSP2: Apply the measurement function.
MSP3: Aggregate the measurement results.

MSP1 identifies the data movement types (Entry, Exit, Read and Write) of each functional process (Abran, Desharnais et al. 2001). Note that it is not the subprocesses that are directly taken into account, but the data movements within a subprocess: in COSMIC, a subprocess is defined as a combination of a data movement and a data manipulation.

Then, by convention, only a portion of the subprocess is taken into account in the use of a measurement scale, that is, only the data movement portion.

This could be similar to taking a subprocess and comparing it to a scale of an etalon, and, since the scale of the COSMIC etalon (defined by convention at a conceptual, rather that at a material level, as is done for the meter or the kilogram etalons) considers only data movements, taking only this portion as input to the measurement function with its measurement unit, that is, the Cfsu.

MSP2 is the next substep in the measurement phase, and involves applying the measurement function by assigning the numerical value of 1 Cfsu to each instance of a data movement (entry, exit, read or write). The results of this substep in COSMIC-FFP are interpreted in the following way: once the data movements have been identified, a measurement scale is used, which is defined as “1 data movement of whichever type”. The measurer assigns to the data movement being measured a measurement unit of 1 with respect to that etalon, and then assigns to it a symbol of 1 Cfsu. It is these results that are taken as the numbers that are counted.

Finally, in the last step, MSP3, the results of assigning 1 to each data movement are added together using formula 1, taking for granted that the results of MSP2 can be ratio numbers to be added.

This measurement of a functional process is very similar to a classic measurement exercise: a measurement scale of 1 data movement is used, and this Read on the measurement scale is the equivalent of the marks (each mark being 1 data movement = 1 Cfsu). The size is then determined in terms of the number of marks – or units – read on the scale.

In conclusion, the COSMIC-FFP measure can at least be considered on the ratio scale. Moreover, zero is meaningful, which means that when size = 0, the software does not have a size in terms of the COSMIC-FFP measurement unit.

As also mentioned previously, the FC formula quantifies the amount of information exchanged in a given scenario. That is done through the following steps:

FC1: Calculate f_i for each event in the given scenario.
FC2: Calculate NE for the given scenario.
FC3: Calculate f_i/NE for each event in the scenario.
FC4: Calculate $\log_2$ of FC3 for each event.
FC5: Multiply FC3 with FC4.
FC6: Add FC5 for all events.
FC7: Multiply FC6 by -1.

FC1 simply counts the frequency of occurrence of the events (that is, it identifies an event occurrence, and then adds all the occurrences identified, and that is equal to frequency). We therefore suggest that at least the ratio scale depends on counting the frequency of events, and, as a result, the unit will be the "event occurrence".

FC2 adds the total number of event occurrences in a scenario, and we also suggest that it be defined at least on the ratio scale. The unit is the "event occurrence".

FC3 is a derived measure dividing FC1 (ratio scale) by FC2 (ratio scale). Its scale type will be the weakest, according to (Henderson-Sellers 1996), and it can therefore be on the ratio scale. Division is done through the unified unit, event occurrence. Now, according to the analysis proposed in (Khoshgoftaar and Allen), because a ratio of quantities with

the same dimensions is itself dimensionless, the end-result is a dimensionless number and is expressed as a percentage. Note that FC3 represents the probability of the ith event occurring in the scenario.

FC4 applies the logarithmic function to FC3. The ratio of the logarithmic function $\log_2 n$ is exactly the number of binary digits (bits) required to represent the probability n of the event occurring. For instance, the number of combinations of a three-digit binary number can be 8 (n=8). Thus, this step transforms the dimensionless probability value into the number of digits required to represent it in bits.

In FC5, the representation size for the probability in bits is multiplied by the probability of occurrence of events of the same type. Each bit is a designator of the probability of one event's occurrence. The result is the total number of bits required to represent the probability of all the occurrences of one event in the sequence. Such a multiplication would normally produce a number with the bit as the measurement unit. We suggest that the scale type be at least on the ratio scale.

In FC6, the representational size for the probability of all occurrences of all events is calculated.

The resulting number in FC6 is negative because of the nature of the logarithmic function (it is negative on values less than one), but the amount of information will be non-negative. In FC7, multiplication by –1 is required in order to obtain the non-negative value for the amount of information. It is a simple transformation that does not change the scale type, since -1 has no unit.

In conclusion, the FC measure quantifies the representational size of the probabilities of all event occurrences in bits, and can be considered at least on the ratio scale. Also, the absolute zero is meaningful, since (theoretically) one scenario may have zero functionality, thus requiring 0 bits to represent it.

Once the linkage across COSMIC-FFP and functional complexity measures has been defined and their scale types investigated, it would be of interest to investigate the use of COSMIC-FFP in both testing effort and reliability prediction. This is accomplished in the following chapter.

CHAPTER VI: TESTING AND RELIABILITY PREDICTION APPROACH

This chapter describes the proposed testing and reliability assessment approaches applied to the COSMIC-FFP models. The subsections are organized around the sequence of activities to be performed during the research process.

6.1 Scenario-based Testing Assessment in COSMIC-FFP

Prior to explaining the proposed testing approach in COSMIC-FFP, the links between the concepts used in COSMIC-FFP models and the scenarios need to be identified.

As with all FSM methods, the design and rules for these methods are independent of technologies and development approaches. When measuring the functional size of software documented using a specific notation such as UML, it is necessary to establish how the generic measurement concepts are mapped to any notation. The mapping of COSMIC-FFP concepts to UML has been documented in (Jenner 2002). Six COSMIC-FFP concepts (boundary, user, functional process, data movement, data group and data attribute) have direct UML equivalents (use-case diagram, actor, use case, operation, class and data attribute) – see Table 7.

Table 7: COSMIC-FFP concepts and their UML equivalents

COSMIC-FFP Concepts	UML Equivalents
Boundary	Use-case diagram
User	Actor
Functional process	Use case
Data movement	Operation (message)
Data group	Class
Data attribute	Class attribute

One of the greatest benefits of the use-case technique is that it provides testers with a set of assets which can directly drive the testing process. An instance of a use case – a scenario – can be seen as a use-case execution that can be tested. Therefore, use cases are sources of potential test cases.

In black-box testing, the details concerning the way in which the data are transformed and manipulated in the scenarios are dropped, since this is a detail design issue (in the design phase, the detailed information related to how the data are transformed corresponds to the data structures and algorithms). This data transformation information is equivalent to the data transformation information in COSMIC-FFP. The COSMIC-FFP measurement method recognizes only the data movement type of subprocess, and includes an approximation assumption whereby each data movement is associated with an average quantity of data transformed (and whereby its true value does not deviate significantly from such an average).

6.1.1 Test-Case Generation

The procedure for generating test cases takes the use-case model as an input. For each scenario identified in the use-case model, we derive a test case by mapping the scenarios to a sequence of events in time (or data movements in COSMIC-FFP), as described in the scenarios. Next, the specific conditions that would cause the test case to execute are identified, and real data values are supplied.

One of the most significant challenges in system testing is the large number of specific scenarios that must be tested to ensure that the system behaves in accordance with its

requirements. Our testing approach targets this problem by reducing the number of test cases while keeping the highest test coverage within given budgetary constraints.

Effective management of the test coverage is the biggest challenge in the testing activity. For the application domains to which the COSMIC-FFP method is applicable, such as real-time, embedded and MIS software, the use cases can drive a significant number of test cases, although testing all of them may not be feasible. In our work here, we propose to manage test coverage by partitioning the generated set of test cases into equivalence classes, prioritizing the test cases within those classes based on the amount of information they process and selecting the most critical test cases based on a balance between cost and priority.

6.1.2 Partitioning into Equivalence Classes

We have adopted a strategy similar to that used in (Alagar, Chen, Ormandjieva and Zheng 2006), with some changes. Our approach is summarized in Figure 13. This approach would allow the use of a metric-based algorithm to partition the set of generated test cases, *STC*, from the test selection domain *V*, where "metric" means the distance between two elements in a set of test cases, defined in terms of their similarity and dissimilarity. Before applying such an algorithm, the test set *STC* is generated from *V* by using the test case generation algorithm. This algorithm is simply applied so that each scenario constitutes one test case, as described in section 5.1.1.

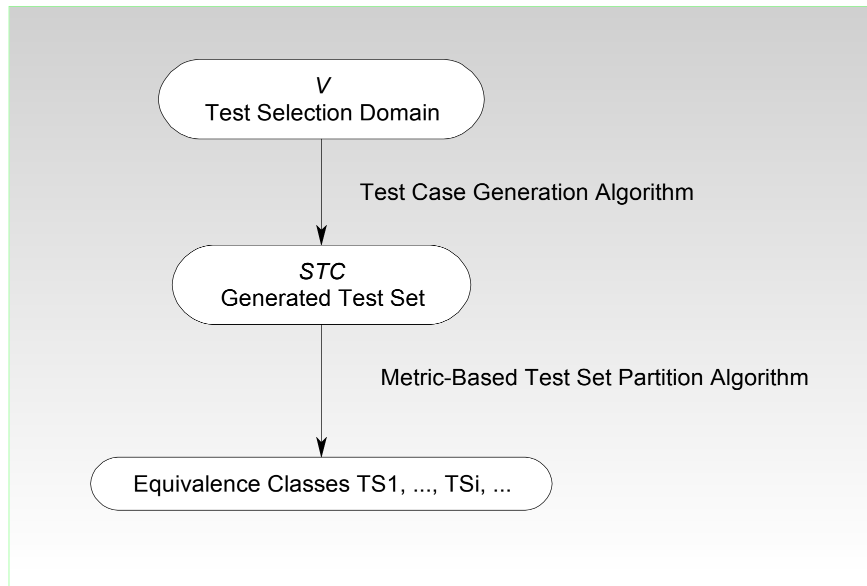

Figure 13: Test set partitioning strategy

Next, in order to select the subset of test cases to be included in one equivalence class, the following constraint must be satisfied to execute the second algorithm:

The distance ε between any two selected test cases should not be greater than a given constant value εmax.

The metric-based test set partitioning algorithm described in Figure 14 will be executed to create the equivalence classes until $STC = \emptyset$. This algorithm divides *STC* into equivalence classes, where each equivalence class TS_i will include test cases with similar functionality and be based on having short distances between the test cases in one equivalence class. Every time the algorithm is executed, one equivalence class TS_i will be created. We stop executing the algorithm when *STC* contains no more test cases. It should be noted that we do not choose the final set of test cases here, we only partition *STC*.

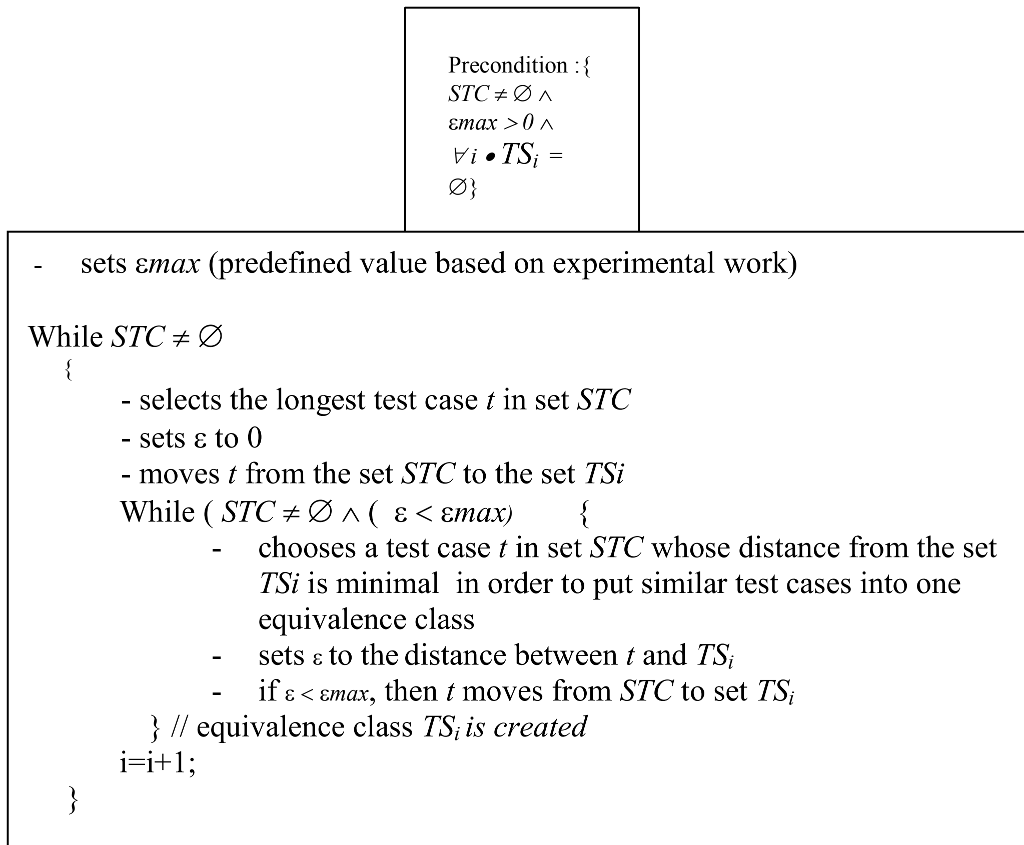

Figure 14: Metric-based Test Case Partitioning Algorithm

Now, we explain how the distance between the two test cases, t_1 and t_2, is calculated. The same formula that was used in (Alagar, Chen et al. 2006) is also used here:

$$td\ (t_1, t_2) = \text{Similarity}\ (t_1, t_2) * \text{Dissimilarity}\ (t_1, t_2) * e^{l}$$

Remember that t_2 is the test case that will be chosen for the second algorithm. l can be calculated as follows: -(*length* (t_1)/*length* (t_2)).

The Similarity (t_1 , t_2) = 2 (- *length* (*LCP* (*t1, t2*)), where *LCP* is the longest common prefix of the two test cases. The range of the Similarity measure is between 0 and 1.

The Dissimilarity measure between the two test cases, t_1 and t_2, is calculated as the number of elementary transformations minimally needed to transform the string (t_1/*LCP* (t_1, t_2)) into the string (t_2/ (*LCP* (t_1, t_2)). For example, the distance between the following two test cases: $e_1.e_2.e_3.e_4.e_1$ and $e_1.e_1.e_1.e_2.e_1 = ½ * 3 * e^{-1}$. The distance formula *td* (t_1, t_2) indicates that the more distance there is between two test cases, the more they will differ.

It is to be noted that to generate a set of test cases from software product and its own specification, in our case from COSMIC-FFP model, a testing method should be defined using a test case selection criterion. The level of test case adequacy, which is the degree to which the software is tested, is to be evaluated as well. The degree of adequacy of testing related to the test adequacy criterion is estimated by the test adequacy measurement. Theoretically valid coverage measures evaluate how well the test suite approximates its target. The measurement of the quality of coverage of the test suite would increase (or decrease) the confidence in tested components. Applying that in our testing approach, we can define the test case selection criterion in terms of number of scenarios that has been taken from COSMIC-FFP model and the most important and risk laden scenarios, which can be determined during the specification phase where the project typically undergoes multiple iterations.

6.1.3 Priority of Test Cases

For large systems, there may be a very large number of test cases, and a priority has to be assigned to them to help increase the testing coverage within the given budgetary

constraints. Prioritization of test cases is an important issue in software engineering because of limited testing budgets, and is usually performed manually.

In our approach, the FC, which was described in section 5.1, is used to prioritize test cases (Abran, Ormandjieva et al.). Intuitively, more, and diverse, functionality of the system would lead to a larger portion of the system being involved in that usage. The entropy calculated on a sequence of events abstracting a scenario quantifies the average information exchange for a given usage of the system. Therefore, it should correlate with the error spans during testing. The formula used to calculate functional complexity is as follows (Abran, Ormandjieva et al. 2004):

$$FC = -\sum_{i=1}^{n} (f_i / NE)\log_2 (f_i / NE)$$

The probability of the ith most frequently occurring event is equal to the percentage of total event occurrences it contributes, and is calculated as $p_i = f_i /$ NE, where f_i is the number of occurrences of the ith event and NE is the total number of events in the sequence. The priority assignment should be performed automatically.

6.1.4 Test Selection Algorithm

Note that total resource consumption is directly proportional to the number of test cases selected. As a result, the total cost of the test set is calculated by multiplying the number of selected test cases by *C,* where *C* is a positive constant scalar denoting the cost of one test case. In order to select the optimal subset of test cases that will be characterized by

the highest test coverage, we balance the budget and the priority of the test cases, as follows:

For all non-empty equivalence classes TS_i

Step 1. Choose the highest-priority test case from the equivalence class TS_i;

Step 2. Add the chosen test case to the Optimal Set and remove it from the equivalence class TS_i;

Step 3. Increase the total testing cost in C.

If (*the total testing cost exceeds a given budget Cmax)*
End the algorithm

End For

The final result of our testing approach is a set of selected test cases affording the best possible coverage (i.e. the test case with the highest FC value, from each equivalence class). However, we cannot claim full fault coverage, since this algorithm maximized the test coverage within the limits of a given budget. That is the limitation of the proposed testing method.

Moreover, we can use the Rational Rose utility, as seen in section 6.1.6, which proposes tools such as Robot, TestManager and the Test Plan for executing our selected test cases. Using these tools may enable us to arrive at a complete testing process that will be ready for interested companies to use, and will be useful to them. In the following section, we apply our testing strategy to a real case study obtained from ISO 14143-4: 2000 Set RUR A.1.

