

**A PERSPECTIVE-BASED COMPLEXITY FRAMEWORK AND METRICS
FOR PREDICTING THE ANALYZABILITY OF UNIFIED MODELING
LANGUAGE BEHAVIORAL DIAGRAMS**

Ann Wambui King'ori

**A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of
Doctor of Philosophy in Information Technology of Murang'a University of
Technology**

June, 2025

DECLARATION

Student Declaration

I hereby declare that this thesis is my original work and to the best of my knowledge has not been presented for a degree in this or any other University. I understand what plagiarism is and I am aware of the University policy in this regard. I understand that any false claim in respect of this work shall result in disciplinary action in accordance with the University Anti-Plagiarism Policy.

Ann Wambui King'ori
SC500/5700/2021

Date

Supervisors' Approval

The undersigned certify that they have read and hereby recommend for acceptance of Murang'a University of Technology a thesis entitled "**A Perspective-Based Complexity Framework and Metrics for Predicting the Analyzability of Unified Modelling Language Behavioral Diagrams**".

Prof. Geoffrey Muchiri Muketha,
Department of Computer Science,
Murang'a University of Technology.

Date

Dr. John Gichuki Ndia,
Department of Information Technology,
Murang'a University of Technology.

Date

DEDICATION

I dedicate this work to my loving parents, Margaret King'ori and Joseph King'ori. To my beloved daughter, Gabriella Wanjiku. To my sisters Virginia King'ori, Beth King'ori, and Catherine King'ori. I also extend to my dear friend, Fortine Mata, and my confidant, John Mulatya.

ACKNOWLEDGEMENT

First, I thank God for providing me the energy and the knowledge for writing this thesis.

Next, I recognize Prof. Geoffrey Muchiri Muketha who is my first supervisor for his invaluable guidance and assistance during the development of my thesis. He has contributed immensely in imparting knowledge on software and software metrics. Also, I acknowledge my second supervisor Dr. John Gichuki Ndia who has contributed in imparting knowledge on software metrics and especially towards shaping my thesis.

I am grateful to the faculty and staff of the School of Computing and Information Technology (SCIT) for their dedication in facilitating the successful completion of my course units, which played a crucial role in identifying my research area. I also extend my appreciation to the administrative staff at Murang'a University of Technology for their support in ensuring I met all the requirements. My sincere gratitude goes to the management of Murang'a University of Technology for making this course available.

I would like to express my heartfelt appreciation to my family for providing a conducive environment throughout my studies. Thank you for sacrificing family time, which enabled me to focus on my academic journey. Your unwavering moral and spiritual support made this achievement possible.

I am also grateful to my classmates for their encouragement, reminders, and support in fulfilling my academic journey. Your positive feedback during presentations and words of motivation kept me on track.

ABSTRACT

Software designers are increasingly adopting Unified Modeling Language (UML) behavioral diagrams to communicate a system's dynamic behavior to other software development team members. UML behavioral diagrams have inherent complexity which gets more complex whenever the diagrams evolve. This leads to complex systems that are not easy to analyze. Although several metric measurement frameworks and measures have been proposed in the literature, they cannot be applied to analyze the complexity of these diagrams due to their limitations. This is because behavioral diagrams have unique features such as elements, control flow, and interaction. The main objective of this study was to develop a perspective-based metrics framework for predicting the analyzability of UML behavioral diagrams. Specifically, the study sought to identify relevant complexity perspectives, design a metrics theoretical validation framework, define perspective-based metrics, develop a metrics automation tool, and empirically evaluate the predictive capability of the metrics. To achieve these objectives, a pragmatism philosophy was chosen. Therefore, a mixed research methodology was adopted to solve the problem. A qualitative approach was used to identify the perspective-based information for the definition and theoretical validation of metrics. Further, a quantitative approach was used to investigate the usefulness of the metrics in predicting the analyzability of the metrics. A metrics tool named Behavioral Complexity Metrics Tool (BCMT) was developed to automate the computation of metric values from the diagrams. The BCMT tool and the structured questionnaire were employed for data collection. The snowballing and systematic sampling techniques were employed to select 11 experts to validate the perspective-based complexity framework and 58 subjects for validation of complexity metrics. The data collected was analyzed through descriptive statistics (frequencies, mean, range, maximum, minimum and standard deviation) and inferential statistics (correlation, and regression analysis). Results reveal that the proposed Perspective-based Complexity Framework (PBCF) can be relied upon as a complexity taxonomy for UML behavioral diagrams. Also, findings indicate that the Perspective-based Metrics Validation Framework (PBMVF) provides a comprehensive and systematic approach of evaluating behavioral metrics. Empirical validations were conducted, and results indicate that the metrics are strongly correlated with analyzability and time to analyze diagrams, and the regression model showed that the metrics are predictors of analyzability. These results will particularly aid the software designers, to guide in decision making on how to design systems that are easy to analyze hence ensuring the quality of these diagrams. It is therefore recommended that the framework and metrics be adopted in software design practices and tools to support the development of more analyzable systems. In conclusion, the proposed metrics and framework are reliable, practical, and contribute significantly to improving the analyzability and overall quality of UML behavioral diagrams.

TABLE OF CONTENTS

DECLARATION.....	ii
DEDICATION.....	iii
ACKNOWLEDGEMENT.....	iv
ABSTRACT.....	v
LIST OF TABLES	xiv
LIST OF FIGURES	xvi
ACRONYMS AND ABBREVIATIONS.....	xvii
DEFINITION OF TERMS.....	xviii
CHAPTER ONE: INTRODUCTION	1
1.1 Background to the Study	1
1.2 Problem Statement	2
1.3 Objectives	4
1.3.1 General Objective	4
1.3.2 Specific Objectives	4
1.4 Research Questions	5
1.5 Significance of the Study	5
1.6 Scope of the Study.....	6
1.7 Limitations of the Study	7
1.8 Contributions of the Thesis	7
1.9 Organization of the Thesis	8
CHAPTER TWO: LITERATURE REVIEW.....	10
2.1 Introduction	10
2.2 UML Behavioral Diagrams	10
2.2.1 Statechart Diagrams.....	10
2.2.2 Sequence Diagrams	11
2.2.3 Activity Diagrams	12
2.2.4 Use Case Diagrams.....	14
2.3 Software Measurement.....	15
2.3.1 Measurement Theory	15
2.3.2 Complementary Cognitive Perspectives.....	16
2.3.2.1 Cognitive Load Theory	16
2.3.2.2 Dual Coding Theory.....	17
2.3.2.3 Mental Model Theory	17
2.3.3 Metrics Definition Frameworks	17
2.4 Analyzability	21
2.4.1 Basic Concepts of Analyzability	21

2.4.2 Analyzability and UML Behavioral Diagrams.....	22
2.4.3 Software Quality Models and Analyzability	22
2.4.3.1 McCall's model.....	23
2.4.3.2 Boehm's Model.....	24
2.4.3.3 FURPS Model.....	24
2.4.3.4 Dromey's model.....	25
2.4.3.5 ISO 25010	26
2.5 Software Metrics	27
2.5.1 Line of code (LOC)	28
2.5.2 Metrics by Halstead.....	28
2.5.3 McCabe's Cyclomatic Complexity	29
2.5.4 Function points (FP)	29
2.5.5 Existing Metrics for UML Behavioral Diagrams.....	30
2.5.6 Existing Cognitive Complexity Metrics for UML Behavioral Diagrams ..	31
2.5.6.1 Cognitive Complexity Metrics for Statechart Diagrams.....	31
2.5.6.2 Cognitive Functional Size (CFS)	32
2.5.6.3 Class Complexity	32
2.5.6.4 Cognitive Information Complexity Measure (CICM)	32
2.5.6.5 Modified Cognitive Complexity Measure (MCCM)	33
2.5.6.6 Cognitive Program Complexity Measure (CPCM).....	33
2.5.6.7 Slice-based Cognitive Complexity Metrics	34
2.5.6.8 Cognitive Complexity for Business Process Model (CCBP)	35
2.5.6.9 Weighted Class Complexity (WCC).....	35
2.5.6.10 Attribute weighted class Complexity (AWCC)	36
2.5.6.11 Improved Cognitive Complexity Metric (ICCM).....	36
2.6 Metrics Tooling	37
2.6.1 SLOC Count	37
2.6.2 NDepend.....	37
2.7 Metrics Validation Approaches.....	38
2.7.1 Theoretical Frameworks	38
2.7.1.1 Weyuker's nine properties	38
2.7.1.2 Kitchenham properties	39
2.7.1.3 Briands' Framework	40
2.7.1.4 Kaner's Framework.....	42
2.7.2 Empirical Validation.....	43
2.7.2.1 Experiments	43
2.7.2.2 Case Studies	43
2.7.2.3 Surveys.....	44
2.8 Conceptual Framework	44

2.9 Summary	45
CHAPTER THREE: METHODOLOGY.....	47
3.1 Introduction	47
3.2 Research Philosophy	47
3.3 Research Design	48
3.4 Research Process	48
3.4.1 Development of a Perspective-Based Complexity Framework.....	49
3.4.2 Definition of Metrics Based on the Perspective-Based Complexity Framework.....	50
3.4.2.1 Element Complexity Metrics	50
3.4.2.2 Control-flow Complexity Metric	51
3.4.2.3 Interaction Complexity Metrics	51
3.4.3 Theoretical Validation of the Metrics.....	51
3.4.4 Development of Behavioral Metrics Complexity Tool	52
3.5 Population.....	52
3.6 Sampling Technique and Sample Size	53
3.7 Pilot Study	53
3.8 Data Collection Instruments	54
3.8.1 Validity of the Research Instruments	54
3.8.1.1 Content Validity	54
3.8.1.2 Face Validity	54
3.8.1.3 Internal Validity	54
3.8.1.4 External Validity	55
3.8.2 Reliability of the Research Instruments.....	55
3.9 Experimental Materials	55
3.10 Data Analysis	56
3.11 Ethical Issues.....	56
3.12 Summary	57
CHAPTER FOUR: A PERSPECTIVE-BASED COMPLEXITY FRAMEWORK (PBCF) FOR UML BEHAVIORAL DIAGRAMS	58
4.1 Introduction	58
4.2 Overview	58
4.3 PBCF Framework Requirements.....	58
4.4 Architectural Design.....	59
4.4.1 Element Perspective	61
4.4.2 Control Flow Perspective	62
4.4.3 Interaction Perspective	63
4.5 Expert Opinion Validation Survey	64
4.5.1 Goal of the Study	64