6.1.5 Case Study: Hotel Accommodation System (Reservation)

The feasibility of our testing approach is demonstrated with the following case study. The Hotel Accommodation System (Reservation) is documented in the ISO technical report: ISO/IEC TR 14143-4 (Version 2000). This ISO document provides various sets of FURs, usually described in a textual format. The purpose of this ISO document is to provide researchers and practitioners with sets of requirements to be used as publicly available documents for measuring the functional size of software.

The main menu of the hotel system offers two choices: accommodation and invoice & payment. The reservation system is part of the accommodation system. It has the following functions:

Function (1): RES Create Reservation

A reservation request can be entered using the RES screen. All data except the reservation number are entered. When changing the reservation data using the RES screen, the reservation number can be found by name, or by part of a name. All data, except the reservation number, can be changed. If there is more than one reservation with the same name, the selection screen (SEL-RES) is shown.

The system further checks whether or not the stated quantity of rooms of the desired room type is available for the desired period (not occupied or not reserved). "Being occupied" is checked on the basis of the data: room type, start date, number of days and quantity of reserved rooms. If necessary, more room types can be stored for the same period. Only room type and quantity of rooms can be entered.

If the request can be met, the acceptance screen ACP-RES stores the reservation and a confirmation of the reservation (CON-RES) is produced for the billing address. If the request cannot be met, a room type report (RT-REP) is called up to look up an alternative choice.

Screens used: RES (request for reservation), SEL-RES (selection of reservation), ACP-RES (accept reservation), RT-REP (room type report), CON-RES (confirmation of reservation).

Function (2): ACP-RES Accept Reservation

This function is performed by the RES function when a reservation request can be met. It displays the reservation details and the assigned reservation number. An accepted reservation can then be confirmed.

Function (3): SEL-RES Select Reservation

This function is performed by the RES function when a reservation request can be met. It displays the reservation details and the assigned reservation number. An accepted reservation can then be confirmed.

Function (4): RT-REP Room Type Report

This report is provided when a requested room type is not available. The Room Type Report shows the number of rooms which:

- are not occupied, and
- are not reserved

Function (5): CON-RES Confirm Reservation

This function is performed when an accepted reservation is confirmed. The confirmation can be made in four languages (EN, FR, GE or NL).

The Use Case (Figure 15) and the Sequence Diagrams (Figure 16-22) have been prepared to facilitate the identification of the data movements, and to ensure that all data movements have been identified.

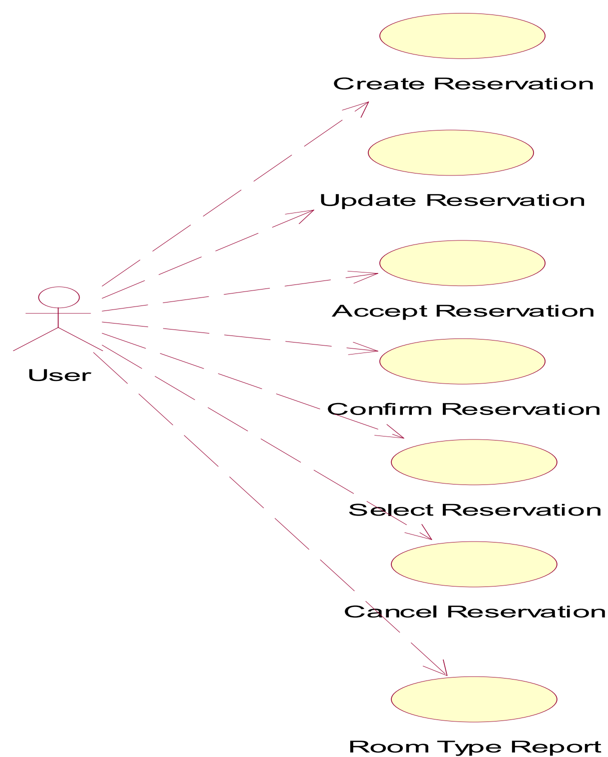

Figure 15: Hotel Reservation System – Use-Case diagram

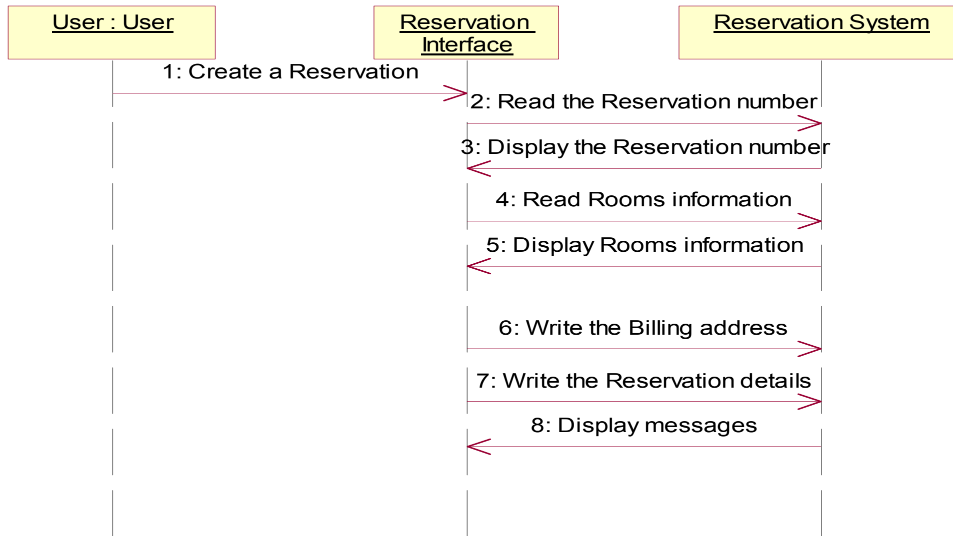

Figure 16: Create Reservation Sequence diagram

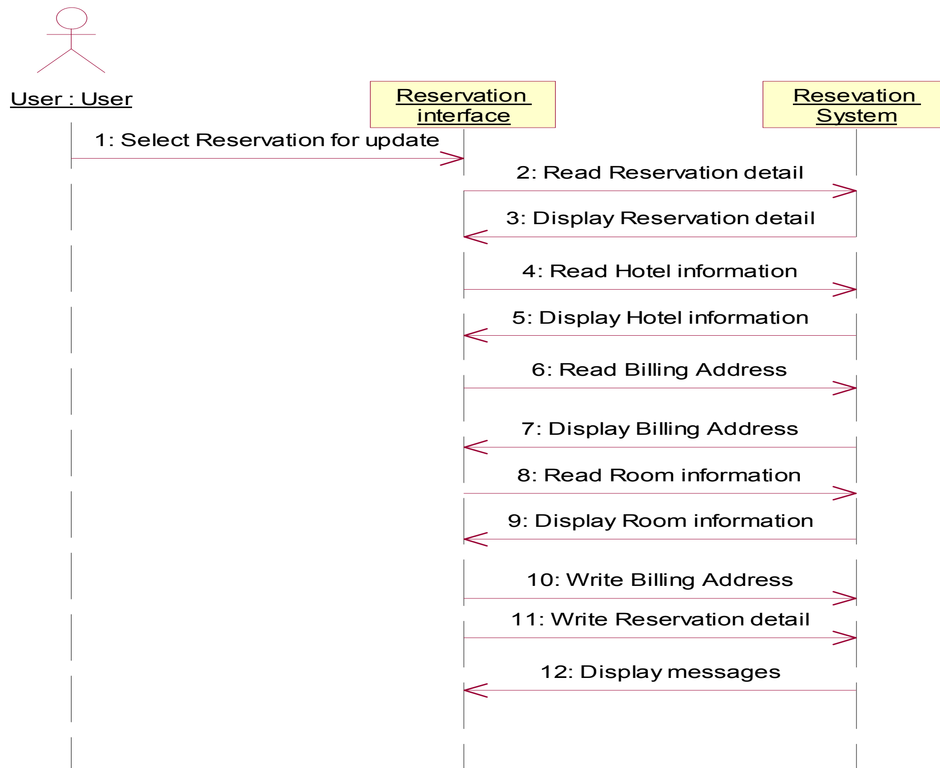

Figure 17: Update Reservation Sequence diagram

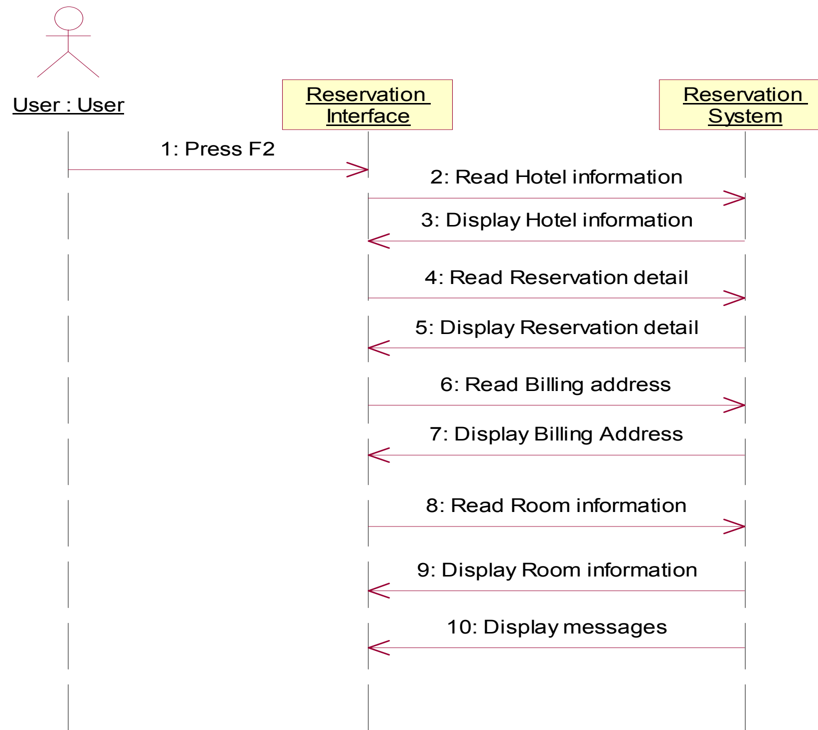

Figure 18: Confirm Reservation Sequence diagram

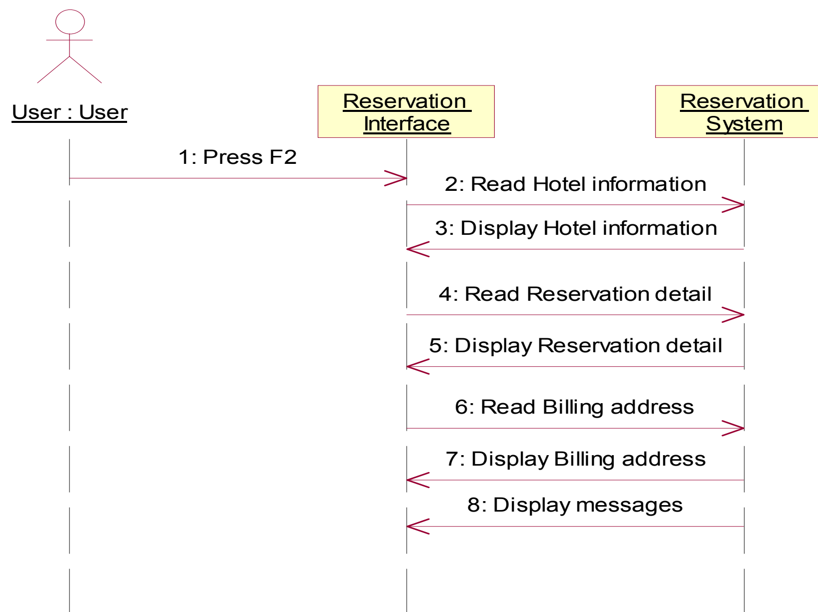

Figure 19: Accept Reservation Sequence diagram

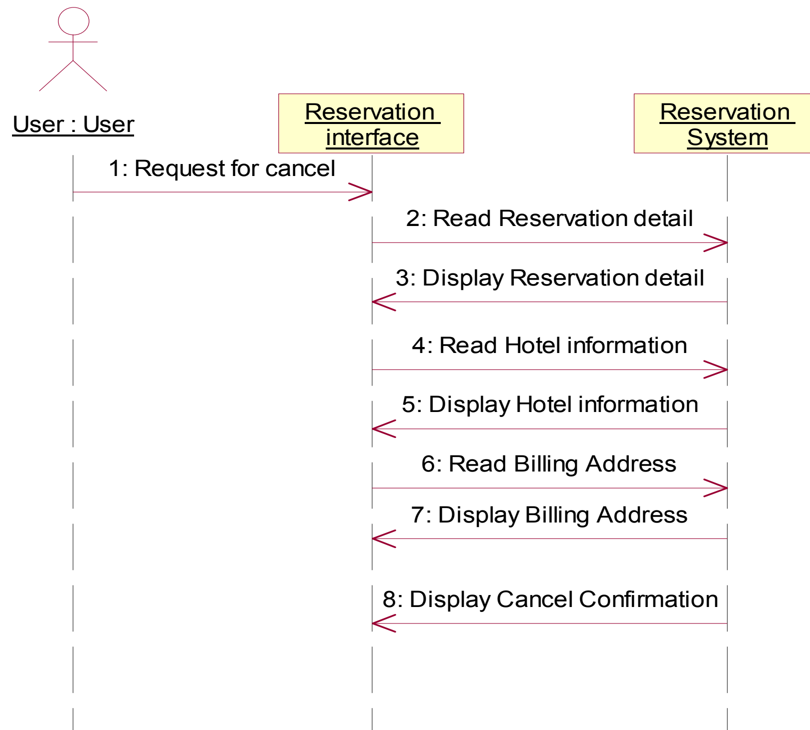

Figure 20: Cancel Reservation Sequence diagram

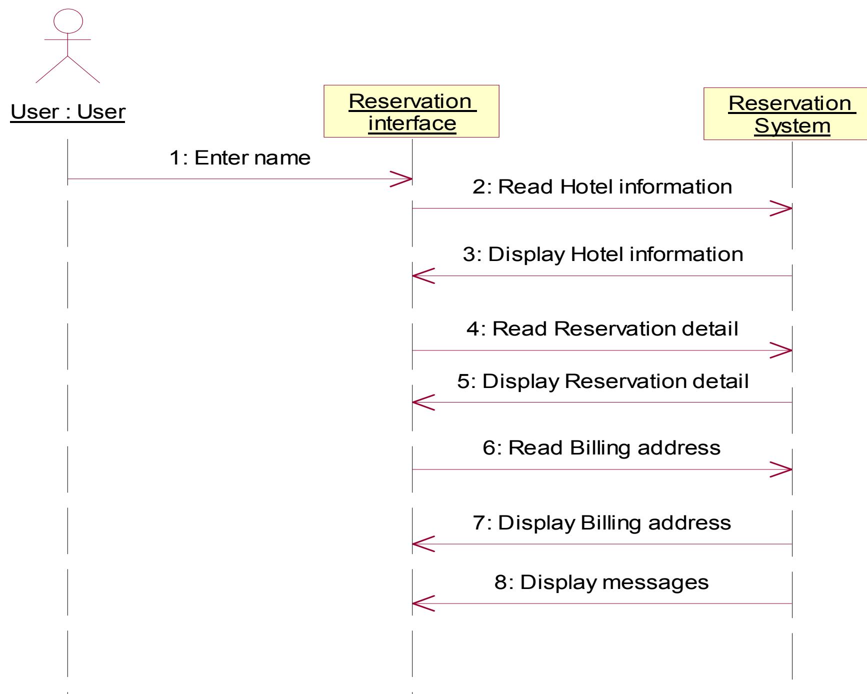

Figure 21: Select Reservation Sequence diagram

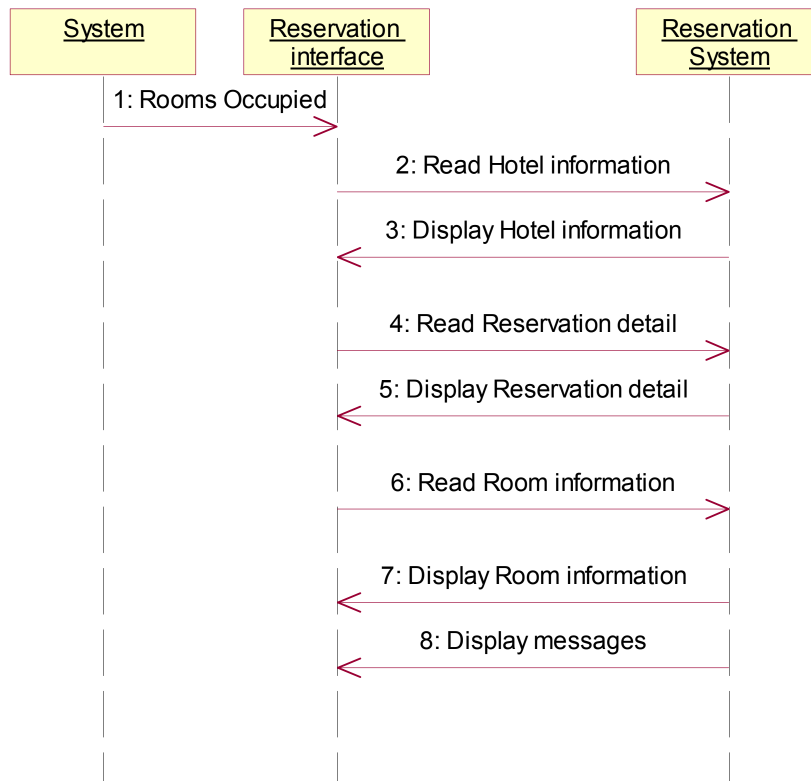

Figure 22: Room Type Report Sequence diagram

Applying the Proposed Testing Approach

1) Test-Case Generation

According to the sequence diagrams, we have the following events for the whole Hotel Reservation System:

e_1: Create a reservation
e_2: Read the reservation number
e_3: Display the reservation number
e_4: Read rooms information
e_5: Display rooms information
e_6: Write the billing address
e_7: Write the reservation details
e_8: Display messages
e_9: Select reservation for update

e_{10}: Read reservation details
e_{11}: Display reservation details
e_{12}: Read hotel information
e_{13}: Display hotel information
e_{14}: Read billing address
e_{15}: Display billing address
e_{16}: Request for cancel
e_{17}: Display Cancel Confirmation
e_{18}: Rooms occupied
e_{19}: Press F2
e_{20}: Enter name

According to the proposed test case generation algorithm, shown in Figure 13 and discussed in sections 5.1.1 & 5.1.2, the generated test set $STC = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, where Table 8 shows the detailed descriptions of the test cases generated from the scenarios (sequence diagrams).