4.5.2 Context Definition	65
4.5.3 Survey Operation	65
4.5.4 Reliability of the Research Instrument	65
4.6 Results	66
4.6.1 Respondent Demographics and Characteristics	66
4.6.2 Academic Qualifications	67
4.6.3 Industry Experience	67
4.6.4 Expertise in UML Modeling	68
4.6.5 Relevance of the Framework	68
4.6.6 Inclusiveness of the Framework	69
4.6.7 Adoption of the Framework	73
4.6.8 Qualitative Results	74
4.7 Discussion	74
4.8 Summary	76
CHAPTER FIVE: A PERSPECTIVE-BASED METRICS VALIDATION	
FRAMEWORK (PBMVF)	77
5.1 Introduction	77
5.2 Requirements for the PBMVF	77
5.3 Theoretical Foundations	78
5.3.1 Definitions	78
5.3.2 Complexity Perspective of Behavioral Diagrams	79
5.3.3 Kaner's Framework and Briand's Metrics Framework	80
5.4 Measurement Attributes and their Properties	81
5.4.1 Element	81
5.4.2 Control Flow	83
5.4.3 Interaction	84
5.5. Results	85
5.5.1 Evaluation with Selected Behavioral Metrics	85
5.5.1.1 Element Metrics	86
5.5.1.2 Control Flow	89
5.5.2 Comparison with Existing Validation Frameworks	91
5.6 Discussion	92
5.7 Summary	94
CHAPTER SIX: A METRICS SUITE FOR UML BEHAVIORAL	
DIAGRAMS BASED ON COMPLEXITY PERSPECTIVES	95
6.1 Introduction	95
6.2 Determination of Measurable Attributes	95
6.3 Metrics Definition	96
6.3.1 Element Complexity Metrics	96

6.3.1.1 Element Complexity Metric for Statechart	97
6.3.1.2 Element Complexity Metric for Sequence diagrams	99
6.3.1.3 Activity Diagrams Metrics.....	102
6.3.2 Aggregate Control Flow Complexity Metric.....	103
6.3.3 Interaction Metrics.....	105
6.4 Results	108
6.4.1 Calculating Real Life Scenarios	108
6.4.1.1 Scenario I: Statechart Diagrams.....	109
6.4.1.2 Scenario II: Activity Diagrams	115
6.4.1.3 Scenario III: Sequence Diagrams.....	122
6.4.2 Comparison with Existing Behavioral Metrics	130
6.4.3 Theoretical Validation Results	132
6.4.3.1 Validation with Weyuker's Properties	132
6.4.3.2 Validation with the PBMVF Framework.....	135
6.4.3.3 Interaction	137
6.5 Discussion	138
6.5.1 Implications of Results from Real-Life Scenario Calculations	138
6.5.2 Implications of Results from Comparison with Existing Metrics	139
6.5.3 Implications of Results from Theoretical Validations.....	139
6.6 Summary	140
CHAPTER SEVEN: DEVELOPMENT OF A BEHAVIORAL	
COMPLEXITY METRICS TOOL (BCMT)	141
7.1 Introduction	141
7.2 Requirements of BCMT Tool.....	141
7.3 Implemented Metrics.....	142
7.4 Structure of Input File	142
7.5 Architectural Design.....	143
7.5.1 Input Component	143
7.5.2 Analyzer Component.....	143
7.5.3 Output Component	144
7.6 User Interface Design.....	146
7.7 Algorithm Design.....	148
7.7.1 Weighted Number of State and Activities (WNSA) Algorithm.....	148
7.7.1.1 Number of Simple States (NSS)	148
7.7.1.2 Number Initial States (NIS)	148
7.7.1.3 Number of Final states (NFS)	149
7.7.1.4 Number of Composite States (NCS).....	149
7.7.1.5 Number of History States (NHS)	149
7.7.1.6 Number of Orthogonal States (NOS).....	149

7.7.1.7 Number of Submachine States (NSMS)	150
7.7.1.8 Number of Activities inside a State (NAS).....	150
7.7.1.9 To measure Weighted Number of States Activities (WNSA)	150
7.7.2 Total Sequence Complexity (TSC) Algorithm.....	151
7.7.2.1 Number of Delay Messages (NDM).....	151
7.7.2.2 Number of Synchronous Messages (NSM)	151
7.7.2.3 Number of Asynchronous Messages (NAM)	151
7.7.2.4 Number of Return Messages (NRM).....	151
7.7.2.5 Number of Lifelines Transmitted Over (NLTO)	152
7.7.2.6 Adjusted Weighted Message into Lifeline (AWMIL)	152
7.7.2.7 Adjusted Weighted Message out of Lifeline (AWMOL)	152
7.7.2.8 Total Sequence Complexity (TSC).....	153
7.7.3 Adjusted Weighted Number of State (AWNS) Algorithm.....	153
7.7.3.1 Number of Action States (NAS).....	153
7.7.3.2 Total Count of Initial States (TCIS).....	153
7.7.3.3 Total Count of Final States (TCFS)	154
7.7.3.4 Weighted Number of States Activities (WNSA)	154
7.7.4 Aggregate Control Flow Complexity (ACFC) Algorithm	154
7.7.4.1 Number of Sequence (NS)	154
7.7.4.2 Number of Decision (ND)	155
7.7.4.3 Number of Loop (NL).....	155
7.7.4.4 Number of Parallel (NP)	155
7.7.4.5 Number of Guards (NG)	155
7.7.4.6 Aggregate Control Flow Complexity (ACFC)	155
7.7.5 Total Interaction Complexity (TIC) Algorithm.....	156
7.7.5.1 Number of Incoming Messages (NIM).....	156
7.7.5.2 Number of Outgoing Messages (NOM)	156
7.7.5.3 Incoming Interaction Complexity (IIC)	156
7.7.5.4 Outgoing Interaction Complexity (OIC).....	156
7.7.5.5 Total Interaction Complexity (TIC).....	157
7.8 Implementation and Validation	157
7.9 Discussion	160
7.10 Summary	161
CHAPTER EIGHT: EXPERIMENTAL VALIDATION OF UML	
BEHAVIORAL METRICS.....	162
8.1 Introduction	162
8.2 Context Definition.....	162
8.3 Experimental Execution Approach	163
8.4 Lessons Learned from the Pilot Study	164

8.5 Subjects' Background.....	165
8.6 Experimental Planning	167
8.6.1 General Description of the Experiment	167
8.6.2 The First Experiment	168
8.6.3 The Second Experiment.....	170
8.6.4 The Third Experiment	172
8.6.5 Validity Evaluation.....	174
8.6.5.1 Threat to Construct Validity	174
8.6.5.2 Threat to Internal Validity	174
8.6.5.3 Threat to External Validity	174
8.6.5.4 Threat to Conclusion Validity.....	175
8.7 Results	176
8.7.1 Results from the First Experiment.....	176
8.7.2 Results from the Second Experiment	179
8.7.3 Results from the Third Experiment	182
8.8 Discussion	185
8.8.1 Implications of Results from the First Experiment	185
8.8.2 Implications of Results from the Second Experiment.....	185
8.8.3 Implications of Results from the Third Experiment.....	186
8.8.4 Global Comparison of Results.....	186
8.9 Summary	187
CHAPTER NINE: CONCLUSION, RECOMMENDATIONS AND	
FUTURE WORK	188
9.1 Conclusion.....	188
9.2 Recommendations	191
9.2.1 Recommendations for Policy.....	192
9.2.1.1 Integration of Perspective-Based Metrics into National Software Development Standards	192
9.2.1.2 Promote Development and Use of Automated Tools through Innovation Policies.....	192
9.2.1.3 Formal Certification and Quality Compliance Standards	193
9.2.2 Recommendations for Practice	193
9.2.2.1 Use of the Framework in Regular Software Design Processes.....	193
9.2.2.2 Application of the BCMT Tool in Development Environments.....	193
9.2.2.3 Training and Professional Development for Practitioners	193
9.2.2.4 Feedback Mechanisms for Continuous Improvement	194
9.2.2.5 Adoption in Agile and Model-Driven Development Environments .	194
9.3 Future Work	194
9.3.1 Extending the Perspective-Based Complexity Framework	194

9.3.2 Extending the Behavioral Complexity Metrics Tool.....	195
9.3.3 Defining Metrics for other Behavioral Diagrams.....	195
9.3.4 Conduct Further Experimentations on the Proposed Metrics.....	195
REFERENCES.....	196
APPENDICES	210

LIST OF TABLES

Table 3.1: Reliability of the Questionnaire	55
Table 4.1: Framework Inclusiveness Reliability	66
Table 4.2: Level of Education.....	67
Table 4.3: Experience in the Software Industry	67
Table 4.4: Knowledge in UML Modeling	68
Table 4.5: Relevance of the Framework	68
Table 4.6: Relevance of the Perspective-Based Framework.....	69
Table 4.7: Inclusiveness of the Element Perspective	70
Table 4.8: Rating of the Inclusiveness of Element Perspective	70
Table 4.9: Inclusiveness of the Control-flow Perspective	71
Table 4.10: Rating of the Inclusiveness of Element Perspective	72
Table 4.11: Inclusiveness of the Interaction Perspective	72
Table 4.12: Rating of the Inclusiveness of Interaction Perspective	73
Table 4.13. Adoption of the Perspective Based Framework	73
Table 4.14: Rating on the Adoption of the Framework	74
Table 5.1: Summary of Validation Results.....	87
Table 5.2: Validation Results.....	88
Table 5.3: Validation of State _{Comp} metric.....	89
Table 5.4: Validation of WNT Metric	90
Table 5.5: Validation of Trans _{Comp} metric.....	91
Table 5.6: Comparison with Existing Validation Frameworks.	92
Table 6.1: Complexity of State Activities.....	99
Table 6.2: Complexity of Messages.....	100
Table 6.3: Complexity of Length of Message	100
Table 6.4: Control Flow Complexity	104
Table 6.5: Complexity of Sequence Guards	105
Table 6.6: Complexity of Incoming Messages	106
Table 6.7: Complexity of Outgoing Messages	107
Table 6.8: Comparison Results	132
Table 6.9: Evaluation using Weyuker's Properties.....	134
Table 6.10: Summary of Validation Results of Element Metrics.	136
Table 6.11: Validation Results of Aggregate Control Flow Metric.....	137
Table 6.12: Summary of Validation Results of Interaction Metrics	138
Table 8.1: Units in Software Engineering.....	165
Table 8.2: Subjects by Number of Programming Languages	165
Table 8.3: Training on UML Diagrams	166
Table 8.4: Knowledge on Behavioral Diagrams	166

Table 8.5: Use of Behavioral Features.....	167
Table 8.6: Metrics for Sequence Diagrams.....	169
Table 8.7: Descriptive Statistics for Sequence Metrics	169
Table 8.8: Metrics for Activity Diagrams.....	171
Table 8.9: Descriptive Statistics for Activity Metrics.....	171
Table 8.10: Metrics for Statechart Diagrams	173
Table 8.11: Descriptive Statistics for Statechart Metrics.....	173
Table 8.12: Correlation between Sequence Metrics	176
Table 8.13: Correlation with Analyzability Ranking.....	177
Table 8.14: Regression Results for Analyzability Rating.....	178
Table 8.15: Relationship with Analyzability Timing	179
Table 8.16: Regression Results for Analyzability Timing.....	179
Table 8.17: Correlation between Activity Metrics.....	180
Table 8.18: Correlation with Analyzability Ranking.....	180
Table 8.19: Regression Results for Analyzability Rating.....	181
Table 8.20: Relationship with Analyzability Timing	181
Table 8.21: Regression Results with Analyzability Timing	182
Table 8.22: Correlation between Statechart Metrics.....	182
Table 8.23: Relationship with Analyzability Ranking.....	183
Table 8.24: Regression Results for Analyzability Rating.....	184
Table 8.25: Relationship with Analyzability timing.....	184
Table 8.26: Regression Results with Analyzability Timing	185

LIST OF FIGURES

Figure 2.1: Statechart Diagram	11
Figure 2.2: Sequence Diagram	12
Figure 2.3: Borrow a Book Activity Diagram	14
Figure 2.4: Library Management System Use Case Diagram	15
Figure 2.5: New Metrics Definition Process	16
Figure 2.6: ISO-25010 Maintainability Sub-characteristics	27
Figure 2.7: Conceptual Framework	45
Figure 3.1: Research Process	49
Figure 4.1: Perspective-based Complexity Framework	60
Figure 4.2: Statechart Elements	61
Figure 4.3: Sequence Diagram Structure	61
Figure 4.4: Activity Diagram Structure	62
Figure 4.5: Control Flows in Behavioral Diagrams	63
Figure 4.6: Reservation Statechart Diagram	64
Figure 6.1: The Element Perspective	96
Figure 6.2: Statechart Diagram	97
Figure 6.3: Elements of a Sequence	99
Figure 6.4: Elements of an Activity Diagram	102
Figure 6.5: The Control Flow Perspective	104
Figure 6.6: The Interaction Perspective	106
Figure 6.7: Computer Gaming Statechart Diagram	109
Figure 6.8: Order Processing Statechart Diagram	113
Figure 6.9: Questionnaire Processing Activity Diagram	116
Figure 6.10: Quadratic Roots Activity Diagram	119
Figure 6.11: Timer Functionality Sequence Diagram	122
Figure 6.12: Procurement Process Sequence	127
Figure 7.1: XML File	143
Figure 7.2: Architecture of the BCMT Tool	145
Figure 7.3: BCMT Structure Chart	146
Figure 7.4: User Interface Design	147
Figure 7.5: Use Case for the System	148
Figure 7.6: Base Metrics Values	158
Figure 7.7: Derived Metrics Values	159
Figure 7.8: Calculated Metrics Values in Text File	160

ACRONYMS AND ABBREVIATIONS

BCMT:	Behavioral Complexity Metrics Tool
BSC:	Balanced Scorecard
EAM:	Entity Attribute Metric Model
FP:	Function Points
GAM:	Goal Attribute Measure
GQM:	Goal Question Metric
LOC:	Lines of Code
MEFOOS:	Metrics-Based Maintainability Estimation Framework for Object-Oriented Software
MMM:	Metrics Measurement Model
MTBF:	Mean Time between Failures
PBCF:	Perspective Based Complexity Framework
PBMVF:	Perspective Based Metrics Validation Framework
SDLC:	Software Development Life Cycle
SPGQM:	Structure Prioritized Goal Question Metrics
SPSS:	Statistical Package for Social Sciences
TCF:	Technical Complexity Factor
UFP:	Unadjusted Function Points
UML:	Unified Modeling Language
XML:	Extensible Markup Language

DEFINITION OF TERMS

Analyzability: Quality attribute that assesses how a software system can be examined to identify defects.

Goal-Question-Metric (GQM): A structured approach to defining metrics by setting goals, formulating questions, and identifying corresponding metrics.

Metrics empirical validation: The process of evaluating a metric to determine its practical applicability.

Metrics Measurement Framework: A framework that provides guidelines and methodologies on defining and validating metrics.

Metrics reliability: The degree to which a metric yields consistent results when repeated under identical conditions.

Metrics theoretical validation: Process of evaluating the mathematical soundness of metrics.

Metrics validity: The extent to which a metric measures what it claims to measure.

Software metrics: Quantitative measure used to evaluate quality attributes such as analyzability and reliability.

Software quality attributes: Characteristics that define the overall quality of a software system, such as analyzability, reliability, and security.

Perspective: Scope of the attributes to be measured in UML behavioral complexity measurement.

CHAPTER ONE

INTRODUCTION

1.1 Background to the Study

In the modern digital age, the development and analysis of software systems have become significantly more complicated due to the ever-increasing complexity of software applications [1], [2]. As more features and functionalities are demanded by users, software systems continue to grow in scale and intricacy. This shift has led to a stronger emphasis in software engineering on modeling techniques that support the design of large, complex systems. Among the most widely adopted approaches for this purpose is the Unified Modeling Language (UML), which has become a standard for designing and documenting software architectures [3], [4], [5], [6], [7].

UML provides a consistent and systematic way of visualizing both the structure and behavior of software systems [8]. It combines advanced modeling principles and standardized symbols to help developers, designers, and stakeholders understand system designs at various stages of the software development life cycle. UML includes a variety of diagram types that allow users to represent different views of a system. The structural diagrams describe how different parts of a system are organized and connected, while behavioral diagrams focus on how the system behaves during runtime [9], [10], [11].

Sequence diagrams, statechart diagrams, and activity diagrams are among the most commonly used behavioral diagrams. These diagrams are important because they provide insights into how software systems respond to various events and how different components interact during execution. Sequence diagrams depict message flows and the order in which operations occur. This makes them suitable for use case modeling and understanding object communication [12]. Statechart diagrams show how an object changes from one state to another, depending on specific events. They are essential in modeling systems where behavior is dependent on internal states [12], [13]. Activity diagrams illustrate the flow of control from one activity to another and are useful in modeling workflows, business processes, and concurrent operations.

Over time, UML has undergone continuous refinement since its early versions. It originated in the 1990s when Grady Booch, James Rumbaugh, and Ivar Jacobson

unified their separate modeling approaches—namely Booch’s Method, OMT (Object Modeling Technique), and OOSE (Object-Oriented Software Engineering). This collaboration led to the creation of the first standardized UML specification. In 1997, UML was formally accepted by the Object Management Group (OMG) as an industry standard, and it has since been widely embraced in both academic research and commercial development environments [4], [6].

Despite its many advantages, the use of UML, especially its behavioral diagrams, poses challenges when the models become complex. As the size and functionality of a system grow, its diagrams tend to become crowded with elements and relationships, making them difficult to interpret. This results in higher diagram complexity, which can negatively impact the efficiency of development. To manage this complexity effectively, software engineers rely on measurement, which aids in quantifying important attributes of the software and its models [14].

The measurement process is relevant at all phases of software development, from initial design to post-deployment maintenance [15], [16], [17]. To make measurements consistent and useful, metrics frameworks are used. These frameworks guide developers in defining what to measure, how to measure it, and how to interpret the results. Over the years, several frameworks have been introduced to evaluate UML diagrams, providing tools to assess aspects like size, structure, readability, and complexity.

One quality aspect that can be assessed using these metrics is analyzability, which refers to how easily a system can be examined to identify causes of faults or understand how it works [13], [18], [19], [20]. Analyzability is a key component of overall software maintainability and directly influences the amount of time and effort required during system updates or debugging. Many researchers have proposed specific models and metrics for measuring the complexity of UML diagrams. These frameworks and models reveal the growing need to develop and refine metrics that provide better insights into how UML behavioral diagrams can be improved.