Table 8: Test-case descriptions

Test case number	Test case description	Sequence Diagram that a test case is obtained from	Length of test case
t_1	$e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8$	Create Reservation	8
t_2	$e_9.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_4.e_5.e_6.e_7.e_8$	Update Reservation	12
t_3	$e_{19}.e_{12}.e_{13}. e_{10}.e_{11}. e_{14}.e_{15}.e_4.e_5.e_8$	Confirm Reservation	10
t_4	$e_{19}.e_{12}.e_{13}. e_{10}.e_{11}. e_{14}.e_{15}.e_8$	Accept Reservation	8
t_5	$e_{16}.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_{17}$	Cancel Reservation	8
t_6	$e_{20}.e_{12}.e_{13}. e_{10}.e_{11}. e_{14}.e_{15}.e_8$	Select Reservation	8
t_7	$e_{18}.e_{12}.e_{13}. e_{10}.e_{11}. e_4.e_5.e_8$	Room Type Report	8

Test Case Partitioning Algorithm

The metric-based test case partitioning algorithm, which is shown in Figure 14, is applied now to the test cases generated so that we put similar test cases into one equivalence class.

We assume that εmax =1. We start with t_2, which is the longest test case (length = 12). STC = $\{t_1, t_3, t_4, t_5, t_6, t_7\}$ and the first element in the first equivalence class is t_2 ($TS_1 = \{t_2\}$). The distance between t_2 and the remaining test cases in STC is calculated, as shown in the following table, so that the test case with the minimal distance will be included in the same equivalence class TS_1.

Table 9: Distance calculated between t_2 and the remaining test cases

t_i	Similarity (t_2 , t_i)	Dissimilarity (t_2 , t_i)	e^{l}	td (t_2, t_i)
t_1	$2^{(-5)} = 0.03125$	7	$e^{(-12/8)}$	0.0488
t_3	$2^{(-2)} = 0.25$	5	$e^{(-12/10)}$	0.3765
t_4	$2^{(-2)} = 0.25$	7	$e^{(-12/8)}$	0.3905
t_5	$2^{(-4)} = 0.0625$	6	$e^{(-12/8)}$	0.0837
t_6	$2^{(-2)} = 0.25$	7	$e^{(-12/8)}$	0.3905
t_7	$2^{(-2)} = 0.25$	7	$e^{(-12/8)}$	0.3905

Note that, while calculating Similarity, we tried another approach: take the largest common subsequence starting from an input external event (i.e. $e_4.e_5.e_6.e_7.e_8$ for t_1 and t_2). This is because the use cases might include common behavior, described as requests from the environment (input events) and the system's answers to these requests. This common behavior can occur anywhere in the scenario, not only at the beginning, and we shall take it into account. Such behavior might indicate low cohesion in the use case, and thus the

need to factor it out as a function-level use case included in, or extending, the user-level use cases.

Table 10 explains in greater detail how we obtained the results shown in the Dissimilarity column.

Table 10: How to calculate the dissimilarity between two test cases

t_i	Dissimilarity (t_2, t_i)	Dissimilarity (t_2, t_i) descriptions	Minimal & elementary transformations
t_1	7	Change: $e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8$ To: $e_9.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_4.e_5.e_6.e_7.e_8$	- Change $e_1.e_2.e_3$ by $e_9.e_{10}.e_{11}$ - insert $e_{12}.e_{13}.e_{14}.e_{15}$
t_3	5	Change: $e_{19}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_4.e_5.e_8$ To: $e_9.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_4.e_5.e_6.e_7.e_8$	- insert $e_6.e_7$ to t_3 - move $e_{10}.e_{11}$ ahead - change e_{19} by e_9
t_4	7	Change: $e_{19}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_8$ To: $e_9.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_4.e_5.e_6.e_7.e_8$	- insert $e_4.e_5.e_6.e_7$ - move $e_{10}.e_{11}$ ahead - change e_{19} by e_9
t_5	6	Change: $e_{16}.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_{17}$ To: $e_9.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_4.e_5.e_6.e_7.e_8$	- change e_{16} by e_9 - change e_{17} by e_4 - insert $e_5.e_6.e_7.e_8$
t_6	7	Change: $e_{20}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_8$ To: $e_9.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_4.e_5.e_6.e_7.e_8$	- change e_{20} by e_9 - move $e_{10}.e_{11}$ ahead - insert $e_4.e_5.e_6.e_7$
t_7	7	Change: $e_{18}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_4.e_5.e_8$ To: $e_9.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_4.e_5.e_6.e_7.e_8$	- insert $e_6.e_7$ - insert $e_{14}.e_{15}$ - move $e_{10}.e_{11}$ ahead - change e_{18} by e_9

As a result, t_2 is a minimal distance from t_1 and both are in the same equivalence class, where $TS_1 = \{t_2, t_1\}$, $STC = \{t_3, t_4, t_5, t_6, t_7\}$ and ε is set to 0.0488.

Now, we must proceed with the following steps:

1. Calculate the distances between all the elements in STC and t_1.
2. For each element t in STC, set td (t,TS_1) to the min between (t,t_2) and (t,t_1).
3. Find an element t in STC, with $td(t,TS_1) < \varepsilon$.
    a. If one exists, add it to TS_1, change ε, STC, and calculate the distances between the remaining elements in STC and the new element in TS_1, and so on (similar to steps 1 and 2 above).
    b. Otherwise, start the process of creating TS_2.

The distance between t_1 and the remaining test cases in STC is calculated, as shown in Table 11, so that the test cases with the distance < ε will be included in the same equivalence class TS_1.

Table 11: Distance calculated between t_1 and the remaining test cases

t_i	Similarity (t_1, t_i)	Dissimilarity (t_1, t_i)	e^1	$td(t_1, t_i)$
t_3	$2^{(-2)} = 0.25$	9	$e^{(-10/8)}$	0.6446
t_4	$2^{(-1)} = 0.5$	7	$e^{(-1)}$	1.2875
t_5	$2^0 = 1$	8	$e^{(-1)}$	2.943
t_6	$2^{(-1)} = 0.5$	7	$e^{(-1)}$	1.2876
t_7	$2^{(-1)} = 0.5$	7	$e^{(-1)}$	1.2876

Table 12 explains in greater detail how we obtained the results shown in the Dissimilarity column.

Table 12: How to calculate the dissimilarity between two test cases

t_i	Dissimilarity (t_1 , t_i)	Dissimilarity (t_1, t_i) descriptions	Minimal & elementary transformations
t_3	9	Change: $e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8$ To: $e_{19}.e_{12}.e_{13}.e_{10}.e_{11}.e_{14}.e_{15}.e_4.e_5.e_8$	- remove $e_6.e_7$ - change $e_1.e_2.e_3$ by $e_{19}.e_{12}.e_{13}$ - insert $e_{10}.e_{11}.e_{14}.e_{15}$
t_4	7	Change: $e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8$ To: $e_{19}.e_{12}.e_{13}.e_{10}.e_{11}.e_{14}.e_{15}.e_8$	Change $e_1.e_2.e_3.e_4.e_5.e_6.e_7$ by $e_{19}.e_{12}.e_{13}.e_{10}.e_{11}.e_{14}.e_{15}$
t_5	8	Change: $e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8$ To: $e_{19}.e_{12}.e_{13}.e_{10}.e_{11}.e_{14}.e_{15}.e_8$	Change: $e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8$ by $e_{19}.e_{12}.e_{13}.e_{10}.e_{11}.e_{14}.e_{15}.e_8$
t_6	7	Change: $e_{20}.e_{12}.e_{13}.e_{10}.e_{11}.\ e_{14}.e_{15}.e_8$ To: $e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8$	Change $e_{20}.e_{12}.e_{13}.e_{10}.e_{11}.e_{14}.e_{15}$ by $e_1.e_2.e_3.e_4.e_5.e_6.e_7$
t_7	7	Change: $e_{18}.e_{12}.e_{13}.e_{10}.e_{11}.\ e_4.e_5.e_8$ To: $e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8$	- insert $e_6.e_7$ - change $e_{20}.e_{12}.e_{13}.e_{10}.e_{11}$ by $e_1.e_2.e_3$

As a result, $TS_1 = \{t_2, t_1\}$ stays the same, and we start generating TS_2 to include the first element which is t_3, the second longest test case. STC = $\{t_4, t_5, t_6, t_7\}$ and Table 13 calculates the distance between t_3 and the remaining test cases in STC.

Table 13: Distance calculated between t_3 and the remaining test cases

t_i	Similarity (t_3 , t_i)	Dissimilarity (t_3, t_i)	e^1	td (t_3, t_i)
t_4	$2^{(-7)} = 1/128$	2	$e^{(-10/8)}$	.00448
t_5	$2^{(-2)} = 0.25$	6	$e^{(-10/8)}$	0.4297
t_6	$2^{(-6)} = 1/64$	3	$e^{(-10/8)}$	0.0134
t_7	$2^{(-4)} = 1/16$	3	$e^{(-10/8)}$	0.0537

Table 14 explains more how we obtained the results shown in the Dissimilarity column.

Table 14: How to calculate the dissimilarity between two test cases

t_i	Dissimilarity (t_3 , t_i)	Dissimilarity (t_3, t_i) descriptions	Minimal & elementary transformations
t_4	2	Change: $e_{19}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_8$ To: $e_{19}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_4.e_5.e_8$	- insert $e_4.e_5$
t_5	6	Change: $e_{16}.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_{17}$ To: $e_{19}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_4.e_5.e_8$	- change e_{16} by $e_{19}.e_{12}.e_{13}$ - change e_{17} by $e_4.e_5.e_8$
t_6	3	Change: $e_{20}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_8$ To: $e_{19}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_4.e_5.e_8$	- change e_{20} by e_{19} - insert $e_4.e_5$
t_7	3	Change: $e_{18}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_4.e_5.e_8$ To: $e_{19}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_4.e_5.e_8$	- insert $e_{14}.e_{15}$ - change e_{18} by e_{19}

As a result, t_4 is a minimal distance from t_3 and both are in the same equivalence class, where $TS_2 = \{t_3, t_4\}$. Now $STC = \{t_5, t_6, t_7\}$ and ε is set to 0.00448.

Now, we have to do the following:

1. Calculate the distances between all the elements in STC and t_4.

2. For each element t in STC, set td (t,TS_2) to the minimal distance between (t,t_3) and (t,t_4).

3. Find an element t in STC with $td(t,TS_2) < \varepsilon$.

   a. If one exists, add it to TS_2, change ε, STC, and calculate the distances between the remaining elements in STC and the new element in TS_1, and so on (similar to steps 1 and 2 above).

   b. Otherwise, start the process of creating TS_3.

The distance between t_4 and the remaining test cases in STC is calculated as shown in Table 15, so that the test case with the distance < ε will be included in the same equivalence class TS_2.

Table 15: Distance calculated between t_4 and the remaining test cases

t_i	Similarity (t_4 , t_i)	Dissimilarity (t_4 , t_i)	e^1	td (t_4, t_i)
t_5	$2^{(-2)} = 0.25$	4	$e^{(-1)}$	0.3679
t_6	$2^{(-7)} = 1/128$	1	$e^{(-1)}$	0.00287
t_7	$2^{(-4)} = 1/16$	3	$e^{(-1)}$	0.06898

Table 16 explains in greater detail how we obtained the results shown in the Dissimilarity column.

Table 16: How to calculate the dissimilarity between two test cases

t_i	Dissimilarity (t_4 , ti)	Dissimilarity (t_4 , t_i) descriptions	Minimal & elementary transformations
t_5	4	Change: $e_{16}.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_{17}$ To: $e_{19}.e_{12}.e_{13}. e_{10}.e_{11}. e_{14}.e_{15}.e_8$	- change e_{16} by e_{19} - change e_{17} by e_8 - move $e_{12}.e_{13}$ ahead
t_6	1	Change: $e_{20}.e_{12}.e_{13}. e_{10}.e_{11}. e_{14}.e_{15}.e_8$ To: $e_{19}.e_{12}.e_{13}. e_{10}.e_{11}. e_{14}.e_{15}.e_8$	- change e_{20} by e_{19}
t_7	3	Change: $e_{18}.e_{12}.e_{13}. e_{10}.e_{11}. e_4.e_5.e_8$ To: $e_{19}.e_{12}.e_{13}. e_{10}.e_{11}. e_{14}.e_{15}.e_8$	- change e_{18} by e_{19} - change $e_4.e_5$ by $e_{14}.e_{15}$

As a result, $TS_2 = \{t_3, t_4, t_6\}$, since the distance between t_4 and t_6 < ε and therefore t_6 will be included in TS_2. STC = $\{t_5, t_7\}$ and ε is set to 0.00287. The distances between all the elements in STC and t_6 will be calculated in Table 17, in order to see whether or not we may include another test case in TS_2.

Table 17: Distance calculated between t_6 and the remaining test cases

t_i	Similarity (t_6 , t_i)	Dissimilarity (t_6 , t_i)	e^{I}	td (t_6 , t_i)
t_5	$2^{(-2)}$= 0.25	4	$e^{(-1)}$	0.36788
t_7	$2^{(-4)}$= 1/16	3	$e^{(-1)}$	0.06898

Table 18 explains in greater detail how we obtained the results shown in the Dissimilarity column.

Table 18: How to calculate the dissimilarity between two test cases

t_i	Dissimilarity (t_6 , t_i)	Dissimilarity (t_6 , t_i) descriptions	Minimal & elementary transformations
t_5	4	Change: $e_{16}.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_{17}$ To: $e_{20}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_8$	- change e16 by e20 - change e17 by e8 - move e12.e13 ahead
t_7	3	Change: $e_{18}.e_{12}.e_{13}.e_{10}.e_{11}.\ e_4.e_5.e_8$ To: $e_{20}.e_{12}.e_{13}.\ e_{10}.e_{11}.\ e_{14}.e_{15}.e_8$	- change $e_4.e_5$ by $e_{14}.e_{15}$ - change e_{18} by e_{20}

As a result, TS_2 = {t_3, t_4, t_6} stays the same and we start generating TS_3 to include the first element which is t_5. STC = {t_7} and Table 19 calculates the distance between t_5 and t_7.

Table 19: Distance calculated between t_5 and the remaining test cases

t_i	Similarity (t_5 , t_i)	Dissimilarity (t_5 , t_i)	e^{I}	td (t_5 , t_i)
t_7	$2^{(-2)}$ = 0.25	6	$e^{(-1)}$	0.5518

Table 20 explains more how we obtained the results shown in the Dissimilarity column.

Table 20: How to calculate the dissimilarity between two test cases

t_i	Dissimilarity (t_5 , t_i)	Dissimilarity (t_5 , t_i) descriptions	Minimal & elementary transformations
t_7	6	Change: $e_{18}.e_{12}.e_{13}.e_{10}.e_{11}.\ e_4.e_5.e_8$ To: $e_{16}.e_{10}.e_{11}.e_{12}.e_{13}.e_{14}.e_{15}.e_{17}$	- change e_{18} by e_{16} - change $e_4.e_5.e_8$ by $e_{14}.e_{15}.e_{17}$ - move $e_{10}.e_{11}$ ahead

As a result, t_5 is a minimal distance from t_7, which is less than εmax (1), and both test cases will be in the same equivalence class where $TS_3 = \{t_5, t_7\}$.

2) **Priority of Test Cases**

Once the generated test cases have been partitioned into three equivalence classes, it is better to prioritize them within each equivalence class according to their functional complexity (FC). More, and diverse, functionality in the system would lead to a larger portion of the system being involved in that usage, and therefore it will have higher priority than other test cases since it would cover more failures.

The functional complexity for each test case is shown in the following table:

Table 21: The functional complexities for the test cases.

TS_1	
t_1	$-\log_2 1/8$
t_2	$-\log_2 1/12$
TS_2	
t_3	$-\log_2 1/10$
t_4	$-\log_2 1/8$
t_6	$-\log_2 1/8$
TS_3	
t_5	$-\log_2 1/8$
t_7	$-\log_2 1/8$

Note that in TS_1 the FC for t_2 is greater than the FC for t_1, and therefore it will have a higher priority than t_1. The new $TS_1 = \{t_2, t_1\}$ as an ordered set. In TS_2, the FC for t_3 is greater than the FCs for t_4 and t_6. Therefore, t_3 has a higher priority than t_4 and t_6; however, t_4 and t_6 have the same FC. The new $TS_2 = \{t_3, t_4, t_6\}$ or $\{t_3, t_6, t_4\}$. Finally, t_5 and t_7 have the same FC in TS_3. The new TS_3 can either be $\{t_5, t_7\}$ or $\{t_7, t_5\}$ as an ordered set.

6.1.6 Test-Case Execution (Update Reservation)

In our work here, we elected to use a tool suite for the Update Reservation function to illustrate the proposed testing strategy. In order to offer a complete package for testing software, we used the Rational Suite to execute the test cases generated and selected by the proposed testing strategy. Test-case execution comprises four main steps in the Rational Suite.

First, the system requirements have to be created in a Rational RequisitePro project (Shukla 2005). Any mistake in capturing the requirements of the system will cost a great deal in later stages of the system development life cycle. Through tool integration, requirements are made accessible to guide and define the system architecture, the testing and the documentation activities (Shukla 2005).

Then, Rational Administrator, which is also included in the Rational Suite, is responsible for creating what are called "Rational Centralized" projects, and configuring them for software development teams. It is used to integrate the Rational tools, which are RequisitePro, TestManager and Robot (Cammarano 2001).

After creating the project with Rational Administrator, a Test Plan is defined by Rational TestManager, which organizes the manual testing workflow, logs the steps and results of manual scripts as they are performed, and generates automated reports of manual test results and measures (IBM 2004).

The last step in test case execution is the automation of functional testing through Rational Robot. This kind of automatic testing should be carried out individually for each function, but it can be executed for all the functions together through TestManager.

Figure 23 illustrates the introduction of the Hotel Reservation System requirements in a Rational RequisitePro project for testing and tracing by TestManager.

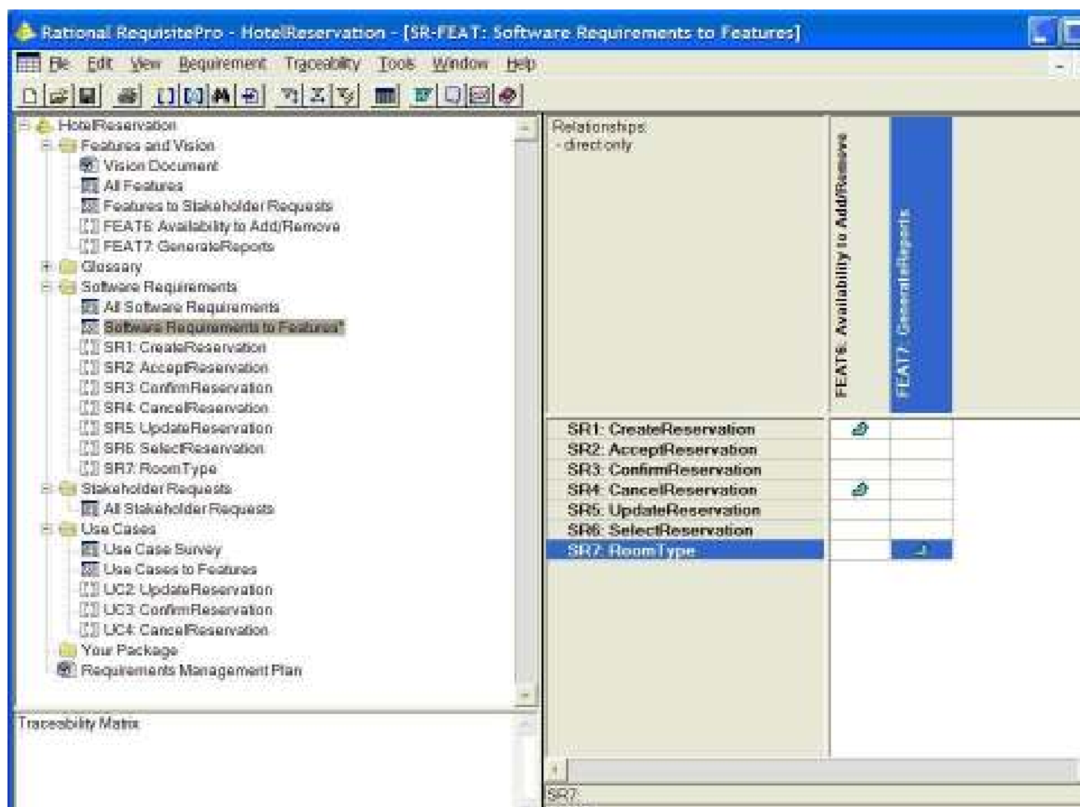

Figure 23: Hotel reservation system requirements management

Figure 24 introduces a part of the test plan we adopted here to assess the progress of the Hotel Reservation System.

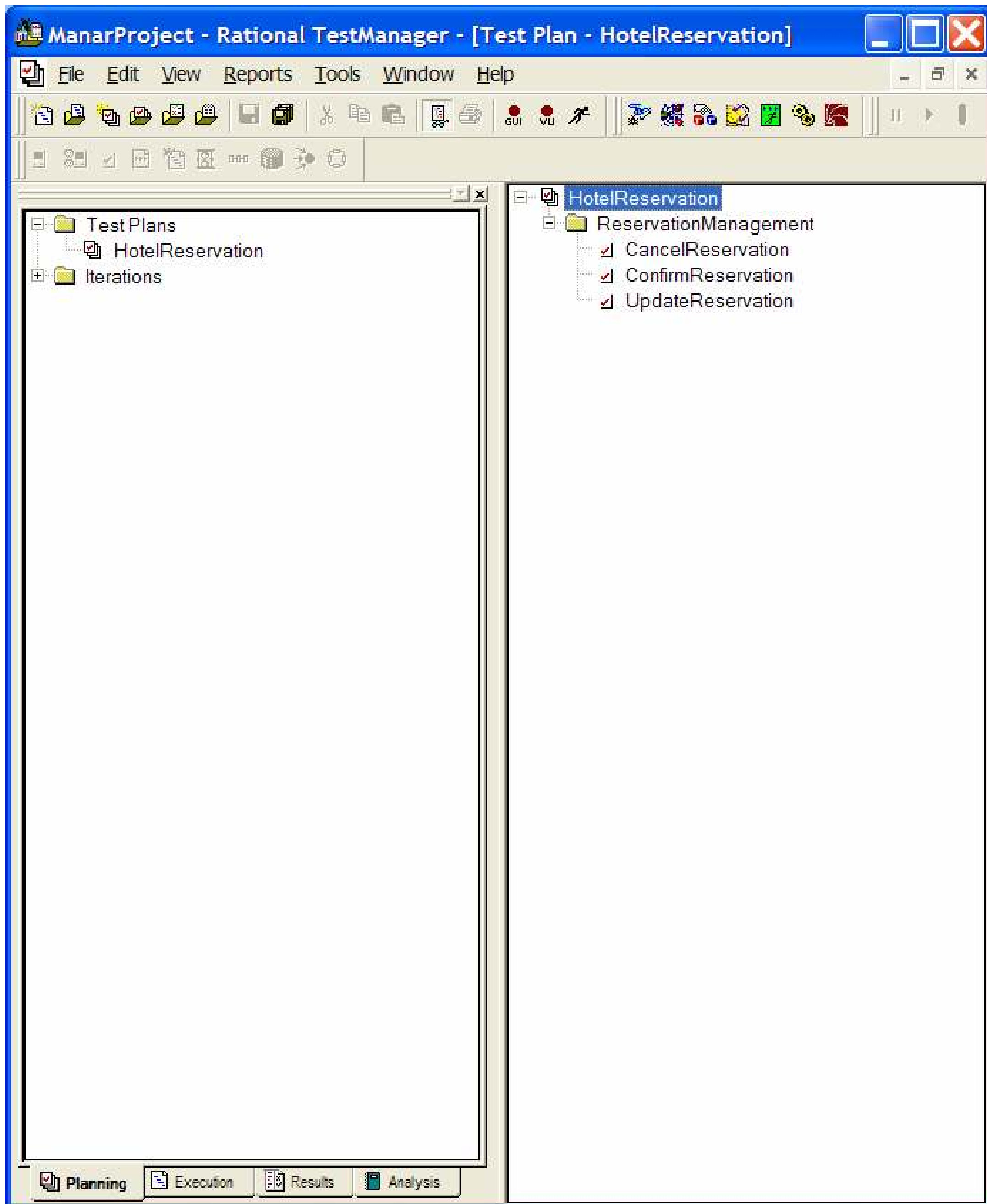

Figure 24: Rational TestManager for the Hotel Reservation System

The script presented in Figure 25 is the result of the automatic function testing produced by Robot. Tested here is the Update a Reservation function of the Hotel Reservation System.

This automatic function testing is linked to its corresponding test case in the Test Plan presented above by TestManager, and will be automatically executed when the tester runs the Test Plan.

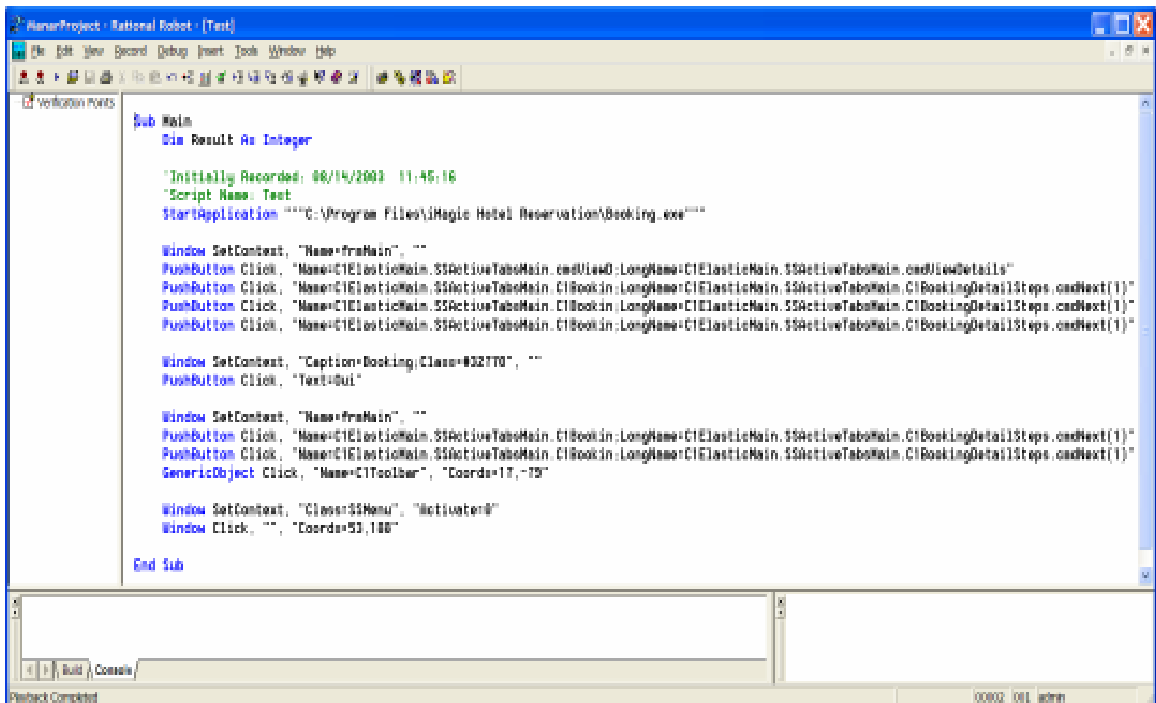

Figure 25: Results of Automatic Function Testing produced by Robot

The screen in Figure 26 provides the Actual Result and the Interpreted Result of the Update Reservation function test.

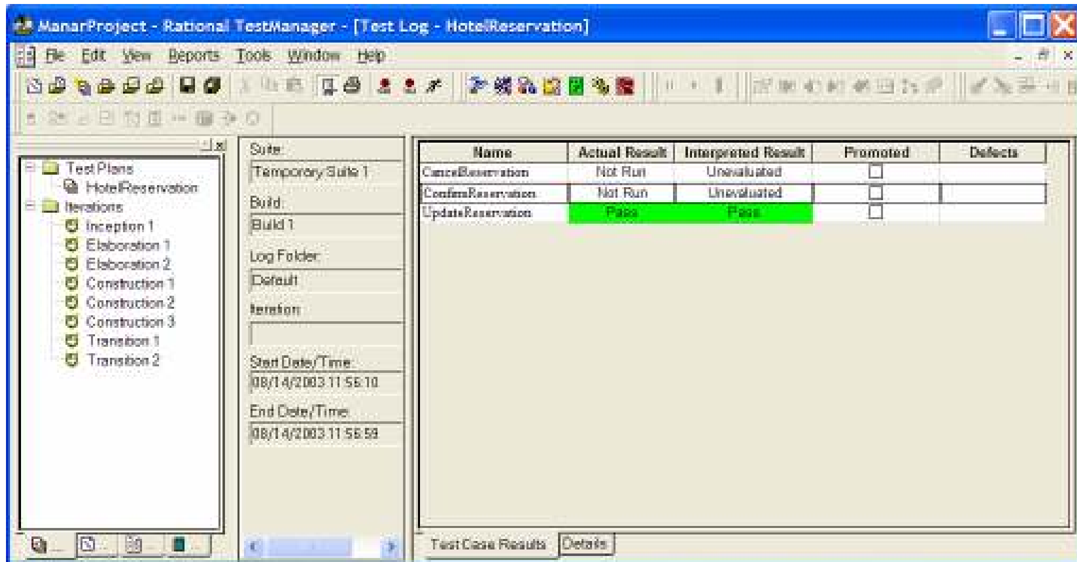

Figure 26: Results produced by the proposed testing procedure

Note that the Rational Suite is a useful product for executing these test cases, which are generated and selected through the proposed testing algorithms; however, it does not specify the details of the test cases (IBM 2004). For testing purposes, it classifies test requirements as black-box and test-case requirements as white-box. Rational Suite is only concerned with test requirements. This means that the test requirements are organized in Requisite Pro and linked by TestManager, where the development team can track test requirement revisions. It is the task of the tester to use his or her skill and experience to introduce and address the details required for test script design, test data design and test result verification purposes from the existing test requirements information.

6.2 Entropy-based Reliability Assessment in COSMIC-FFP

To assess reliability prior to implementation, it is important to understand the complex interactions among entities in software: this can be achieved by modeling their designs

(the entities, their interactions and the probabilities of the interactions) as a Markov system, and assessing the level of reliability by calculating the probabilities of their interaction. Such an approach yields results for both the time-dependent evolution of the system and the steady state of the system.

The research reported in this thesis extends the COSMIC-FFP and reliability assessment model (Ormandjieva 2002) to the component-based system context. In essence, each component of the system is modeled by a discrete time Markov chain and visualized through a finite-state machine. Then, a probabilistic analysis by Markov chains can be performed to analyze the component's entropy based on its behavior specification with extended state machines. The approach presented here of applying the Markov model in the COSMIC-FFP context will be illustrated with the Railroad Crossing case study. The reliability of the whole system is computed from the system's architectural configuration and the reliability of the individual components. The purpose of the reliability assessment is to compare alternative component-based system designs, and to assist in making the architectural changes to the evolving system.

6.2.1 Markov Model and State Machine Diagrams

A state-machine diagram (Primer 2004), referred to as a state diagram in UML 2, is a UML behavioral diagram. It is used to model the dynamic behavior of individual objects and depict the various states that an object may be in and the transitions between those states.

A state (Primer 2004) represents a stage in the behavior pattern of an object, and it is possible to have initial states and final states. A transition (Primer 2004) is a progression from one state to another and will be triggered by an event that is either internal or external to the object. See Figure 27 for an example of a state-machine diagram that models the behavior of the object "Train" for the following railroad system case study (Alur 1999), (Vangalur, Alagar and Periyasamy 1998), where more than one train can cross a gate simultaneously, through multiple parallel tracks. According to the train's destination, it can independently choose the gate it will cross. Each gate is controlled by one controller which must be active all the time to close and open the gate for the railroad crossing. A train enters the crossing within an interval of time units after informing the controller that it is approaching. It also informs the controller that it is leaving the crossing within some time units of sending the approaching message. In response, the controller commands the gate to close when it receives a message from the first train entering the crossing to make sure that no other train can cross the railroad at the same time. It also instructs the gate to reopen when it receives a message from the last train when it is leaving the crossing.

The state machine in Figure 27 models the behavior of the train at a point where it has five states: an initial state (the black circle), "idle", "toCross", "cross", "leave" and no final state, since this is a continuous operation. Note that "Near" is a triggering event that makes the train move from the "idle" state to the "toCross" state. Some of the transitions have conditions, such as (time units> entrance time > 0), which have to be satisfied in order for the object to move to other states. The general description for the following

state machine diagram is that, as the train approaches a gate, it sends a "Near" message to the gate controller. Once the train leaves the gate, it sends an "Exit" message to the gate controller. It is to be noted that the proposed reliability prediction approach is done in the specification level, where the visible states are specified. We are not drawing the detailed state machine diagrams which usually are included in the design level.

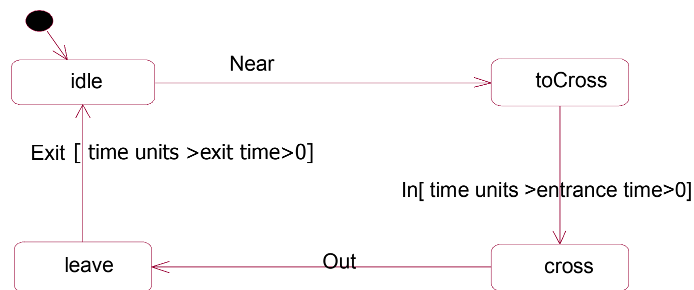

Figure 27: Train state machine

Which transition will be triggered from one state is the same as in a random walk; based on this, the Markov model can be used to analyze the reliability of state machine software (Ormandjieva 2002). Therefore, the prediction of reliability is derived from the steady state of the Markov system. The mapping of the Train object to a Markov system is shown in Figure 28, with a probability of 1 for each event, since there is only one event from each state. In a case where there are two events from one state, then each event would have a probability of ½. P_{12} represents the transition probability that the event will be triggered, and the move is accordingly made from state S_1 to state S_2..