1.2 Problem Statement

Software designers are increasingly adopting UML behavioral diagrams such as use case diagrams, state machine diagrams, and sequence diagrams, among others, as a way of communicating the dynamic behavior of software [24], [25]. These diagrams

have inherent complexity which keeps on increasing whenever the diagrams are modified. High complexity makes it difficult to analyze these diagrams, and consequently, makes it difficult for designers to predict their analyzability and quality. Therefore, there is a need for new approaches to control and measure the complexity of these diagrams.

Researchers have in recent years proposed metrics definition frameworks, metrics validation frameworks, and metrics tools to assess software complexity. For instance, Macharia [21] proposed the Metrics-Based Maintainability Estimation Framework for Object-Oriented Software (MEFOOS), a framework that estimates the maintainability of object-oriented software components based on measurable quality characteristics. The MEFOOS framework is promising in the context of UML because it provides a structured approach for linking metrics to maintainability attributes and supports the integration of metrics across different levels of abstraction. However, it overlooks important attributes that are particularly relevant to UML behavioral diagrams, such as polymorphism, code complexity, abstraction, and class-level interactions factors that heavily influence system behavior at runtime and contribute to diagram complexity.

In [22], a language-independent coupling framework was proposed to evaluate the extent to which different coupling metrics measure similar properties. While this framework contributes to understanding coupling relationships within object-oriented systems, it is highly specialized for coupling analysis. It overlooks other important dimensions of run-time system behavior, such as polymorphism, dynamic binding, and method overriding, which are fundamental in object-oriented modeling and significantly affect the behavioral semantics of UML diagrams. Focusing solely on coupling provides an incomplete view of diagram complexity, especially for behavior-rich representations such as statecharts and sequence diagrams.

One recurring gap in the literature is the lack of a taxonomy of the perspectives of UML behavioral diagram complexity. As a result, measurement in this area is being done haphazardly, without a coherent or standardized framework that categorizes the different dimensions from which diagram complexity can be viewed. Existing metrics are often developed in isolation, without consideration of which behavioral aspects (e.g., transitions, control paths, interactions) are most critical to analyze. This

fragmented approach leads to inconsistencies in metric selection in evaluating behavioral diagrams.

Further, there is a substantial number of object-oriented metrics that have been proposed to assess the complexity of UML behavioral diagrams [23], [26–28]. In [23], cognitive complexity metrics for statechart diagrams are proposed. The metrics, however, overlook the characteristics of each element, such as the activities in a state and the guards in a transition. In [26], a modified cognitive complexity metric aimed at improving readability is introduced, but it does not factor in conditional statements in code. In [27], a set of slice-based cognitive complexity metrics to measure the effort required to understand a code segment is proposed. These metrics calculate only statements of interest, such as control flow statements, but overlook other key software attributes. Additionally, program slicing is resource-intensive and difficult to scale to large systems [27].

Finally, there is a lack of metrics tools that can automate the computation of all the relevant attributes of UML behavioral diagrams. As can be seen clearly, there is a lack of a classification framework based on perspectives of UML behavioral complexity, a suite of metrics for analyzing UML complexity attributes based on perspectives, a theoretical framework for validating UML complexity metrics, and tool support for the same.

1.3 Objectives

This section presents the general and specific objectives of the study.

1.3.1 General Objective

To develop a perspective-based complexity framework and metrics for predicting the analyzability of UML behavioral diagrams.

1.3.2 Specific Objectives

- i. To analyze perspective-based characteristics for defining a UML behavioral diagrams complexity framework.
- ii. To develop a theoretical validation framework for UML metrics based on perspectives.
- iii. To define a suite of theoretically sound metrics based on the complexity framework.

- iv. To develop a static analysis tool to automate metrics computation of complexity metrics for UML behavioral diagrams.
- v. To evaluate whether the proposed metrics can be useful predictors of the analyzability of UML behavioral diagrams.

1.4 Research Questions

- i. What perspective-based characteristics can be used to define a complexity framework for UML behavioral diagrams?
- ii. How can a theoretical validation framework be developed for UML metrics based on perspectives?
- iii. How to define a suite of theoretically sound metrics based on the complexity framework?
- iv. How can a static analysis tool be developed to automate metrics computation of complexity metrics for UML behavioral diagrams?
- v. How to evaluate whether the proposed metrics can be useful predictors of the analyzability of UML behavioral diagrams?

1.5 Significance of the Study

The purpose of this study was to develop a metrics framework informed by various perspectives to estimate the analyzability of UML behavioral diagrams, with attention given to both their structural composition and behavioral dynamics. As part of this effort, a complexity analysis framework was introduced to uncover the core contributors to complexity in behavioral diagrams. The study then defined a perspective-based metrics validation framework to evaluate whether behavioral metrics are theoretically sound and test their practicality. Additionally, the study defined and empirically validated complexity metrics, with findings indicating their effectiveness in predicting the analyzability of behavioral diagrams.

The frameworks proposed in this study offer significant value to researchers in software engineering and modeling by enabling them to pinpoint complexity-related features, design valid measurement methods, and adapt existing metrics to suit different types of UML behavioral representations, including activity diagrams, sequence diagrams, and statecharts.

In addition, designers and software engineers can adopt these metrics to evaluate and enhance the analyzability of behavioral diagrams used in software design.

To support metric analysis, a tool was implemented to automatically gather and compute complexity measures from behavioral diagrams. This allows software designers to make well-informed decisions, such as restructuring complex diagrams to improve system analyzability.

Furthermore, this study lays a foundation for future research in defining complexity metrics for behavioral diagrams, specifically in model-driven engineering, automated model analysis, and software quality assessment areas.

1.6 Scope of the Study

This study was designed to focus on specific aspects related to UML behavioral diagrams in the context of software complexity and quality evaluation. The areas covered in this study are as follows:

The objects of study included three UML behavioral diagrams namely, sequence, activity, and statechart diagrams. These diagrams were purposively selected due to their widespread use and their ability to model different dimensions of system behavior.

The study also focused on defining size and complexity metrics for UML behavioral diagrams, formulated within the boundaries of a proposed perspective-based complexity framework. These metrics are intended to capture behavioral characteristics that impact diagram complexity from different viewpoints relevant to developers and system analysts.

The study focused on maintainability, one of the primary external quality characteristics outlined in the ISO/IEC 25010 software quality model. Specifically, the study examined the analyzability sub-attribute, which reflects the ease with which diagrams can be assessed to understand and identify faults. Analyzability was selected because it is directly influenced by diagram complexity and is essential in early design phases, where understanding model behavior affects downstream development decisions.

1.7 Limitations of the Study

In this study, identifying and accessing UML industry experts for data collection and validation posed a significant challenge. UML experts are not always readily available and their involvement in academic research is often limited due to time constraints and proprietary concerns.

To address this issue, the snowball sampling technique was employed to identify and recruit experts whereby the initial respondents referred other qualified experts within their professional networks.

1.8 Contributions of the Thesis

This thesis made the following contributions to theory:

- i. Development of a Perspective-Based Complexity Framework (PBCF). This was developed and validated through an expert opinion survey, ensuring its relevance, inclusivity, and applicability in assessing behavioral diagram complexity.
- ii. Development of a Perspective-Based Metrics Validation Framework (PBMVF). The framework was operationalized using behavioral metrics.
- iii. Definition of a metric suite for behavioral diagrams. Nine distinct metrics were established to provide quantitative insight into the analyzability of UML behavioral diagrams. These metrics were formulated to reflect critical dimensions of behavioral complexity, providing a more precise evaluation of behavioral analyzability.
- iv. Theoretical validation of the proposed behavioral metrics. The theoretical soundness and validity of the proposed behavioral complexity metrics were evaluated using Weyuker's properties and the proposed perspective-based metrics validation framework.

The thesis also made the following contributions to practice:

- i. Validation of a perspective-based complexity framework. The framework was validated by collecting expert feedback through a structured opinion survey, ensuring its relevance, inclusivity, and applicability in assessing behavioral diagram complexity.
- ii. Evaluation of the perspective-based metrics validation framework. The framework was evaluated with existing behavioral metrics and compared

with existing metrics validation frameworks to ensure its comprehensiveness in assessing the theoretical soundness and practicability of behavioral metrics.

- iii. Development of the Behavioral Complexity Metrics Tool (BCMT). A dedicated software application, referred to as the Behavioral Complexity Metrics Tool (BCMT), was developed and deployed to automatically calculate complexity metrics for UML behavioral diagrams.
- iv. Empirical validation of behavioral metrics for analyzability prediction. The proposed behavioral metrics were assessed in a controlled laboratory setting to determine their practical applicability.

1.9 Organization of the Thesis

This thesis constitutes of nine chapters organized as discussed in the following section. Chapter one presents the introduction comprising of the background of the study, objectives and research questions, statement of the problem, significance of the study, scope, justification, limitations and contributions.

Chapter two presents a review of related literature covering background of UML behavioral diagrams, existing metrics measurement models, existing metrics validation framework, existing behavioral metrics, software analyzability, theoretical and conceptual framework, and the gaps identified.

Chapter three presents methodology used to carry out the research including research philosophy, strategy and design, research process, sampling strategy, research tools and their validation and validity testing, data analysis and presentation, and ethical considerations.

Chapter four introduces the developed Perspective-Based Complexity Framework (PBCF). It describes how the framework has been development to capture the unique features of behavioral diagrams that contribute to their complexity, and applicability of the framework though results of the expert opinion survey.

Chapter five presents the proposed perspective-based metrics validation framework (PBMVF). The chapter describes how the framework has been development by defining measurement attributes and their properties based on the unique features of

behavioral diagrams that contribute to their complexity, and evaluation of the framework.

Chapter six presents the development of a suite of behavioral metrics based on the perspective-based complexity framework. The chapter describes how the metrics have been defined for each behavioral perspective complexity. The chapter concludes by assessing the practicability and theoretical soundness by evaluating against Weyuker's nine properties and the perspective-based metrics validation framework.