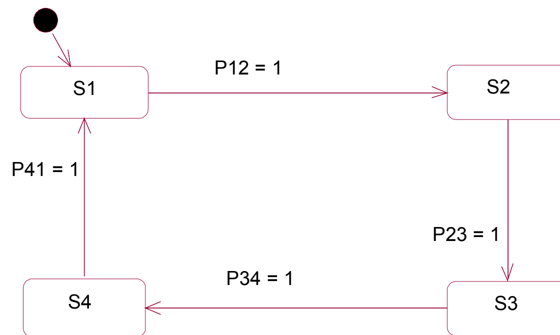

Figure 28: Train state diagram with its transition probabilities P_{ij}

Now, the transition matrix *P* (see Table 22) can be built from this state-machine diagram, and it is a matrix *P*, the ij^{th} entry of which is P_{ij}. Note that the entries in each row add up to 1.

Table 22: Transition matrix *P* for Train object

	S_1	S_2	S_3	S_4
S_1	0	1	0	0
S_2	0	0	1	0
S_3	0	0	0	1
S_4	1	0	0	0

6.2.2 COSMIC-FFP and Sequence Diagrams

A sequence diagram (Primer 2004) is a UML structural diagram that models the flow of logic within the system in a visual manner, enabling both the documentation and validation of the user's logic, and is commonly used for both analysis and design purposes. The sequence diagram is the most popular UML artifact for dynamic modeling, and focuses on identifying the behavior within the system (Primer 2004). It consists of a group of instances (represented by life lines or dashed lines) and the messages they exchange during the interaction (see Figures 29 and 30) that are the sequence diagrams derived from the Railroad Crossing case study for FSM purposes. Both have three

objects, namely Train, Controller and Gate, which interact by sending messages to each other. While measuring the functional size of software using COSMIC-FFP, the sequence diagrams are drawn to define the interactions between the software and its environment and within the software, as illustrated in Figure 2. In COSMIC-FFP, the environment is represented by the users interacting with the software, such users being humans, engineering devices or other software applications. Within the software, the interactions deal with the data read from, or send the data to, persistent storage. Going back to the railroad case study, the controller is the software that has a boundary where the trains interact with the controllers through sensors (many-to-many relationships) and the controllers communicate with the gates through actuators (one-to-one relationships).

In the RUP context (Kruchten and Philippe 2000), the functional processes used in COSMIC-FFP can represent the set of scenarios for the software. For example, in the railroad system, the first sequence diagram (Figure 29) shows that, when a train arrives, a Near message is sent to the controller. The controller then instructs the gate to lower, and, in return, the gate goes down and the train enters the crossing. This process of allowing the train to cross the railroad is considered as a functional process, and is triggered by sending a "Near" message. Similarly, "train leaves crossing" (Figure 30) is a scenario containing a sequence of events between the train and the controller, and this scenario also contains a sequence of events within the system (controller, in this case). Therefore, for each functional process, its subprocesses and its triggering events are sequences of events (or data movements).

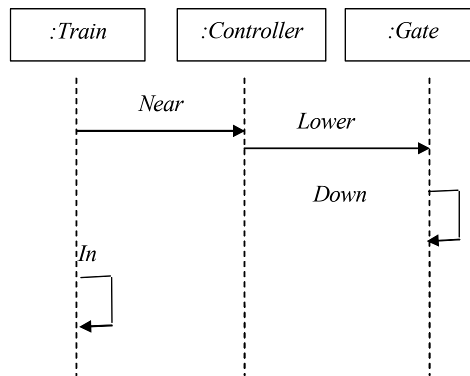

Figure 29: Train Enters Crossing sequence diagram

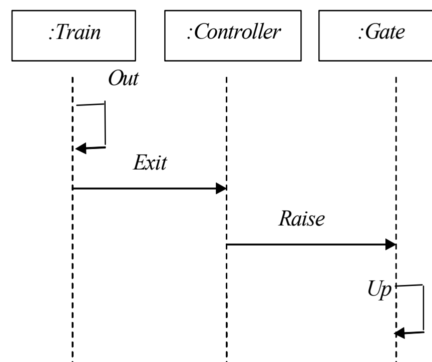

Figure 30: Train Leaves Crossing sequence diagram

6.2.3 Analysis of Linkages across Models

The FSM method COSMIC-FFP can be linked to UML 2.0 state diagrams for modeling behavior. This allows for probabilistic reliability modeling based on discrete Markov chains, since a Markov model is based on state diagram descriptions. The linkages between COSMIC-FFP and the Markov model can be analyzed, since the two have something in common: UML diagrams.

The correspondence of COSMIC-FFP to UML state diagrams requires a mapping of COSMIC-FFP concepts (such as boundary, layer, functional process, triggering event,

data group, data movement and attributes) to state diagram notation. The reliability requirements for autonomic elements and component-based systems have to be specified formally and mapped to system behavior, so that the achievement of reliability can be monitored automatically.

Analysis of Table 23 reveals that the same conceptual level is used for both COSMIC-FFP and UML 2 state-machine diagrams; however, the terms used in the data movements of COSMIC-FFP and in the events of state-machine diagrams have different labels. A summary of the terms used in COSMIC-FFP and state-machine diagrams that have similar meanings is presented in Table 23. For example, in COSMIC-FFP, data movements are classified in four categories: Entry, Exit, Read and Write. The term corresponding to the data movement and its categories that is used in state-machine diagrams is “event”, with two classifications: internal and external.

In addition, the architecture of the system corresponds to the layers in the COSMIC-FFP context where a layer is a result of the functional partitioning of the software environment such that all included functional processes perform at the same level of abstraction. One layer may contain one or more objects, or more than one component, that interact with one another in one layer. Also, data groups, which represent the set of data attributes in COSMIC-FFP, correspond to the term “objects” that is used in state-machine diagrams. These diagrams explore the detailed transitions between states as the result of events (either external or internal) for only one object (Primer 2004). Some additional expressiveness of the state-machine diagrams could be taken into account. For instance, an external event can produce a set of internal events, and this relationship (between

internal and external events) can probably affect the software functional size and should be described in the sequence diagrams in terms of Entry and/or Exit data movements, which may in turn produce a set of Read and/or Write data movements. Another issue which can be carefully analyzed is the possible additional readings that may arise as a result of pre- and/or post-conditions, where their operands can refer to other objects associated with the events of a sequence or state diagram. This may affect the reliability prediction calculations based on Markov chains and its probabilities where conditional probabilities can be applied. Other terms used in both models, such as those interacting with the software, the software boundary and the set of user requirements, have the same labels.

Table 23: Similarity between COSMIC-FFP and the state machine diagram concepts

Concepts	COSMIC-FFP (Data Movement) terms	State Machine Diagrams (Events) terms
System Architecture	Layer	One or more objects
Humans or things interacting with the software	Software users	Software users
Between the environment and the software	Software boundary	Software boundary
Set of User Requirements	Functional Process	Sequence of Events (Scenario)
Data which are part of the interaction	Data groups	Objects
External Input (from the environment)	Triggering event	External event
External Input (from the environment)	Entry data movement	External event
Output (to the environment)	Exit data movement	External event
Internal Input (within the software)	Read data movement	Internal event
Internal Input (within the software)	Write data movement	Internal event

From Table 23, it can be seen that COSMIC-FFP and UML state-machine diagrams have similar concepts, and this has motivated further investigation into the possibility of

deriving state-machine diagrams from COSMIC-FFP notations. Note that, while sequence diagrams have been used in the COSMIC-FFP measurement method to explore the behaviors of one or more objects over a given period of time, the state-machine diagrams for each object in COSMIC-FFP can be used to explore all their details (Primer 2004).

According to the COSMIC-FFP definitions given in its measurement manual and the sequence diagrams drawn based on it, state-machine diagrams can be derived from these sequence diagrams. COSMIC-FFP measurements can be mapped to UML 2.0 state diagrams using the technique proposed in (Vasilache and Tanaka 2004) and illustrated with state-machine diagrams from multiple interrelated scenarios (or sequence diagrams). A number of authors have discussed the way to transform a set of scenarios (or sequence diagrams) into state-machine diagrams, for example (Koskimies, Mannisto, Systa and Tuomi 1998), (Whittle and Schumann 2000) and (Ryser and Glinz 2000). However, the work proposed in (Vasilache and Tanaka 2004) includes the steps and rules for deriving state-machine diagrams from multiple scenarios with regard to the relationships between them. These rules are summarized as follows:

Step 1. Identify and represent all single scenarios as sequence diagrams.

Step 2. Identify and represent the relationships among all scenarios as dependency diagrams based on the time dependencies between scenarios, their cause-effect dependencies and their generalization dependencies. The dependency diagram must have a single start point, which is the initial scenario, but it can have several end points.

Step 3. Synthesize the state-machine diagrams based on the information acquired in the previous two steps.

Step 4. Refine the final state machines and approve the consistency between scenarios and state machines in order to make sure that the behavior of the final state-machine diagrams reflects the information contained in the scenarios.

Now that the linkage between COSMIC-FFP and the UML 2.0 state diagrams has been identified, the state-machine diagrams can be derived accordingly. Going back to the Railroad Crossing case study, step 1 has already been performed in section 6.2.2, where the sequence diagrams are drawn for COSMIC-FFP purposes. Figure 31 shows the dependency diagram needed in step 2, i.e. the relationships among the scenarios (or sequence diagrams) and the order of execution.

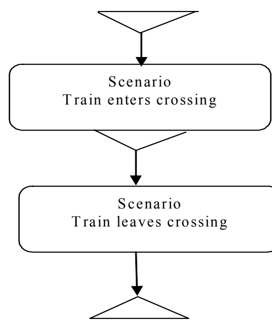

Figure 31: Dependency diagram

One scenario is represented as a rounded rectangle, with connectors for the start point and end point. The initial scenario is "Scenario train enters crossing". At that point, the train crosses the railroad, and the next scenario starts to be executed, which is "Scenario train leaves crossing". This is simply a dependency diagram, where there are no alternative scenarios.

Step 3 uses the information obtained in the previous steps to derive the corresponding state-machine diagrams. Note that each sequence diagram shows the sequence of events

(or data movements in the COSMIC-FFP context). Each event is a tuple: (O_i, O_j, M_{ijk}), where O_i and O_j belong to the set of objects involved in the software and M_{ijk} is the message that is exchanged between them. Therefore, the sequence diagram in Figure 29 has the following set of tuples = {(Train, Controller, Near), (Train, Train, In), (Controller, Gate, Lower), (Gate, Gate, Down)}, and the sequence diagram in Figure 30 has the following set of tuples = {(Train, Train, Out), (Train, Controller, Exit), (Controller, Gate, Raise), (Gate, Gate, Up)}. There are three objects involved in each scenario, and we can therefore synthesize three state-machine diagrams (one for each object).

For each object, one initial state-machine diagram can be created for each scenario, and the final state-machine diagram can then be synthesized from all the state-machine diagrams, based on the information in the dependency diagrams. The state-machine diagram for the Train object in Figure 27 is obtained from the two initial state-machine diagrams shown in Figures 32 and 33.

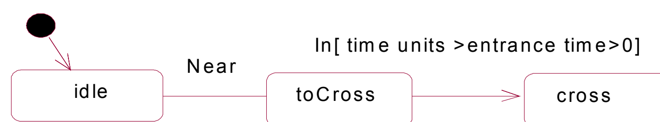

Figure 32: Initial state-machine diagram from Figure 29

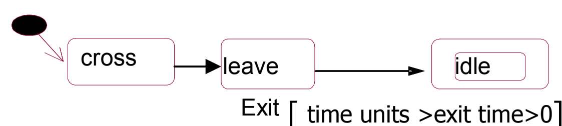

Figure 33: Initial state-machine diagram from Figure 30

Similarly, state-machine diagrams for the objects Controller and Gate are created as shown in Figures 34 and 35.

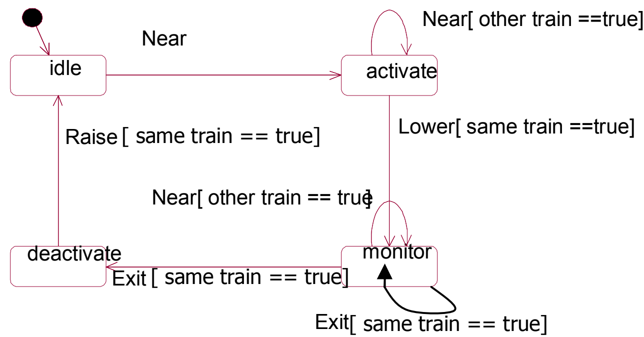

Figure 34: Controller state-machine diagram

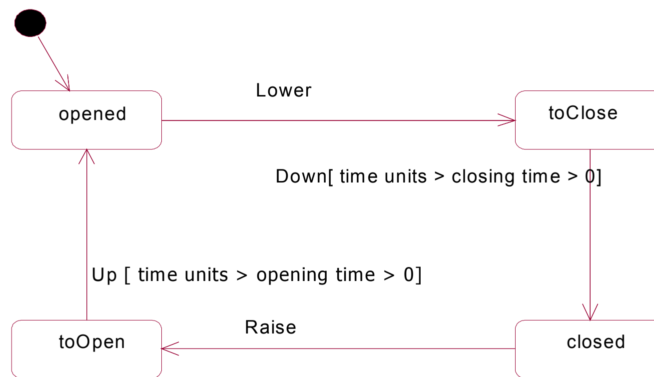

Figure 35: Gate state-machine diagram

The COSMIC-FFP method recognizes that a component-based system can be measured accurately only when the sizes of the components are measured separately, especially when they are in different layers or are different peer items. To size the total functionality to be delivered by a component-based system, guidance is needed to help decide if FURs or the component comprises one or more layers or peer items. The measurement analyst has to identify the requirements of each component that will be created or modified,

identify the functional processes involved for each, identify and count the data movement types.

6.2.4 Reliability Model for a Component-based System

The research reported in this thesis extends the COSMIC-FFP and reliability assessment model (Ormandjieva 2002) to the component-based system context. In essence, each component of the system is modeled by a discrete-time Markov chain and visualized through a finite-state machine. Next, the reliability of the whole system is computed from the system's architectural configuration and the reliability of the individual components, as stated in (Peters and Pedrycz 2000). The purpose of this computation is to compare alternative component-based system designs, and to assist in making the architectural changes to the evolving system.

The ability to take into account the functionality measures early on where sequence diagrams are derived, such as with COSMIC-FFP, makes it possible to consider the uncertainty in the operational profile (i.e. the uncertainty of environmental events) in addition to the uncertainty in component failure behavior, based on:

- Knowledge of the software architecture requirements (corresponds to the layers in COSMIC-FFP where the behavior of software objects can be modeled with state diagrams within a layer, and then the Markov model is applied in each layer).
- Prediction of component reliability, where a component is a group of interacting software objects whose behaviors are modeled in the extended state machine.
- Probabilities of state transitions of an object due to events (environmental or internal to the object) are calculated as shown in (Ormandjieva 2002), where the environmental events are considered random and not controlled by system laws.

6.2.4.1 Reliability of a Component

The state-machine diagrams of the objects are graphical representations of the corresponding extended-state machines, and they are synthesized from the interrelated sequence diagrams that are drawn in the COSMIC-FFP context. The product machine of the objects' state machines belonging to one component results in an extended-state machine describing the behavior of the component. The states of each state-machine diagram (object/component), their transitions and the probabilities for the transitions are formalized as a Markov system. The theoretical foundation for the proposed reliability prediction model is the property of Markov processes stating that, given the current state of the object/component, the future state of the object/component is independent of its history. The analysis of Markov models yields results for both the time-dependent evolution of the object/component and the steady-state properties of the object/component. An algebraic representation of a Markov model is a matrix, called a transition matrix, in which the rows and columns correspond to the states, and the entry p_{ij} in the ith row, j^{th} column is the transition probability for being in state j at the stage following state i. The initial probabilities p_{ij} for all the transitions in the state machine of an object in one component are calculated. The algorithm for calculating such probabilities for a state is based on the following assumptions: 1) all external events that can occur in that state have identical and independent probability distributions; 2) all internal events that can occur in that state have the same probability of occurring; and (3) these external and internal events are, in general, different. We assume the most common stochastic queuing model for the arrival time of the external events, namely a Poisson distribution.

Let Ext_i be the set of external events and Int_i the set of internal events which can trigger a change of $state_i$ to another state. The transition probabilities are calculated as follows:

- If Ext_i is empty, then the probability of a transition due to an internal event is $p_{ij} = 1/| Int_i|$.
- Similarly, the probability of transition due to an external event is $p_{ij} = 1/| Ext_i|$ when Int_i is empty.
- If both Ext_i *and* Int_i are non-empty, the probability of a transition due to an external event is calculated first as follows: $p^{ext}_{ij} = \Sigma_i (1/n) / |Ext_i|$ (where n is the total number of external events for the component, and 1/n is the equal probability of each external event occurring). Next, the probability of an internal event is calculated: $p^{int}_{ij} = (1- \Sigma_i (p^{ext}_{ij}) / |Int_i|$.
- When there is more than one transition with identical source and destination states, the above transitions are substituted in the Markov model by one whose probability p_{ij} is equal to the sum of the probabilities of the corresponding transitions.

Note: The following property holds for the calculated probabilities:

$$\Sigma_j p_{ij} = 1$$

We construct the Markov model of a component in two steps. In the first step, we construct the Markov models for its objects. In the second step, we construct the Markov model for the whole component consisting of synchronously interacting objects. The synchronous product machine dynamically changes as and when …? joins or leaves the component, and hence the transition probability matrices also change, and should be recomputed. The detailed description of the algorithms for constructing Markov matrices is given in (Ormandjieva 2002).

The reliability prediction for a component composed of *n* objects can be defined as the level of certainty quantified by excess entropy, as follows (Alagar and Ormandjieva 2002):

$$Reliability\ (Component) = \Sigma_{i=1,k} H_i - H$$

$$H = -\Sigma_i v_i \Sigma_j p_{ij} \log_2(p_{ij})$$

where H is an entropy which quantifies the level of uncertainty in the Markov chain corresponding to the whole component; H_i is a level of uncertainty in a Markov chain corresponding to an object; v is a steady state distribution vector for the corresponding Markov chain; and the p_{ij} values are the transition probabilities in the extended-state machines modeling the behaviors of the ith object. For a transition matrix P, the steady state distribution vector v satisfies the property $v*P = v$, and the sum of its components v_i is equal to 1. A steady state or equilibrium state is one in which the probability of being in a state before and after transitions is the same as time progresses. It must be mentioned that the steady-state vector does not always exist. The theorem for this purpose states that, if P is a positive transition matrix (no zero entries) and p_0 is any initial probability vector, then $P_k p_0$ approaches the steady-state vector as k approaches infinity (Strook 2005).

H is related exponentially to the number of paths that are "statistically typical" of the Markov system. The higher the entropy value, the more sequences must be generated to accurately describe the asymptotic behavior of the Markov system. A higher value of the reliability measure implies that there is less uncertainty in the model, and thus a higher level of software reliability. In consequence, the analysis of Markov chains yields results for both the time-dependent evolution of the system and the steady-state properties of the system.

6.2.4.2 Reliability of a System

The reliability measures of the components are used to determine the reliability of the whole component-based system measured in the COSMIC-FFP context, based on the configuration of its n components.

There are two interesting limit cases of the p-out-of-n configuration, namely, parallel and serial structures (Peters and Pedrycz 2000). We have a parallel configuration when there is at least one component is necessary to ensure that the entire component-based system functions. Assuming the independence of failures of the corresponding components, the reliability is calculated as follows: $R(t) = 1 - \Pi_{k=1,n} (1 - R_k(t))$

In a serial reliability structure the functioning of the system is ensured while all the components are functioning properly. In this case, the reliability is given by the following formula: $R(t) = \Pi_{k=1,n} R_k(t)$

In this reliability analysis, all the components are considered to function independently, and therefore it is assumed that there is independence of failures of the corresponding components.

6.2.5 Case Study: Railroad System

The feasibility of our reliability prediction approach is demonstrated now, with the following case study. Referring back to the Railroad Crossing case study (Train-Gate-Controller), in order to predict the reliability of the component-based system, the Controller and Gate objects must be mapped to their corresponding Markov systems, as

was done previously with the Train object (Figure 27). The mapping of the Controller object to a Markov system is shown in Figure 36, with a probability of 1 for two events, since there is only one event coming from the C_1 and C_4 states and with a probability of ½ for each of the two external events causing transitions from state C_2, and a probability of 1/3 for each of the external three events coming from C_3 (See Figure 34). The mapping of the Gate object to a Markov system is shown in Figure 37, and is similar to the Train object mapping.

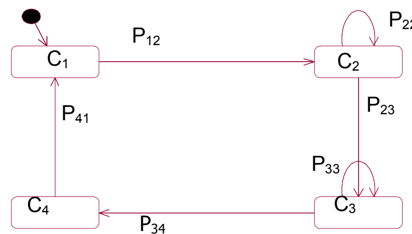

Figure 36: Controller state diagram with its transition probabilities P_{ij}

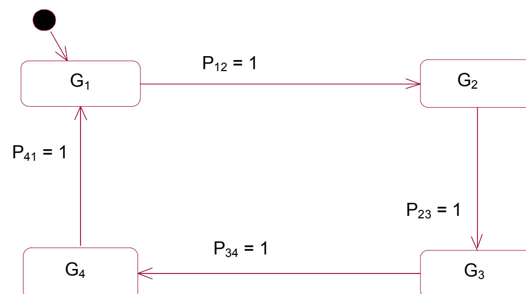

Figure 37: Gate state diagram with its transition probabilities P_{ij}

Now the corresponding transition matrices P (see Tables 24, 25) can be built from these state-machine diagrams, and each is a matrix P whose ijth entry is P_{ij}. Note that the entries in each row add up to 1.

Table 24: Transition matrix P for the Controller object

	C_1	C_2	C_3	C_4
C_1	0	1	0	0
C_2	0	½	½	0
C_3	0	0	2/3	1/3
C_4	1	0	0	0

Table 25: Transition matrix P for the Gate object

	G_1	G_2	G_3	G_4
G_1	0	1	0	0
G_2	0	0	1	0
G_3	0	0	0	1
G_4	1	0	0	0

In order to calculate H, the level of uncertainty of the Markov system corresponding to a subsystem or component, the synchronous product of Train, Controller and Gate is built. The interaction between two objects is due to shared events. The interaction behavior of the objects is completely described by their synchronous product machine M (see Figure 38).

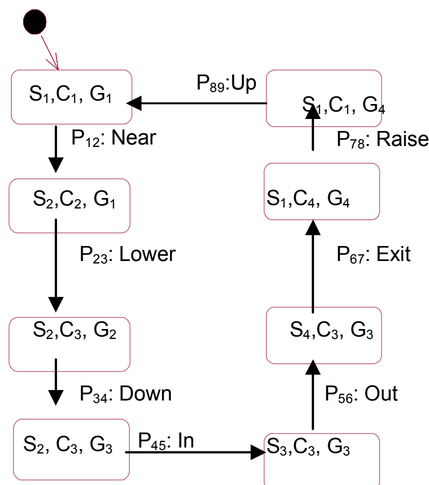

Figure 38: Synchronous product of Train, Controller and Gate

The corresponding transition matrix P of a synchronous product (see Table 26) can be built from the above state-machine diagram. Note also that the entries in each row add up to 1.

Table 26: Synchronous product of Train, Controller and Gate

	$S_1C_1G_1$	$S_2C_2G_1$	$S_2C_3G_2$	$S_2C_3G_3$	$S_3C_3G_3$	$S_4C_3G_3$	$S_1C_4G_3$	$S_1C_1G_4$
$S_1C_1G_1$	0	1	0	0	0	0	0	0
$S_2C_2G_1$	0	0	1	0	0	0	0	0
$S_2C_3G_2$	0	0	0	1	0	0	0	0
$S_2C_3G_3$	0	0	0	0	1	0	0	0
$S_3C_3G_3$	0	0	0	0	0	1	0	0
$S_4C_3G_3$	0	0	0	0	0	0	0	0
$S_1C_4G_3$	0	0	0	0	0	0	0	1
$S_1C_1G_4$	1	0	0	0	0	0	0	0

The next step will be to calculate *H* for each object and for the synchronous product based on their steady vectors. For a transition matrix *P*, the steady state distribution vector *v* satisfies the property *v*P = v*.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad |[wxyz]| = |[wxyz]|$$

where w = 0.25, x = 0.25, y = 0.25 and z = 0.25

So, the steady-state vector of the Train object is [0.25, 0.25, 0.25, 0.25] and its *H* =

-((0.25 * (0log0 + 1log1 + 0log0 + 0log0) +
(0.25 * (0log0 + 0log0 + 1log1 + 0log0) +
(0.25 * (0log0 + 0log0 + 0log0 + 1log1) +
(0.25 * (1log1 + 0log0 + 0log0 + 0log0))
= 0

It is the same calculation for the Gate object, where its steady-state vector is [0.25, 0.25, 0.25, 0.25] and its $H = 0$. Having an entropy value of 0 means that there is no level of uncertainty in a Markov system for that object, and therefore its behavior is not complicated and does not require the generation of more sequences to describe it.

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 2/3 & 1/3 \\ 1 & 0 & 0 & 0 \end{pmatrix} \quad [wxyz] = \|[wxyz]\|$$

where w = 0.25, x = 0.25, y = 0.25 and z = 0.25

So, the steady-state vector of the Controller object is [0.182, 0.364, 0.273, 0.182] and its H =

$$- ((0.182 * (0\log0 + 1\log1 + 0\log0 + 0\log0) +$$
$$(0.364*(0\log0 + \tfrac{1}{2}\log\tfrac{1}{2} + \tfrac{1}{2}\log\tfrac{1}{2} + 0\log0) +$$
$$(0.273*(0\log0+0\log0+1/3\log1/3+2/3\log2/3) +$$
$$(0.182 * (1\log1 + 0\log0 + 0\log0 + 0\log0))$$
$$= 0.615$$

Finally the steady-state vector for the synchronous product machine of the Train-Controller-Gate component is 0 and its $H = 0$.

The reliability for the Railroad Crossing (Train-Controller-Gate) case study is therefore:

$$(H_{\text{Train}} + H_{\text{Controller}} + H_{\text{Gate}}) - H_{\text{Train-Controller-Gate}}$$
$$= (0 + 0.615 + 0) – 0$$
$$= 0.615$$

Another component-based system configuration for the same case study can be two trains – and one controller and one gate – where there are two railroad tracks and one controller to monitor the gate. The synchronous product of Train, Gate and Controller would differ

from that in the previous case (Figure 38). Figures 39 and 40 show the result of the synchronous products of the objects in this configuration.

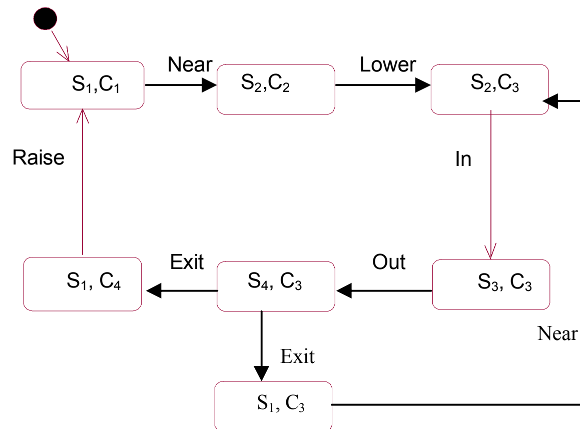

Figure 39: Synchronous product of Train and Controller (second configuration)

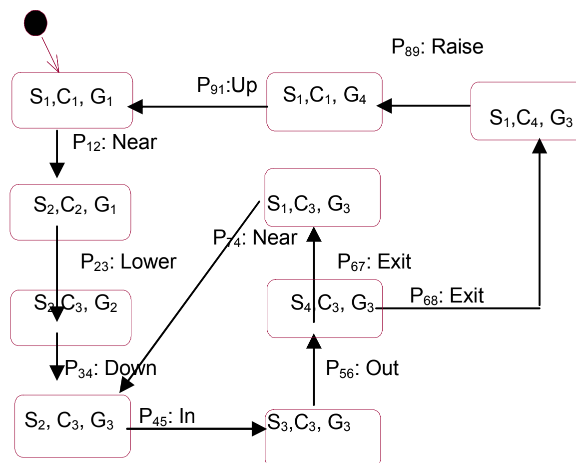

Figure 40: Synchronous product of Train, Controller and Gate (second configuration)

The corresponding transition matrix P of a synchronous product (see Table 27) can be built from the above state-machine diagram. Note, too, that the entries in each row add up to 1.

Table 27: Transition matrix P for a synchronous product

	$S_1C_1G_1$	$S_2C_2G_1$	$S_2C_3G_2$	$S_2C_3G_3$	$S_3C_3G_3$	$S_4C_3G_3$	$S_1C_3G_3$	$S_1C_4G_3$	$S_1C_1G_4$
$S_1C_1G_1$	0	1	0	0	0	0	0	0	0
$S_2C_2G_1$	0	0	1	0	0	0	0	0	0
$S_2C_3G_2$	0	0	0	1	0	0	0	0	0
$S_2C_3G_3$	0	0	0	0	1	0	0	0	0
$S_3C_3G_3$	0	0	0	0	0	1	0	0	0
$S_4C_3G_3$	0	0	0	0	0	0	½	½	0
$S_1C_3G_3$	0	0	0	1	0	0	0	0	0
$S_1C_4G_3$	0	0	0	0	0	0	0	0	1
$S_1C_1G_4$	1	0	0	0	0	0	0	0	0

H_{Train}, $H_{\text{Controller}}$ and H_{Gate} stay the same as in the previous system configuration; however, $H_{\text{Train-Controller-Gate}}$ is equal to the following:

- ((0.083 * (0 + 1log1 + 0) +
(0.083 * (0 + 1log1 + 0) +
(0.083 * (0 + 1log1 + 0) +
(0.167 * (0 + 1log1 + 0) +
(0.167 * (0 + 1log1 + 0) +
(0.167 * (0 + ½ log½ + ½log½ + 0) +
(0.083 * (0 + 1log1 + 0) +
(0.083 * (0 +1 log1) +
(0.083 * (1log1 + 0)
= 0 + 0.167 + 0
= 0.167

Therefore, the reliability for the second Train-Controller-Gate configuration is:

$= (H_{\text{Train}} + H_{\text{Controller}} + H_{\text{Gate}}) - H_{\text{Train-Controller-Gate}}$
= (0 + 0.615+ 0) – 0.167
= 0.448

As a result, the reliability for the first configuration is greater than the reliability for the second configuration for the case study given above. A higher reliability value implies that there is less uncertainty present in the model, and thus a higher level of software

reliability. Controlling one train means that there is a lower level of uncertainty in a Markov system for the Controller object, and therefore its behavior is not complicated, i.e. it does not require the generation of more sequences to describe it, which is not the case for one controller controlling two trains.

To conclude this chapter, we can state that, as COSMIC-FFP is applicable in the early development phase where we only know about the specifications of the software, COSMIC-FFP measurement details have been used in the scenario-based black-box testing strategy and for reliability prediction purposes. In the next chapter, we discuss the methodology for formalizing COSMIC-FFP in a specific context, the AS-TRM model, which allows early complexity assessment from the formal specifications.

CHAPTER VII: FORMALIZING COSMIC-FFP IN THE AS-TRM

The COSMIC-FFP method can complement complexity management in the AS-TRM, which would allow early complexity assessment from the formal specifications. The formalization of COSMIC-FFP in the AS-TRM context requires mapping the COSMIC-FFP concepts (such as boundary, layer, functional process, triggering event, data group, movement and attributes) to the AS-TRM notation. Clear rules of COSMIC-FFP measurement shall be defined for AS-TRM specifications according to the COSMIC-FFP informal textual definitions given in (Abran, Desharnais et al. 2001). The reliability requirements for autonomic elements and systems have to be specified formally and mapped to system behavior so that the achievement of the reliability can be monitored automatically.

7.1 Mapping between COSMIC-FFP and the AS-TRM

Table 28 displays the starting point of the mapping analysis between COSMIC-FFP and the AS-TRM formalism.

Table 28: Mapping COSMIC-FFP concepts to AS-TRM notations

COSMIC-FFP concepts	*AS-TRM formalism notations*
Boundary	Reactive Component interface
Layer	Tier in the formal model
Functional process	Reactive task or self-management task
Triggering event	Shared input event
Data Group	LSL trait
Data Movement	Internal & External event (input & output)
Data Attribute	Operation in the LSL trait

The boundary concept in COSMIC-FFP corresponds physically to the Reactive Component interface notation in the AS-TRM formalism. The Reactive Component interface is considered rather than the ports that model the interfaces of the generic reactive classes (GRCs) at the design phase, since we are only considering the specification level here. In the specification activity, the user (either a human or an engineered device) wants to communicate with the software in order to receive/enter information. For example, a user communicates with an elevator through buttons in order to go up or down, and, at the specification level, the design details of the elvevator are not of interest. As is also shown in Table 28, the layer concept in COSMIC-FFP corresponds to the tier in the formal model in the AS-TRM formalism, where each upper tier communicates only with the tier immediately below it. The tier structure describes the system configuration, the autonomic group of synchronously interacting reactive components (ACG), the autonomic reactive component (AC), the generic reactive classes (GRCs) and their relative Abstract Data Types used to model attributes in the AS-TRM. The functional process and the subfunctional process concepts in COSMIC-FFP are mapped to the Reactive task or self-management task notation in the AS-TRM formalism. A reactive task is performed during the synchronous communication among the ACs belonging to one Autonomic Component Group, and these cooperate in completing a group task. Each ACG can accomplish a complete real-time reactive task independently. A transition specification describes the computational step associated with the occurrence of an event. A transaction is triggered by an event, and it causes a reaction in the form of an occurrence of either an internal event or an output event. There may be a timing constraint on the occurrence of the reaction. The triggering event and data

movement (i.e. Entry) concepts in COSMIC-FFP correspond to the external (input) event that occurs at a port and represent message transmission in the AS-TRM formalism. Also, a data movement corresponds to an internal (Read from internal storage, Write) or output (Exit) event in the AS-TRM formalism. Finally, a data group corresponds to an LSL trait, the lowest tier and the basic specification unit that represents the Abstract Data Type used in modeling the attributes of the system and of the generic reactive classes. The operation in this LSL trait corresponds to a Read or a Write, the data movement in COSMIC-FFP; an LSL trait included in a given class corresponds to a data attribute in COSMIC-FFP. The above mapping between the COSMIC-FFP and AS-TRM concepts conforms to the AS-TRM event-driven modeling paradigm, where the components communicate through events, and these events carry information.

Once the work detailed in Table 28 has been completed, COSMIC-FFP can be formalized in the specified context, which is the AS-TRM. It is then possible to apply the rules of the COSMIC-FFP FSM method to AS-TRM specifications, as will be seen in the case study presented in the following section. It will also be relevant to obtain feedback about the functional complexity and the reliability prediction of such software specifications. Moreover, by applying the testing approach proposed in previous chapter, it is also possible to generate and execute the test cases in order to estimate the reliability of the AS-TRM specification activity.