Chapter seven presents the development of a static analysis tool named the behavioral complexity metrics tool (BCMT). The chapter describes the requirements of the tool, the architecture, and the use case of the tool. In addition, it describes the algorithms for the implementation of the metrics.

Chapter eight presents the empirical evaluation of complexity metrics applied to behavioral diagrams. It describes an experiment comprising both subjective and objective phases. The chapter presents validation results using descriptive and inferential statistics namely, correlation and regression analysis.

Chapter nine acts as the summary of the thesis. It presents the conclusion, recommendations, and future work. The content in this chapter is structured around the main objective and each of the specific objectives.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

This chapter provides an in-depth review of existing work related to UML behavioral diagrams, metrics measurement models, automation tools, existing complexity metrics, and methods of metric validation. In addition, the chapter highlights the gaps that served the basis of conducting this research.

2.2 UML Behavioral Diagrams

UML is a collection diagrams that comprehensively describe software intensive system [29], [30]. UML provides a set of diagrams that help software engineers to describe both the structural and behavioral features of a system [29], [30]. The visual representation provided by UML enhances communication and understanding to software modelers and business analyst. UML is associated with advantages such as helps in analyzing components of a system, provides a standard way documenting systems, supports automated code generation from diagrams, and helps in system testing and debugging [31]. Behavioral diagrams including sequence, activity, state, use case, and collaboration diagrams are utilized to represent the dynamic behavior of a system during execution.

2.2.1 Statechart Diagrams

Statecharts are vital in modeling complex behaviors of applications such as embedded systems, real-time processing, event-driven applications, and user interface workflows. A state machine diagram models the behavior of a system by depicting its possible states, transitions, triggering events, and corresponding actions [29], [31], [32]. They provide details of a system entity by showing how they respond to various conditions and stimuli over time. It starts with an initial state, progresses through different states and concludes at a final state. A transition is caused by a triggering event, and the system may execute specific actions or activities in response to these state changes [29], [31], [32].

The building components of a statechart are states, transitions, events, entry and action states and guard conditions. A state represents a particular condition in which an object exists for a certain period. A transition is the process by which an object moves from

one state to another in response to an event. It consists of an event trigger and a target state to which the object transitions. A transition may also include a guard condition that determines whether the transition should take place and an action that is executed during the transition. An action is an atomic computation that can be executed during a transition [29], [31].

The states include simple states, composite states and orthogonal states. Composite and orthogonal states contain nested sub-states and can execute specific actions upon entering or exiting. These actions may include entry action which is performed when an object enters a state and an exit action which is executed when an object leaves a state before transitioning to the next state. Transitions in a statechart diagram may include entry action executed when entering a state, exit action executed when leaving a state [31], [32]. An external transition occurs between two separate states while an internal transition takes place within the same state, without causing the object to exit the state.

The benefits associated with statechart diagrams are; provide a detailed view of an object's life cycle, captures all possible states and transitions, make it easier to model complex behaviors, and system analysts verify system correctness and validate expected behaviors. Figure 2.1 shows the components of a statechart diagram.

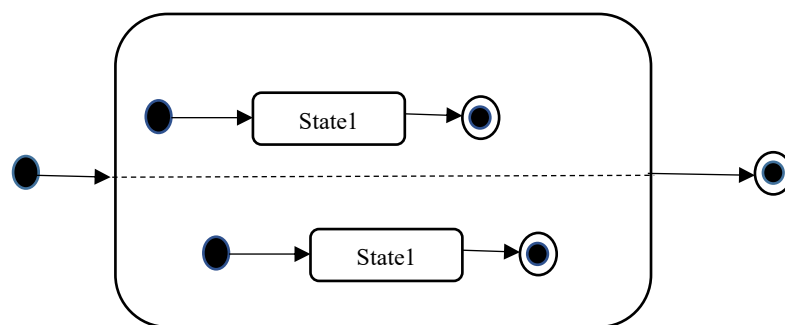


Figure 2.1: Statechart Diagram

2.2.2 Sequence Diagrams

Sequence diagrams are employed to represent how different system components exchange messages in a specific time order. They represent how the objects interact with each other in specific situations. It is used to describe interaction among classes in terms of an exchange of message over time [29], [31], [32]. The sequence diagram uses time to show the order of interactions. Time passes from top to bottom and

interaction which commence from the top of the diagram ends at the bottom. A lifeline denotes an object that takes part in an interaction within a sequence diagram. Actors symbolize external users who interact with the system, whereas objects denote internal system components responsible for sending messages [29], [31], [32]. Sequence diagrams incorporate various message types, including asynchronous, synchronous, return, and delay messages. An asynchronous message is illustrated using a line with an open arrowhead pointing right, indicating that the sender transmits the message and proceeds with the next task without awaiting a reply from the receiver. Synchronous message is represented as a line with a solid arrowhead pointing to the right. It represents a call operation where the sender sends a message and must wait for a response before proceeding. The receiving lifeline processes the request and eventually returns a response to the sender. A return message is depicted using a dashed line with a solid arrowhead pointing to the left. It represents the response from a procedure call. A delay line is depicted by a slanted dashed line, illustrating a time delay before the next action occurs. It represents a real-world scenarios where a process is intentionally delayed due to external factors, such as waiting for a response from a system or a time-dependent operation [32]. Figure 2 shows the components of a sequence diagram.

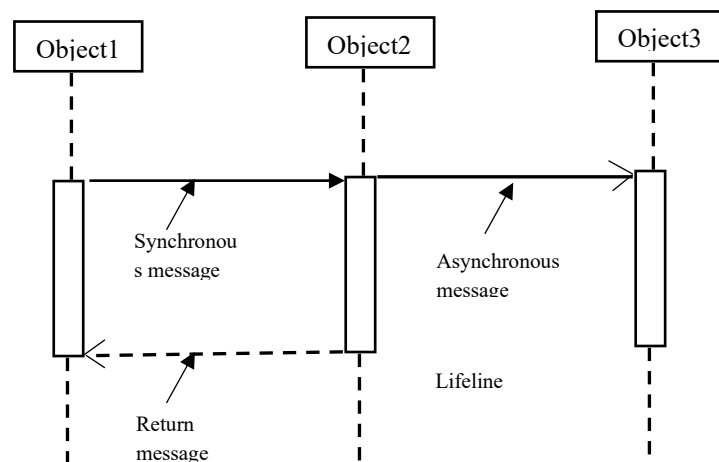


Figure 2.2: Sequence Diagram

2.2.3 Activity Diagrams

An Activity Diagram illustrates the flow of behavior within a system's processes. It provides a structured way to model business workflows and operational processes, helping analysts and developers understand how different tasks and decisions within

a system are executed. An activity diagram simplifies complex processes. The benefits associated with activity diagrams is their ability to present multiple conditions and choices within a workflow in a clear and visually structured way thus helps to determine the different paths that a process can take under varying conditions [29].

Activity diagrams consist of elements namely the Initial state represents the starting point of a workflow. It is represented as a black-filled circle and indicates where the process begins. In some cases, an activity diagram may have multiple initial nodes if a workflow has different possible starting points [115], [116], [117]. The control flows show the direction of the process, indicating the sequence in which activities occur [29]. These are represented by arrows that connect different action states within the diagram.

The action states are the individual tasks performed in a workflow. Each action state represents a specific action within the system. They are represented as rounded triangles. The Decision Nodes represent choice points where a workflow can take different paths based on certain conditions. These are shown as diamond-shaped symbols and allow designers to define multiple possible outcomes in a process [29].

The Guard Conditions specify the criteria that must be met for a workflow to proceed along a specific path. These conditions are usually placed within brackets and follow decision nodes to guide the execution of activities. The Fork Nodes are used to split a process into two or more parallel activities, meaning multiple tasks can be executed simultaneously. These nodes are essential for modeling workflows that involve parallel execution. The initial state indicate the completion of a workflow [118], [119]. They are represented by a black-filled circle within another circle and mark the termination of the activity sequence. Figure 2.3 shows the components of an activity diagram.

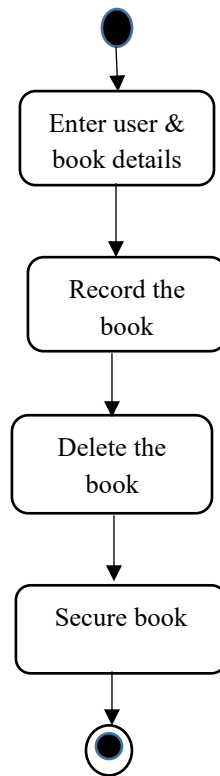


Figure 2.3: Borrow a Book Activity Diagram

2.2.4 Use Case Diagrams

Use case diagrams are dynamic diagrams that are used to define the functionality and graphics of a system in terms of actors, use cases, and relations [112]. They are considered for high-level functional requirement analysis of a system. The diagrams consist of actors, use cases, and relationships. The actors can be human users, internal applications, or external applications. Actors are the environment of the modeled system and use cases from this system. They are objects which interact with the target system [113]. The use cases define the specific goals or functions that the system performs in response to the actions of the actors. Each use case represents a distinct piece of functionality or a user goal. These are depicted as ovals and serve to describe the system's expected behavior. Relationship is an association, which connects an actor to a use case they participate in [112] [113]. Figure 2.4 depicts a use case diagram for a library management system.

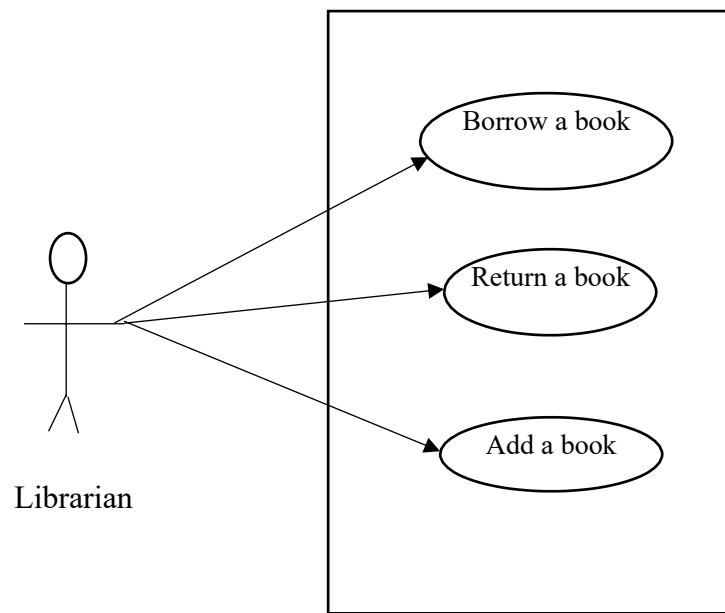


Figure 2.4: Library Management System Use Case Diagram

2.3 Software Measurement

This section presents software measurement theory and metrics definition frameworks.

2.3.1 Measurement Theory

Measurement theory explains how to give values to the features of software diagrams clearly and consistently. It helps researchers and developers decide what can be measured and how to measure it properly. In addition, it helps in defining, collecting, and analyzing data to help manage software complexity [120], [121], [122].

This theory is important because it ensures that the values we assign to software characteristics are meaningful. Based on this theory, models like the Entity-Attribute-Metric (EAM) model have been created. The EAM model connects elements of a diagram (entities), their properties (attributes), and the numbers we use to describe them (metrics). For example, it can be used to measure features of UML behavioral diagrams by defining clear steps for measuring each part [100]. This helps make software complexity measurements more accurate and easier to compare.

According to Fenton & Bieman [51], quality can be categorized into three key aspects namely, resources, processes, and products. The product category is further divided into internal and external attributes. Measurement theory ensures that both internal and external attributes of an entity are accurately measured.

The process of defining metrics follows the approach suggested by Fenton & Pfleeger [100]. Developing new metrics involves a systematic method which involves first, identifying the entity to be measured (e.g., a sequence diagram), then determining its measurable attributes (such as messages and lifelines), and finally, defining specific measures for evaluating each attribute. Figure 2.5 represents this metric definition process, as cited from Fenton & Pfleeger [100].

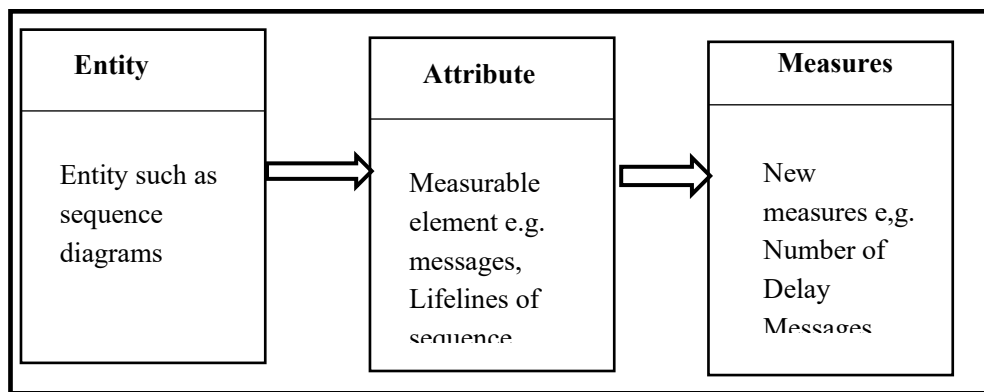


Figure 2.5: New Metrics Definition Process

2.3.2 Complementary Cognitive Perspectives

The measurement theory provides the technical grounding for defining and assessing software metrics. However, it may not fully capture the human dimension of understanding UML behavioral diagrams. The cognitive theories shed light on how individuals process visual models. This section introduces three theories namely, Cognitive Load Theory, Dual Coding Theory, and Mental Model Theory, to complement the measurement theory.

2.3.2.1 Cognitive Load Theory

Cognitive Load Theory (CLT) [97] focuses on the limitations of working memory when processing information. UML behavioral diagrams that include multiple branches, interactions, or parallel processes may require a high level of mental effort, making them harder to analyze. When the structure of a diagram exceeds the user's

cognitive capacity, the result is often confusion or misinterpretation. Metrics that account for the number of transitions, message flows, or nested activities can serve as indicators of this mental strain. Integrating CLT into complexity assessment helps explain why two diagrams with similar structural metrics may differ in perceived difficulty.

2.3.2.2 Dual Coding Theory

Dual Coding Theory [98] argues that people understand information more effectively when it is delivered using both images and words. In the case of UML diagrams, this suggests that adding meaningful labels, annotations, or accompanying descriptions can enhance understanding. Diagrams that rely solely on symbols without explanatory text can be visually clear but cognitively demanding. Metrics that measure the presence and quality of textual elements, such as labels on transitions or actor roles can help capture this dimension of analyzability. Applying DCT emphasizes the importance of multimodal communication in diagram interpretation.

2.3.2.3 Mental Model Theory

Mental Model Theory [99] describes how people mentally represent systems based on external information. For UML behavioral diagrams to be analyzable, they should support the creation of accurate internal models of system behavior. Diagrams that are poorly structured, lack modular clarity, or are semantically inconsistent may hinder this cognitive process. Therefore, metrics that reflect clarity of flow, logical sequencing, and component organization are useful in assessing whether a user can easily form a correct mental model. Incorporating this theory aligns metric development with the actual process of human reasoning.

2.3.3 Metrics Definition Frameworks

Metrics measurement frameworks offer structured approaches for formulating and validating software metrics. These frameworks serve as useful guides for defining measurement procedures and can significantly improve how organizations oversee growing software projects. Furthermore, they provide valuable insights for developers during software modification and reengineering processes. Numerous frameworks have been introduced in the academic literature, each focusing on assessing software complexity through specific quality attributes. For each framework, various metrics are associated with these attributes to facilitate the evaluation of software quality [34].

The Goal-Attribute Measure (GAM) methodology emphasizes the assessment of software products, development processes, and supporting resources [101]. In GAM, the process begins by recognizing the target users of measurement data and clarifying their goals. Next, relevant attributes and their underlying components are defined. These attributes are categorized into sub-attributes, which in turn serve as the basis for designing the corresponding metrics. This framework centers its attention on clearly structuring and detailing attributes while maintaining a tight focus on the measurement objects.

The Balanced Scorecard (BSC) framework [36] evaluates an organization's ability to realize strategic goals by aligning operations with its mission and long-term vision. Within BSC, strategic drivers that influence goal attainment are identified, and performance indicators are assigned accordingly. BSC introduces four dimensions: financial (shareholder perspective), customer (value delivery), internal (operational efficiency), and learning and growth (future capacity). The BSC metric derivation process starts with a review of the organizational mission and vision, followed by the definition of strategic goals, identification of supporting drivers, and assignment of relevant indicators [37].

Goal Question Metric (GQM), originally developed by Basili [38], remains one of the most widely used measurement frameworks. GQM organizes measurement into three hierarchical layers: the goal layer defines what is to be achieved, the question layer articulates specific inquiries based on those goals, and the metric layer consists of quantifiable measures linked to the questions. The GQM model assumes that meaningful evaluation requires predefined goals, structured questions targeting key attributes, and metrics that offer measurable responses [23], [38].

The Entity-Attribute Model (EAM), proposed by Fenton and Pfleeger [100], outlines a three-step process: identifying measurable entities, determining measurable characteristics of those entities, and developing metrics to evaluate those characteristics. Entities may include software components, and their attributes may cover features such as size, structure, or behavior. EAM categorizes entities into processes, products, and resources, where processes denote software development activities, products refer to the deliverables of those activities, and resources include hardware and software assets used during development [100].

The MEFOOS framework (Metrics-Based Maintainability Estimation Framework for Object-Oriented Software), introduced by Macharia et al. [21], targets maintainability in object-oriented systems. It measures key characteristics like understandability, portability, and modifiability using metrics such as coupling, cohesion, and inheritance. Although MEFOOS has been tested on practical applications and shown to be effective, it overlooks additional object-oriented features such as polymorphism, abstraction, code complexity, and class size.

Yue and Li [40] developed a MOF-based framework that allows for the formal specification of quality attributes such as completeness, correctness, and redundancy in modeling languages built on the Meta Object Facility (MOF). The framework enables the definition of metrics at various complexity levels, ranging from broad to detailed. Metrics are based on elements within the language's metamodel, offering access to the structural details and their relationships. While this model has been successfully used to define metrics for UML diagrams like class, sequence, and state diagrams, it lacks a generic user interface and does not support metric definitions beyond UML or other MOF-based languages [40].

Metrics Measurement Model (MMM) by Rajesh and Chandrasekar [41] outlines six core steps. The process begins by selecting a sample model, such as a class diagram and identifying relationships. Following this, quality-relevant metrics are gathered and measured. The average modifiability of the design is then calculated before the final validation step. This framework supports attributes like extendibility, reusability, and understandability but fails to incorporate key factors such as reliability, performance, and maintainability [42].

GQM+Strategies (GQM+S-EI), introduced by Petersen et al. [43], extends the standard GQM by integrating the iterative elicitation of strategic goals and objectives. The model enables practitioners to better align measurement programs with organizational strategies and decision-making. Although it has been validated via case studies, the model lacks detailed guidance on how to structure the collected data. Its application across different organizational settings is also underexplored.

GQM-DSFMS (Decision Support Framework for Metric Selection) by Gencel et al. [42] builds upon GQM principles to assist in selecting the most relevant metrics while

accounting for constraints such as organizational goals, changing priorities, and available resources. The framework includes four key stages: identifying goals, eliciting questions, mapping metrics to questions and attributes, and making informed decisions. Despite its usefulness, GQM-DSFMS does not include indicators to track measurement progress or effectiveness.