7.2 Case Study: Steam Boiler

The Steam Boiler Control specification problem of J. R. Abrial and E. Brger (Abrial 1991) was derived from an original text written by Lt. Col. J. C. Bauer for the Institute

for Risk Research at the University of Waterloo, Ontario, Canada. The original text has been submitted as a competition problem to be solved by the participants of the International Software Safety Symposium organized by the Institute. It provides the specification design that ensures the safe operation of a steam boiler by maintaining the ratio of the water level in the boiler and the amount of steam emanating from it with the help of the corresponding measuring devices. The Steam Boiler System consists of the following physical units:

- Steam Boiler: the container holding the water.
- Pump: for pouring water into the steam boiler.
- Valve: for evacuating water from the steam boiler.
- Water Level Measuring Device: a sensor to measure the quantity of water q (in liters) and inform the system whenever the minimum or maximum amounts allowed are reached.

Figure 41 shows the Steam Boiler and the relationships between its components. The Steam Boiler is assumed to start up with a safe amount of water, and the Controller runs a control cycle every 5 minutes to check on the current amount of water. It triggers the Water Level Measuring Device to measure the level and sends the result to the Controller. Then, the Controller receives the current level and checks whether or not it is normal, or above or below normal: if the water level is normal, the Controller will do nothing; if there is a risk that the water level will exceed the minimum safe level, it will trigger the Pump to pour water into the Steam Boiler; and, if there is a risk that the water will exceed the maximum safe limit, it will trigger the Valve to evacuate water.

The way the specification problem is written is ambiguous from a software viewpoint: it talks about a single controller which is the ‘system controller’, but when the text is read, there is in practice two controllers (a hardware part and a software part). The specifications about the interactions between the hardware and the software are sometimes ill-defined in real time applications: what is really done by hardware, and what is really done by software? For instance, in this case study, the requirements above are at the 'system level', not at the 'software level'. For this case study, an assumption is made that it is the hardware part of the controller that reads the water level and it makes the decision (calculation) about the min-max (and the risk of getting close to the min-max), and then sends the outcome (min-max-normal) to the software part of the controller. The software is then only responsible of sending close-open messages to the valve and the pump.

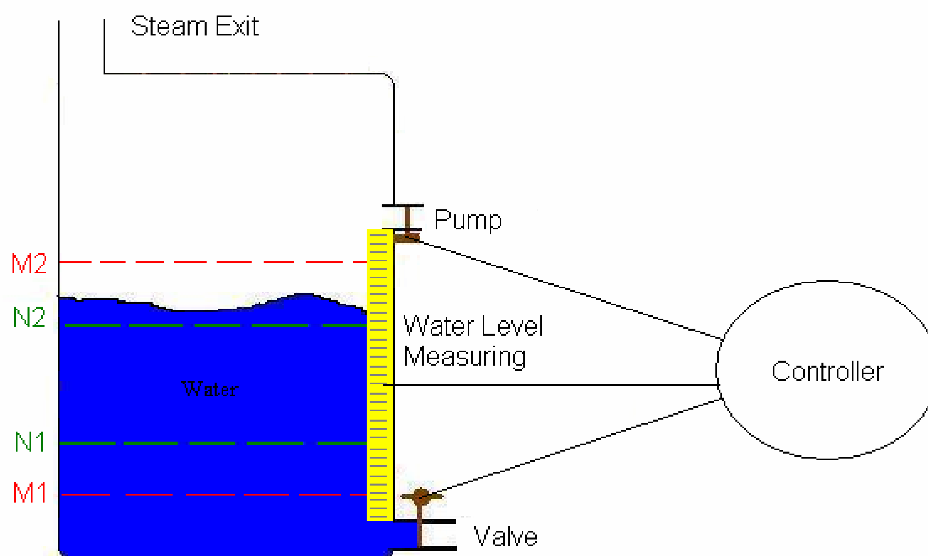

Figure 41: Steam Boiler Controller

It is of interest here to measure the Steam Boiler Controller software (Figure 42), which is located in the AC tier. The Steam Boiler Controller is bounded by its interface, which separates it from the other components.

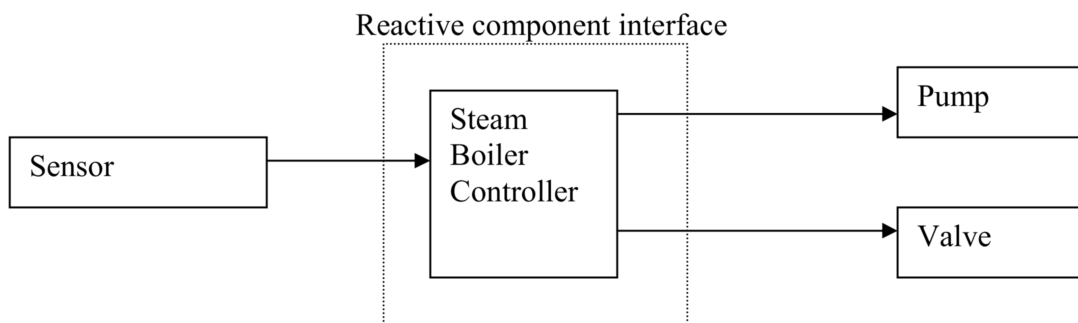

Figure 42: Steam Boiler Controller and its interface

The Steam Boiler Controller has two reactive tasks (or self-management tasks) to perform. These are shown in the following two figures as a sequence of events to be considered in the FSM process. The total number of these events in one sequence diagram corresponds to the total functional size for one reactive task. In other words, one event corresponds to one data movement, the basic elementary unit used in the COSMIC-FFP measurement method.

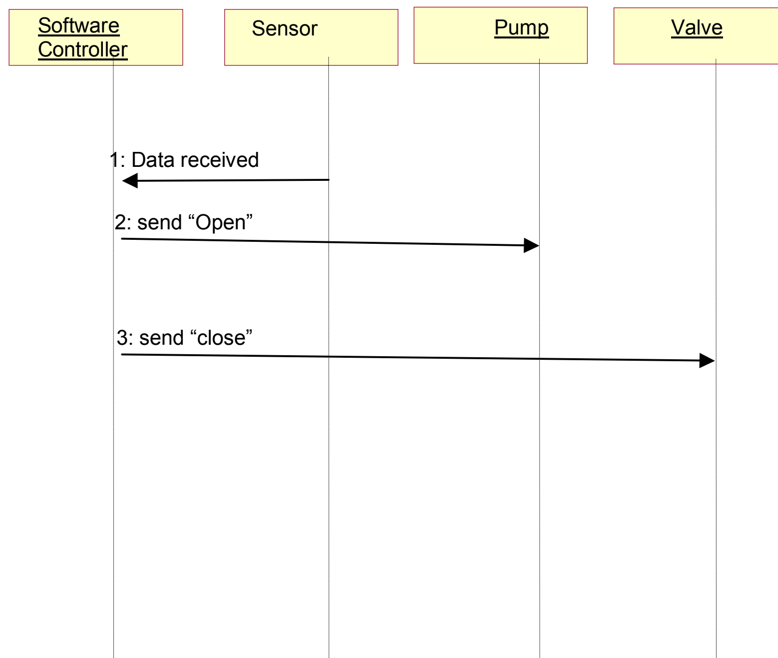

Figure 43: Controller reactive task (1)

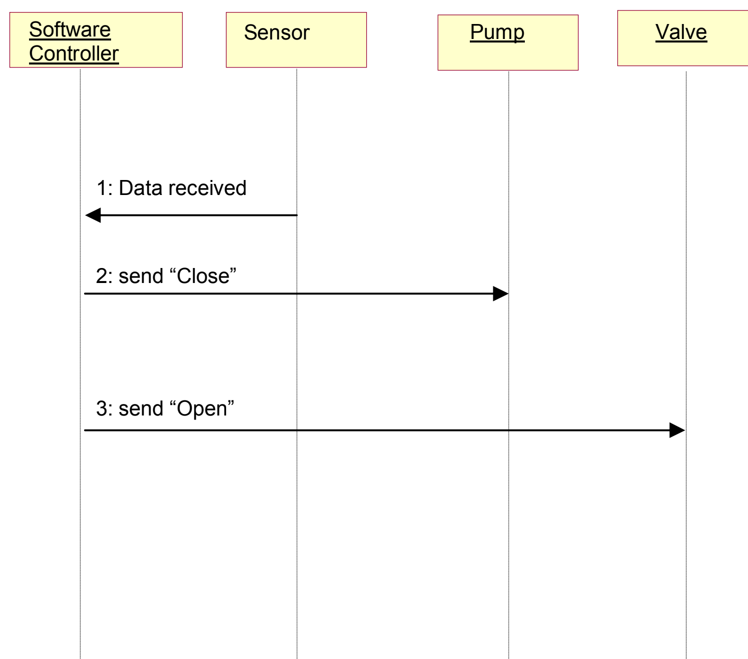

Figure 44: Controller reactive task (2)

Figure 43 shows the first reactive task to be performed through the interaction of the Controller with the other components; that is, when the level of water is below the minimum level. The Controller sends the message "open" to the pump and the Pump reacts accordingly. It also sends the message "close" to the Valve. That reactive task is triggered by a shared input event which is the "cycle" that is received on request of the Controller to check the measuring level every 5 minutes. The second reactive task shown in Figure 44 is also triggered with the "cycle" event. It checks whether or not the measuring level is approaching the maximum safe level, and sends a "close" message to the Pump component and an "open" message to the Valve component. Pump and Valve react accordingly. The total functional size for both tasks is shown in the following table:

Table 29: Total Functional Size for the Steam Boiler using AS-TRM terms

Tier	Reactive task	Sequence of events	Type of event	Corresponding functional size
AC	Maintain Water Level (see Figures 7 & 8)	1. Obtain the water level measurement (value = below normal, normal or above normal)	Shared input event	1
		2. (Logic) Check if any action is needed; if not, terminate the cycle		
		3. Send message to Pump (value = open or close)	External output event	1
		4. Send message to Valve (value = open or close)	External output event	1
Total Functional size of Steam Boiler Controller software				3 Cfsu

Once the rules of COSMIC-FFP measurement have been clearly defined for the AS-TRM specifications in this chapter, we conclude our work using the Basili framework, and present future work in the next chapter.

CHAPTER VIII: CONCLUSIONS AND FUTURE WORK

8.1 Summary of Significant Results

COSMIC-FFP and the Entropy-based Functional Complexity Measure are both measures of software functionality, but their purposes are different, one measuring size and the other the complexity of one usage (functionality) of the software. This thesis has explored the similarities and differences between the two measures. Even though the two measures use very different terminology, a comparative analysis of the concepts behind these distinct terminologies reveals important similarities in terms of how they view and represent software from a functional perspective. Our findings include, in particular, significant similarities in the way both measures view software, and in their generic model of software functionality when the interactions of the software with its environment, and the interactions within the software itself, are considered. There are also important similarities, but not full equivalence, within the software components that they take into account in their respective measurement processes. Finally, each measure obviously has different measurement functions (that is, the formulas they use to transform information into numbers are different): while COSMIC-FFP strictly involves an additive aggregation of data movements, our functional complexity measure is much more complex and is based on the concept of entropy, itself derived from information theory.

COSMIC-FFP measurement concepts and procedures are well documented, and, through its international acceptance as an ISO standard, the method has achieved international recognition as a measurement method supported by the international community

specializing in measurement of any kind. However, the field of software functional size itself has very limited depth in terms of research and theoretical support on which to draw. Its use has therefore been fairly limited, extending only to productivity studies and estimation, with almost no reported use in quality and reliability analysis.

By contrast, entropy has been used extensively in many fields, such as in entropy-based measures, which have been used, for instance, for performance and reliability estimation, both with very strong theoretical and empirical support. The reliability of software can, as a result, be estimated using an input reliability model in a particular input domain.

This is why studying the similarities between these two measures has also enabled us to explore an interesting link between COSMIC-FFP and testing, and between COSMIC-FFP and reliability prediction. As well, we have seen how the concept of scale is used in the COSMIC-FFP method to ensure the meaningfulness of the numbers obtained from its measurement process. We have also defined the measurement unit for the FC measure.

This thesis has described the development of a model for scenario-based black-box testing, in which the scenarios are derived using the COSMIC-FFP method for identifying functionality. Based on that model, the mechanisms of both test-case generation and the prioritization of those test cases, have been elaborated to ensure good fault coverage in black-box testing. First, the set of scenarios in the COSMIC-FFP context represents the set of test cases required to form the generated test set. Second, this set of test cases was partitioned into equivalence classes. Third, the test-selection

algorithm was run on the set of non-empty equivalence classes. The result is a set of selected test cases affording the best possible coverage (i.e. the test case with the highest FC value in each equivalence class). However, we cannot claim full fault coverage, since this algorithm maximizes test coverage within given budgetary constraints. This could be a limitation of the proposed testing method.

Moreover, the candidate linkages between the Markov models and the COSMIC-FFP functional size measurement method were investigated in this thesis for the prediction of software reliability based on Markov concepts in a COSMIC-FFP context. This was achieved as follows: First, the state-machine diagrams were synthesized from the corresponding COSMIC-FFP sequence diagrams. Second, a Markov system was formalized by using the derived COSMIC-FFP state-machine diagrams. Third, the steady state distribution vector for the corresponding Markov system was calculated. The reliability prediction for a component composed of n objects can be defined as the level of certainty quantified by a level of uncertainty in a Markov system corresponding to an object, and a level of uncertainty of a Markov system corresponding to a component. The reliability of the component-based system was calculated from the reliability of the components and the topology of the system.

Finally, we drew the basic lines for formalizing COSMIC-FFP in the AS-TRM context by mapping the COSMIC-FFP concepts (such as boundary, layer, functional process, triggering event, and data group, movement and attributes) to the AS-TRM notation. Clear rules of COSMIC-FFP measurement shall be defined for AS-TRM specifications according to the informal COSMIC-FFP definitions. Current AS-TRM research includes

modeling the non-functional requirements (NFRs) and functional requirements (FRs) within the same formal framework, which would, in future, allow for complexity and testing assessment of both FRs and NFRs with COSMIC-FFP.

The summary of significant results is shown in the following table using the Basili et al. framework described in the first chapter.

Table 30: Summary of Significant Results

I Definition					
Motivation	**Object**	**Purpose**	**Perspective**	**Domain**	**Scope**
Extend the use of COSMIC-FFP, the software FSM method, for testing and reliability prediction purposes	- Information theory-based measures - Functional size measures	- Characterize the link between these two types of measures in order to extend the use of functional measures for scenario-based black-box testing and for reliability prediction purposes - Formalize COSMIC-FFP in the AS-TRM context	Researcher	- Software researcher - 3 case studies	3 case studies – 1 software researcher

II Planning		
Design	**Criteria**	**Measurement**
Three case studies selected: 1. Hotel Accommodation System for the testing approach 2. Railroad System for the reliability prediction approach 3. Boiler Controller for formalizing COSMIC-FFP	Direct criteria: - length of the test case - length of the longest common prefix of two test cases - selected test cases - test set - number of occurrences of an event - total number of events in a sequence - transition matrix	For direct criteria: *length(t)* $LCP\ (t_1, t_2)$ *STC* TS_i *f* *NE* *P*

<table>
<tr><td></td><td><u>Indirect criteria:</u>
- distance between two test cases
- similarity between two test cases
- dissimilarity between two test cases
- functional complexity
- probability of occurrence of the most frequent event
- steady vector
- entropy of the whole component and an object
- reliability of a component and a system</td><td><u>For indirect criteria:</u>
td (t1, t2)
similarity (t1, t2)
dissimilarity (t1, t2)
FC
pi
Vi
H, Hi
Ri, R(t)</td></tr>
<tr><td colspan="3" align="center">III Operation</td></tr>
<tr><td>Preparation
Training period of a few months for the measures</td><td>Execution
<u>Data Collection:</u>
1. Set of scenarios containing the sequence of events
2. Set of state diagrams for each object
<u>Data Validation:</u>
- The formulas and steps stated in the design step must be applied correctly.
- Revisions are performed by experts and the technical reviewers of conferences and journals.
- Theoretical validation is carried out to test the meaningfulness of the numbers obtained using scales, units and scale-type concepts.
- The case studies by themselves demonstrate feasibility.
- The tools available in the TROM lab are used to validate the numbers obtained in the reliability prediction approach.
Conclusion: No errors were found.</td><td>Data Analysis
- Scenarios were created subjectively, according to the case study specifications.
- Scenarios were converted to state diagrams using the steps described in chapter 6.
- The formulas needed in both approaches using the scenarios created and the converted state diagrams were applied, and numbers were obtained to be interpreted in next phase in this framework; for example, the numbers obtained for FC, H, R(t), etc.
- The correlation between these numbers and the overall behavior of either a component or a system in terms of functional complexity and reliability was studied; for example, what happens to the system when numbers increase/decrease, and under what conditions do this happen?</td></tr>
</table>

IV Interpretation	
Interpretation context	**Extrapolation**
Technique Framework: - The greater the number and variety of events contained in a test case, the higher the functional complexity will be. - The higher the value of the reliability measure, the less uncertainty there will be in the model, and thus a higher level of software reliability. Study Purpose: The purpose was achieved, in spite of the difficulties encountered. Field of research: - The metric-based test case partitioning approach and test case technique together generate the most highly recommended test cases which cover the most functionality, given budgetary constraints. This is better than the outcome with other models which use test cases randomly. - Both the testing and reliability prediction approaches developed in this thesis need to be carefully compared with other models proposed in the literature.	Positive factors: - The proposed testing approach cannot test all the possibilities, but it can at least test the maximum number of test cases which cover the most functionality, given budgetary constraints. - There are clearly stated steps and well-defined rules in both approaches. - The existing tools in the TROM lab can be used to validate the results obtained in the reliability approach. Negative factors - The number of case studies was small. What about the scalability problem, and how do these approaches behave with a large amount of data? - More specific templates are needed to generate the scenarios and to convert this set of scenarios into state diagrams.