Amara et al. [44] proposed a Reliability Measurement Framework to enhance reliability across all stages of the software development lifecycle. This framework includes a combination of tools, methods, and metrics that aim to reduce the cost and effort of error correction. However, it has not yet undergone validation in practical scenarios, limiting its demonstrated effectiveness.

Tempero and Ralph [22] presented a Framework for Coupling Metrics, focusing on defining coupling in a way that is both precise and language-independent. This framework clarifies how different coupling metrics measure similar or distinct phenomena and resolves ambiguities found in incomplete metric definitions. Though it has been applied successfully to assess object coupling, it does not account for runtime properties such as polymorphism.

The GQM+Strategies framework by Basili et al. [47] expands upon GQM by establishing a connection between software metrics and business-level objectives. The model enables metrics to be interpreted in alignment with overarching organizational strategies. Although the framework is comprehensive, it lacks a well-defined approach for stakeholder engagement and does not offer tool support for managing the measurement process [50].

Mandić and Vozdenović [48] extended GQM+Strategies by introducing causal analysis, which allows users to model dependencies among goals and identify their influence on each other. This feature supports the formulation of more effective strategies. Nevertheless, the framework does not include time-sensitive data, lacks analytical tools for grid-based evaluation, and requires further empirical testing in real-world organizational contexts.

The Structured Prioritized GQM (SPGQM) by Tahir and Gencel [49] enhances traditional GQM by introducing prioritization, traceability, and reusability of goals,

questions, and measures. The framework uses structured templates to define goals and questions, incorporating fields such as purpose, category, stakeholders, and scenarios. While SPGQM has been successfully tested in case studies, it does not integrate advanced prioritization or decision-making techniques for selecting the most impactful measures.

2.4 Analyzability

Analyzability is one of the important quality attributes used to evaluate software systems. It shows how easy it is to understand, check, or find problems in a software design [57]. This makes it a key factor in software maintainability, as systems that are easier to analyze are also easier to improve and fix. In the context of UML behavioral diagrams, assessing analyzability is useful during design because it can help spot diagrams that are too complex or unclear. As a result, analyzability can support better design decisions and reduce the cost of later maintenance.

2.4.1 Basic Concepts of Analyzability

Analyzability means how easily a software model or diagram can be tested or fixed for errors. It depends on aspects such as understandability, detectability, and traceability. Understandability refers to how clearly a diagram shows the system's behavior or structure. A diagram is considered easy to understand if someone can look at it and quickly know what it represents. A model that is too complex or disorganized is hard to interpret, which slows down the process of reviewing or analyzing it.

Detectability refers to how easily errors, missing parts, or wrong connections can be found in a diagram. When a model is well-designed and follows standard rules, problems are easier to notice. Detectability helps in identifying areas that may cause confusion or lead to software bugs later in development. A clear diagram makes it easier to check for mistakes and verify that the behavior is correct.

Lastly, traceability refers to the ability to follow the connection between different parts of a model, or between the model and the system's original requirements. Traceability helps ensure that all parts of the software are covered and that nothing is missing. It also makes it easier to see how changes in one part of the model affect others. For example, in a UML diagram, traceability allows you to follow the flow of actions, states, or messages and see how they relate to system goals or user needs.

These basic concepts are widely used in evaluating software models because they affect how useful and practical a diagram is. A model that is easy to understand, check, and trace will help in making better design choices, reducing future errors, and make maintenance easier. According to the ISO 9126 [56] and ISO/IEC 25010 standard [57], such qualities are part of good software engineering practice and help improve the overall quality of software systems.

2.4.2 Analyzability and UML Behavioral Diagrams

UML models need to be easy to study and understand during system design and review. If a diagram is too complex or poorly organized, it becomes difficult to check for mistakes, follow system behavior, or suggest improvements. This can lead to delays in development and make maintenance more difficult.

Behavioral diagrams often become harder to work with when they include too many elements such as actions, states, or messages. For instance, activity diagrams with many branches, sequence diagrams filled with lifelines and messages, or statechart diagrams with numerous transitions can be overwhelming. These types of diagrams reduce clarity and make it harder to follow the intended logic.

To solve this, researchers have developed several metrics and frameworks to help assess how easy or difficult it is to analyze UML diagrams. These tools often count the number of elements, check how they are linked, and identify parts that increase complexity. Diagrams that are well-structured and contain fewer, clearer elements are usually easier to review and analyze.

The metrics and frameworks help improve the readability and quality of UML models. It also supports better decisions during design and makes testing and maintenance easier. As stated in ISO/IEC 25010 [57], systems that are easier to check and correct are considered more maintainable and reliable.

2.4.3 Software Quality Models and Analyzability

A quality model can be described as a set of characteristics and association between them, which form the basis for determining quality [123], [124], [125], [126]. Software quality assurance is determined by quality models such as Boehm's,

McCall's, FURPS, ISO 25010, Dromey's. These quality models are categorized as per their generation and are hierarchical [34].

2.4.3.1 McCall's model

McCall's model [52] is one of the oldest frameworks for evaluating software quality. It categorizes software quality factors into two main levels namely, external parameters (which can be measured directly), and quality (which can be evaluated either objectively or subjectively). This model considers both technical and user-oriented aspects of software performance.

The model further divides software quality into three major categories. The first category, product operation, focuses on how well the software functions during its execution. This includes accuracy, which ensures that the software produces correct and reliable results; reliability, which measures the system's ability to function correctly over an extended period; efficiency, which evaluates how optimally the software utilizes system resources; integrity, which safeguards data and system security; and usability, which assesses the ease with which users can interact with the software [52].

The second category, product revision, emphasizes the ability of software to evolve and improve over time. This includes maintainability, which refers to the ease with which developers can modify the software to fix defects; testability which examines how easily the software can be tested for correctness and reliability; and flexibility which evaluates the extent to which the software can accommodate new changes without requiring extensive modifications [52].

The third category, product transition, deals with the software's adaptability to different environments. This includes portability, which ensures that the software can operate on various platforms and hardware configurations; reusability, which measures how easily components of the software can be repurposed for other applications; and interoperability, which evaluates the software's ability to integrate and function seamlessly with other systems [52].

McCall's model provides a foundation that guides software development and evaluation. In addition, it has formed the advancements in software quality

assessment. The model does not offer an explicit approach to facilitate the analyzability attribute.

2.4.3.2 Boehm's Model

Boehm's quality model [53] expands upon McCall's framework by introducing a hierarchical structure for software quality characteristics. The model introduces three distinct levels namely, primitive, intermediate, and high-level characteristics. These levels provide a systematic approach to evaluating software quality. At the primitive level, it considers fundamental software quality attributes that form the foundation for higher-level assessments [53]. These attributes are then grouped into intermediate-level characteristics, namely usability, testability, maintainability, portability, flexibility, human engineering, and understandability. Usability focuses on how effectively users can interact with the software, while testability evaluates how easily the software can be tested for errors. Maintainability assesses the ease of modifying and updating the software, and portability examines its ability to function across different environments. Flexibility ensures that the software can be adapted to new requirements, while human engineering assesses the software's ability to provide an intuitive user experience. Understandability focuses on the cognitive ability of developers and users on software's structure and functionality [53].

At the highest level, Boehm's model integrates these intermediate characteristics into a holistic evaluation of software quality. In addition, the model considers hardware performance, thus useful for evaluating software that operates in hardware-dependent environments, such as embedded systems and real-time applications. The Boehm's model does not explicitly consider the analyzability attribute.

2.4.3.3 FURPS Model

FURPS model introduces a more structured classification of software quality, dividing it into five attributes namely functionality, usability, reliability, performance, and supportability [54]. Functionality measures the extent to which the software meets specified requirements. It is further subdivided into software capabilities, which assess the range of features provided by the software, and feature completeness, which evaluates whether all necessary functionalities are implemented.

Usability focuses on the user experience, assessing how easily users can interact with the software. It considers factors such as the intuitiveness of the user interface, ease of navigation, and the availability of user training and documentation [54].

Reliability evaluates the software's ability to operate without failure over time. This includes accuracy, which ensures that outputs are correct; the mean time between failures (MTBF), which quantifies the average time the software operates before encountering an issue; and failure severity, which determines the impact of failures on system performance [54].

Performance assesses the efficiency of the software under different conditions. It considers response time, which measures how quickly the software processes user inputs; resource utilization, which examines how efficiently the software manages system resources; and operational efficiency, which assesses the overall performance under different workloads [54].

Supportability focuses on the software's maintenance and adaptability. This includes serviceability, which determines how easily the software can be repaired or updated; adaptability, which evaluates its ability to accommodate new requirements; testability, which examines how easily defects can be identified and fixed; and compatibility, which assesses how well the software integrates with other systems [54].

FURPS model is widely used in software engineering due to its well-defined classification system, making it a valuable tool for both developers and quality assurance teams. However, it does not comprehensively offer an approach on assessing the analyzability attribute.

2.4.3.4 Dromey's model

Dromey's model [55] provides a unique approach to software quality by linking software attributes to software product properties. This model emphasizes the relationship between quality attributes and subparameters, ensuring that specific characteristics can be systematically evaluated.

Dromey identified three components in his model namely, product properties, high-level quality parameters, and the relationship between product properties and quality

attributes [55]. Product properties define the structural and functional characteristics of the software and are further subdivided into descriptive properties, which provide insights into the software's composition; contextual properties, which assess the software's suitability for different environments; correctness properties, which ensure the accuracy and reliability of the software; and internal properties, which describe the internal organization and structure of the software [55].

High-level quality parameters define the overall quality of the software, including reliability, maintainability, and efficiency. The relationship between product properties and quality attributes allows for a detailed assessment of how specific software components contribute to overall quality [55].

Dromey's model is particularly valuable for software development teams seeking to establish clear links between software design decisions and quality outcomes. The Dromey's model does not explicitly accommodate analyzability attribute.