8.2 Future Work

Future work could include implementing a testing tool following the proposed testing approach, applied to a large number of study cases with more scenarios and carrying out an analysis to compare the results on all these cases. The Rational Unified Process (RUP) case studies would be a good choice to begin with, since they provide a clear description of some software specifications, along with their corresponding detailed use cases and scenarios. Moreover, there are more testing issues that can be enhanced in the proposed testing approach such as:

- improving the overall system testability, which is based on simplifying the complexity of use cases, for example, use cases can be related to each other through “include” and “extend” relationships,
- reducing the number of equivalent classes if they are large by finding a specific selection criteria that defines how many equivalence classes should be considered,
- defining a priority strategy when two test cases have the same size.

Research in progress is also looking into exploring the use of predictions to compare alternative component-based system designs, and to gather data from empirical studies to access the effectiveness of the reliability model and the degree of confidence of the predicted values. Moreover, a comparison (if available) can be performed with the results obtained using alternative methodologies to support the validity of the application of the proposed methodology. In addition, there are more reliability issues that can be enhanced in the proposed reliability prediction approach such as improving the assumptions for assigning the probabilities for the transitions. For example, the probabilities for the

external and the internal events where we can't control the input, but we control the output and the output depends on the inputs.

The application of the metric-based scenario-driven black-box testing method has been adapted to a specific class of projects, namely Enterprise Resource Planning (ERP). ERP projects are perceived as mission-critical initiatives in many organizations. They form parts of business transformation programs and are instrumental in improving organizational performance. In ERP implementations, testing enterprise-wide solutions based on ERP systems is an activity that is critical to ensure that the functionality embedded in the solution matches the business users' requirements. However, little is known about the way to make the testing process more predictable and to increase its chances of success. In (Daneva, Abran et al. 2006), a first attempt was made towards improving the testing process in ERP projects by using the metric-based approach we have proposed here. This paper reports on how this approach was adapted to an ERP package-specific project context, how it was applied to five settings in a mid-sized project, and what was learned by using it.

The original requirements-based ERP approach complements test-case derivation by finding partitions in the input and output data sets and by suggesting that testers perform ERP transactions with values derived from these partitions. The reported study revealed that this method has the potential to complement traditional ERP testing approaches, such as the ones built into and assumed in the ERP packages. The study also revealed two issues:

- The business process documentation should be up to date and valid, so that testers are sure they are testing the most recently required functionality;
- The transactions should be mapped to the scenario processes from the business requirements.

Also recommended, however, is a replicated follow-up case study to be carried out to conduct a deeper investigation into the validity threats associated with the testing method. More case studies will also help promote the use of the proposed method.

Further research efforts are planned to focus on adapting and applying the COSMIC FFP technique (for testing and reliability prediction purposes) to a variety of ERP project contexts characterizing new ERP implementations, upgrades and cross-organizational alignments.

REFERENCES

Abran, A., J.-M. Desharnais, S. Oligny, D. St-Pierre and C. Symons (2001). COSMIC-FFP - Manuel de mesures - version 2.2. Montreal, Université du Québec à Montréal.

Abran, A., O. Ormandjieva and M. Abu Talib (2004). Information Theory-based Functional Complexity Measures and Functional Size with COSMIC-FFP. Proceedings of the 14th International Workshop on Software Measurement (IWSM2004), Germany.

Abran, A. and P. N. Robillard (1994). "Function Points: A study of Their Measurement Processes and Scale Transformations." Journal of Systems and Software **25(2)**: 171-184.

Abrial, J. R. (1991). Steam Boiler Control Specification Problem. In J. R. Abrial, E. Borger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and programming the Steam Boiler Control*, volume 1165 of *LNCS*. Springer, October 1996.

Abu Talib, M., A. Abran and O. Ormandjieva (2006). Markov Model and Functional Size with COSMIC-FFP. IEEE International Symposium on Industrial Electronics: Special session on Software Measurement (ISIE2006), Montreal, Canada.

Abu Talib, M., O. Ormandjieva and A. Abran (2007). "Reliability Model for Component-Based Systems in COSMIC-FFP: A Case Study." Submitted to International Journal of Software Engineering and Knowledge Engineering: Testing and Quality Assurance for Component-Based Systems.

Abu Talib, M., O. Ormandjieva, A. Abran, A. Khelifi and L. Buglione (2006). "Scenario-based Black-Box Testing in COSMIC-FFP: A Case Study." Software Quality (ASQ) Journal.

Achthan, R. (1995). A formal Model for Object-Oriented Development of Real-Time Reactive Systems. Montreal, Canada, Concordia University. **Ph.D. thesis**.

Alagar, V. S., R. Achuthan and D. Muthiayen (1996). TROMLAB: A Software Development Environment for Real-Time Reactive Systems. Montreal, Canada, Concordia University.

Alagar, V. S., M. Chen, O. Ormandjieva and M. Zheng (2006). "Automated Generation of Test Suits from Formal Specifications of Real-Time Reactive Systems." IEEE Transactions on Software Engineering Journal.

Alagar, V. S. and O. Ormandjieva (2002). "Reliability assessment of Web applications." Proceedings 26th Annual International Computer Software and Applications Conference (COMPSAC 2002), pp: 405 - 412.

Alagar, V. S., O. Ormandjieva and M. Zheng (2000). Managing Complexity in Real-Time Reactive Systems. Sixth IEEE International Conference on Engineering of Complex Computer Systems, Japan.

Alur, R. (1999). Lecture Notes in Computer Science. Computer-Aided Verification: 11th International Conference (CAV'99) Trento, Italy, Springer-Verlag GmbH.

Bai, X., L. C. Peng and H. Li (2002). An Approach to Generate Thin Threads from UML Diagrams. S. E. R. Group, School of Computer and Information Science, Edit Cowan University.

Bai, X., W. T. Tsai, K. Feng and L. Yu (2002). Scenario-based Modeling and Its Applications to Object-Oriented Analysis Design and Testing. IEEE Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002), San Diego, USA.

Basili, V. R., R. W. Selby and D. H. Hutchens (1986). Experimentation in Software Engineering. IEEE Transactions on Software Engineering.

Beizer, B. (1990). Software Testing Techniques, second edition, Van Nostrand Reinhold, ISBN: 1850328803.

Bertolino, A. (2004). Knowledge Area Description of Software Testing Guide to the SWEBOK, http://www.swebok.org.

Boehm, B. (2002). Software Cost Estimation with COCOMO II. COCOMO II manuals.

Cammarano, B. (2001). Leveraging points of integration in rational suite: An introduction, http://www-128.ibm.com/developerworks/rational/library/content/RationalEdge/jul01/LeveragingPointsofIntegrationinRationalSuiteJuly01.pdf.

Chen, M. (2002). The Implementation of Specification-based Testing System for Real-time Reactive System in TROMLIB Framework. Department of Computer Science. Montreal, Canada, Concordia University. **Master Major Report**.

Chow, T. S. (1978). Testing Software Design Modeled by Finite State Machines. IEEE Transactions on Software Engineering.

UK Software Metrics Association – Metrics Practices Committee. (1998). "MkII FPA Counting Practices Manual - version 1.3.1."

Daneva, M., A. Abran, O. Ormandjieva and M. Abu Talib (2006). A case study of metric-based and scenario-driven black-box testing for SAP projects. International Workshop of Software Measurement.

Davis, J. S. and R. J. Leblanc (1988). A Study of the Applicability of Complexity Measures. IEEE Transactions on Software Engineering.

DeMarco, T. (1982). Controlling Software Projects. New York, Yourdon.

EnNouaary, A., R. Dssouli and F. Khendek (2002). Timed Wp-Method: Testing Real-Time Systems. IEEE Transactions on Software Engineering.

Institute of Electrical and Electronics Engineers (1991). ANSI/IEEE Standard Glossary of Software Terminology, IEEE Std**:** 729-1992.

Fenton, N. E. and S. L. Pfleeger (1998). Software Metrics: A Rigorous and Practical Approach, PWS Publishing Company.

Goˇseva-Popstojanova, K. and S. Kamavaram (2004). Software Reliability Estimation under Uncertainty: Generalization of the Method of Moments. the Eighth IEEE International Symposium on High Assurance Systems Engineering (HASE 2004).

Goˇseva–Popstojanova and S. Kamavaram (2003). Assessing Uncertainty in Reliability of Component-Based Software Systems. 14th International Symptoms Software Reliability Engineering,.

Haidar, G. (1999). Reasoning System for Real-Time Reactive Systems. Department of Computer Science. Montreal, Canada, Concordia University. **Master's thesis**.

Hamming, R. (1980). Coding and Information Theory. Englewood Cliffs, NJ: Prentice-Hall.

Harrison, W. (1992). An Entropy-Based Measure of Software Complexity. IEEE Transactions on Software Engineering.

Henderson-Sellers, B. (1996). Object-Oriented Metrics: Measures of Complexity. New Jersey, Prentice-Hall.

IBM (2004). IBM rational system testing family, http://www.squadra.com.br/novosite/sitetest/ibm/RationalFunctional%20testing.pdf.

IFPUG (2005). The International Function Point Users' Group (IFPUG), http://www.ifpug.org/default.htm.

ISO14143-1 (1988). Functional size measurement - Definitions of concepts. Geneva, International Organization for Standardization - ISO.

ISO-graph, http://www.isograph-software.com/index.htm.

ISO (1993). International Vocabulary of Basic and General Terms in Metrology (VIM). Geneva, International Organization for Standardization.

ISO (2002). ISO 15939:2002 Software Engineering - Software Measurement Process. Geneva, International Organization for Standardization.

ISO/IEC19761 (2003). Software Engineering - COSMIC-FFP - A functional size measurement method. Geneva, International Organization for Standardization - ISO.

Item-software "Item software", http://www.itemsoft.com.

Jenner, M. (2002). Automation of Counting of Functional Size Using COSMIC-FFP in UML. 12th International Workshop on Software Measurement (IWSM 2002), Magdeburg, Germany.

Khoshgoftaar, T. M. and E. B. Allen "Applications of information theory to software engineering measurement." Software Quality Journal **3 (2)**: 79 -103.

Koskimies, K., T. Mannisto, T. Systa and J. Tuomi (1998). "Automatic support for dynamic modeling of object-oriented software." IEEE Software **15(1)**: 87-94.

Kruchten and Philippe (2000). The Rational Unified Process: An Introduction, Addison-Wesley Pub. Co.

Lee, F.-A. (2003). Reliability Measurement Based on the Markov Model for Real-time Reactive Systems: Design and Implementation. Department of Computer Science. Montreal, Canada, Concordia University. **Major report**.

Martin, N. F. G. and J. W. England (1981). Mathematical Theory of Entropy, Addison-Wesley Pub. Co.

Musa, J. D. and K. Okumoto (1982). Software reliability models: concepts, classification, comparisons, and practice. The Electronic Systems Effectiveness and Life Cycle Costing Conference, Norwich, U.K., Springer-Verlag, Heidelberg.

Muthiayen, D. (1996). Animation and Formal Verification of Real-Time Reactive Systems in an Object-Oriented Environment. Department of Computer Science. Montreal, Canada, Concordia University. **Master's thesis**.

Muthiayen, D. (2000). Real-Time Reactive System Development: A Formal approach based on UML and PVS. Department of Computer Science. Montreal, Canada, Concordia University. **Ph.D. Thesis**.

NESMA. (2004). "Differences between NESMA & IFPUG." From http://www.nesma.nl.

Oligny, S. and A. Abran (1999). On The Compatibility Between Full Function Points And IFPUG Function Points. The 10th European Software Control and Metrics Conference (ESCOM SCOPE 99), Herstmonceux Castle, England.

Ormandjieva, O. (2002). Deriving New Measurement for Real Time Reactive Systems. Computer Science & Software Engineering Department. Montreal, Concordia University.

Peters, J. F. and W. Pedrycz (2000). Software Measures in Software Engineering: An Engineering Approach, J. Wiley.

Pham, H. (1999). Software Reliability. New York, Springer-Verlag, ISBN: 9813083840.

Pompeo, F. (1999). A Formal Verification Assistant for TROMLIB environment. Department of Computer Science. Montreal, Canada, Concordia University. **Master's thesis**.

Popistas, O. (1999). Rose-GRC Translator: Mapping UML Visual Models onto Formal Specifications. Department of Computer Science. Montreal, Canada, Concordia University. **Master's thesis**.

The Object Primer (2004). Agile Model-Driven Development with UML 2, Cambridge University Press, third edition, ISBN: 0-521-54018-6.

Relex Software, http://www.relexsoftware.com.

Ryser, J. and M. Glinz (2000). Using dependency charts to improve scenario-based testing. The 17th International Conference on Testing Computer Software (TCS2000), Washington, D.C.

Sellami, A. and A. Abran (2003). The contribution of metrology concepts to understanding and clarifying a proposed framework for software measurement validation. International Workshop on Software Measurement (IWSM), Montreal, Shaker-Verlag.

Shannon, C. E., Weaver and Warren (1969). The Mathematical Theory of Communication. Chicago, University of Illinois Press.

Shukla, S. (2005). Lecture 04: Team Organization, Tools & Case Tools. School of Information Technology and Engineering, University of Ottawa.

Srinivasan, V. (1999). Graphical User Interface for TROMLIB Environment. Department of Computer Science. Montreal, Canada, Concordia University. **Master's thesis**.

Strook, D. W. (2005). An Introduction to Markov Processes. Berlin, Heielberg, Springer-Verlag.

Symons, C. (1999). Conversion between IFPUG 4.0 and MK II Function Points - version 3.0, http://www.gifpa.co.uk/library/Papers/Symons/AlbvMkII_v3b.pdf.

Abu Talib, M., A. Abran and O. Ormandjieva (2005). COSMIC-FFP & Entropy: A Study of Their Scale Transformations. Proceedings of the 15th International Workshop of Software Measurement (IWSM2005).

Abu Talib, M., O. Ormandjieva, A. Abran and L. Buglione (2005). Scenario-based Black-Box Testing in COSMIC-FFP. Proceedings of the 2nd Software Measurement European Forum 2005 (SMEF 2005), Italy.

Tao, H. (1996). Static Analyzer: A Design Tool for TROM. Department of Computer Science. Montreal, Canada, Concordia University. **Master's thesis**.

Trvedi, A. K. (1975). Computer Software Reliability: Many-State Markov Modeling Techniques, Polytechnic Institute of Brooklyn. **Ph.D. dissertation**.

V. S. Alagar, O.Ormadjieva and J. Shen (2004). Scenario-Based Performance Modeling and Validation in Real-Time Reactive Systems. the First Software Measurement European Forum (SMEF 2004), Rome, Italy.

Vangalur, S., V. S. Alagar and K. Periyasamy (1998). Specification of Software Systems, Springer Verlag.

Vasilache, S. and J. Tanaka (2004). Synthesis of state machines from multiple interrelated scenarios using dependency diagrams. Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004), Orlando, Florida, USA.

Vassev, E., H. Kuang, O. Ormandjieva and E. Paquet (2006). Reactive, Distributed and Autonomic Computing Aspects of AS-TRM. 1st International Conference on Software and Data Technologies-ICSOFT2006, Setubal, Portugal.

Weiss, S. N. and E. J. Weyuker (1988). An Extended Domain-Based Model of Software Reliability. IEEE Transactions Software Engineering.

Whitmire, S. A. (1997). Object Oriented Design Measurement, John Wiley & Sons.

Whittle, J. and J. Schumann (2000). Generating Statechart Designs from Scenarios. Proceedings of International Conference on Software Engineering (ICSE2000), Limerick, Ireland.

Wikipedia-Encyclopedia, http://en.wikipedia.org/wiki/Main_Page.

Zheng, M. (2002). Automated Generation of Test Suits from Formal Specifications of Real-Time Reactive Systems. Department of Computer Science. Montreal, Canada, Concordia University. **Ph.D. thesis**.

Zuse, H. (1991). Software Complexity Measures and Methods. Berlin, New York, Walter de Gruyter.

ABBREVIATIONS

This thesis has used many abbreviations. The list for the whole abbreviations is shown as following:

FSM: Functional Size Measurement

COSMIC-FFP: Common Software Measurement International Consortium – Full Function Point.

Cfsu: Cosmic Functional Size Unit.

FUR: Functional User Requirements.

FP: Function Point.

UFC: Unadjusted Fucntion Point.

TDI: Total Degree of Influence.

GSC: General System Characteristics.

EI: External Input.

EO: External Output.

EQ: External Inquiry.

ILF: Internal Logical Files.

EIF: External Interface Files.

LT: Logical Transaction.

DET: Data Element Type.

UML: Unified Modeling Language.

RUP: Rational Unified Process.

OO: Object Oriented.

MTTF: Mean Time to Failure.

MTBF: Mean Time Between Failures.

MIS: Management Information System.

IFPUG: International Function Point Users Group.

NESMA: Netherlands Software Metrics Users Association.

MK2: Mark 2.

LOC: Lines of Codes.

V: Test Selection Domain.

STC: Generated Test Set.

TS: Equivalence Class.

LCP: Longest Common Prefix.

FC: Functional Complexity based entropy.

TROM: Timed Reactive Object Model.

TROM-SRMS: TROM System Reliability Measure.

AS-TRM: Autonomic Systems Timed Reactive Object Model.

AC: Timed Reactive Autonomic Component.

ACG: Group of synchronously interacting AC.

AS: Autonomic System.

AGM: ACG Manager.

GM: Global Manager.

LSL: Larch Shared Language.

NFR: Non Functional Requirements.

FR: Functional Requirements.

ERP: Enterprise Resource Planning.