2.4.3.5 ISO 25010

ISO 25010 is an advanced software quality model as a result of the modification of the ISO 9126 model. It categorizes software quality into eight distinct attributes namely functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. These attributes are further subdivided into sub attributes [57].

Functional suitability evaluates whether the software meets the intended functional requirements, while performance efficiency assesses responsiveness, resource utilization, and system throughput. Compatibility evaluates the software's ability to integrate with other systems, and usability focuses on the ease with which users can interact with the software [57].

Reliability evaluates the software's stability and fault tolerance, ensuring that it executes as expected under different conditions. Security evaluates the software's ability to protect against threats, unauthorized access, and vulnerabilities. Maintainability is further categorized into modularity, which ensures that components can be modified independently; reusability, which evaluates the potential for reusing

software components; analyzability, which evaluates the ease of diagnosing issues; and changeability, which examines the effort required for modifications [57].

Portability ensures that the software can function across different platforms and environments. ISO 25010 integrates internal, external, and quality-in-use parameters to provide a comprehensive assessment framework. It provides a standardized approach for ensuring high-quality software development [57].

Maintainability includes aspects, such as modularity, reusability, ease of analysis, ability to change, ease of modification, stability, test support, and adherence to standards. These aspects are illustrated in Figure 2.6.

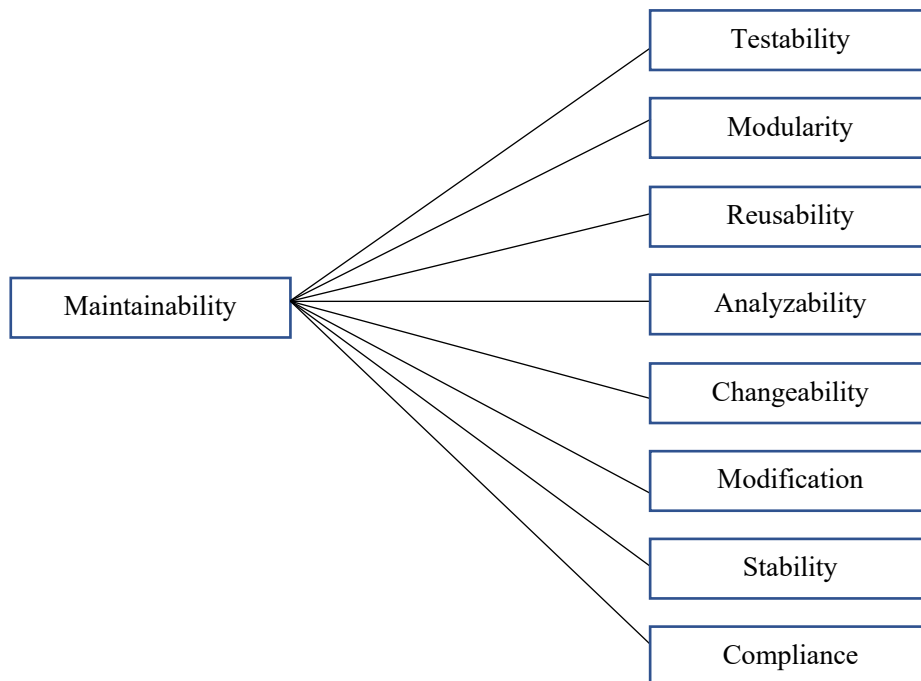


Figure 2.6: ISO-25010 Maintainability Sub-characteristics [57]

2.5 Software Metrics

Software metrics serve as indicators for software attributes, aiding in quality management, budget planning, and cost estimation. They are classified into product, process, and project metrics [58], [59], [60], [61], [127], [128]. Product metrics are subdivided into static (evaluated during SDLC) and dynamic (assessed during runtime). Process metrics examine development team performance, including error

rates and defect removal time. Project metrics focus on aspects, such as time, effort, and cost [33], [61].

2.5.1 Line of code (LOC)

Lines of code (LOC) is a traditional size metrics for assessing the program complexity by counting lines of a program codes. The LOC can be calculated in the following ways:

- i. LOC: calculates all lines of code.
- ii. SLOC: calculates source lines of code.
- iii. CLOC: calculates comment lines of code
- iv. S&CLOC: calculates source lines of code with comment lines.
- v. BLOC: calculates blanks lines of code.
- vi. PLOC: calculates physical lines of code.
- vii. LLOC: calculates logical lines of code

LOC has been criticized for only counting the lines, hence not considering control structures like branches, loops and jumps and also, it is calculated in different ways hence values calculated from different assessors [68]. Therefore, LOC cannot be relied upon on evaluating behavioral diagrams.

2.5.2 Metrics by Halstead

Halstead's Software Science [64] metrics focuses on operators and operands as contributors to software complexity. Halstead's aim was to demonstrate that the number of operands and operators is highly correlated to the number of errors found in a program.

The metrics include the Number of distinct operators, Number of distinct operands, Number of occurrences of operators, and Number of occurrences of operands [61].

Halstead's software metrics are sound from a measurement theory perspective. However, it is difficult to calculate since they fail to differentiate the number of operators and operands adequately [65]. Therefore, it is not appropriate for analyzability of UML behavioral diagrams

2.5.3 McCabe's Cyclomatic Complexity

M McCabe's Cyclomatic complexity [62] metric is used to evaluate the control flow structure of a program. This metric is widely adopted by researchers since its computation is straightforward. The metric utilizes control structures to make a control flow matrix which is used to generate a connected graph. The graph illustrates the control paths within a module according to McCabe's [62]. The metric is calculated as follows:

$$V(G) = e - n + p$$

Where e is the number of edges, n is the number of nodes, and p is the number of connected components in a graph, and G represents the graph.

Alternatively, McCabe's complexity can be calculated directly from the source code without using a graph for single entry or single exit modules as:

$$V(G) = \text{Number of Decisions} + 1.$$

In addition, the metrics also account for nested hierarchical structure complexity by summing connected component complexity. Modules with cyclomatic complexity less than 10 tend to have fewer bugs than those with a cyclomatic complexity greater than 10 [62].

M McCabe's Cyclomatic complexity measure is limited because of assigning same weights to both branch and loop statements [66]. Further, it only considers the control structure of the software [67].

2.5.4 Function points (FP)

Allan Albrecht [63] developed the Function Points (FP) metric for assessing software size and complexity. The metric focuses on the functionality delivered to the end user rather than the programming effort.

FP evaluates the relative size of a software by analyzing the information processing size and technical complexity factor. It is calculated as follows:

$$\text{Function Points} = \text{UFP's} \times \text{TCF}$$

Where UFP represents the sum of all system components seen by the end user (i.e. number of input types, number of output types, number of inquiry types, number of logical internal files and number of interfaces) while TCF is the degree of influence of 14 components.

Function point metric is difficult to calculate because the values are calculated manually and also requires experienced personnel to accurately assess [68].

2.5.5 Existing Metrics for UML Behavioral Diagrams

Kim & Boldyreff [102] proposed software measures for features of UML modelling. The metrics were classified into model, class, message, and use case metrics. The Number of the packages in a model (NPM), Number of the classes in a model (NCM), Number of actors in a model (NAM), Number of use cases in a model (NUM), Number of objects in a model (NOM), Number of the messages in a model (NMM), Number of the associations in a model (NASM), Number of aggregations in a model (NAGM), and Number of the inheritance relations in a model (NIM) were model metrics. The use case metrics are: Number of actors associated with a use case (NAU), Number of messages associated with a use case (NMU), Number of system classes associated with a use case (NSCU).

Omar [87] proposed a concept for defining complexity metrics for UML models. The approach involves assigning a complexity rate for every element within a UML diagram. The complexity Rate (Co_{Rate}) is to discriminate the level of complexity for each and every part and element in each design from primitive, simple, to complex. The primitive represents the simplest complexity assigned a rate of one (1), the moderate complexity is simple with complexity of two (2), and the most complicated element is assigned a complexity rate three (3). Omar [87] used this approach to define (Att_{Comp}), ($method_{Comp}$) and ($Asso_{Comp}$) metrics for UML class diagrams. In addition, he defined complexity metrics for statechart diagrams namely, the (SM_{Comp}) which is the sum of state complexity ($State_{Comp}$) and transition complexity ($Trans_{Comp}$).

Even so, Omar [87] used this approach to evaluate the complexity of only UML class and statechart diagrams and did not define metrics for other UML diagrams such as sequence, activity and collaboration diagrams. In addition, he did not factor all the elements in the computation of the metrics for the diagrams.

2.5.6 Existing Cognitive Complexity Metrics for UML Behavioral Diagrams

Cognitive complexity metrics are measures used to assess the human effort required to comprehend a given code or software system. Researchers have defined several cognitive complexity measures:

2.5.6.1 Cognitive Complexity Metrics for Statechart Diagrams

King'ori et al.[23] proposed a suite of cognitive complexity metrics for UML diagrams based on statechart diagrams. The metrics defined are Weighted Number of States (WNS), Weighted Number of Transitions (WNT) and the Weighted Number of Events (WNE).

The Weighted Number of States (WNS) is a function of a category of state and the weight assigned to the state. It is denoted as:

$$WNS = \sum_{i=1}^n (W_i X_i)$$

The Weighted Number of Transitions (WNT) measure is as a result of the modification of the Cognitive Functional Size (CFS) measure [85]. WNT is the function of types of transitions in the statechart diagram and the cognitive weights (W_j). It is calculated as:

$$WNT = \sum_{j=1}^n (W_j T_j)$$

The Weighted Number of Events (WNE) is a function types of events in the statechart diagram and the weight assigned to the type of event (W_k). It is calculated as follows:

$$WNE = \sum_{k=1}^n (W_k E_k)$$

Even though the metrics have been tested through theory and experiments, they still have limitations because they do not consider the detailed features of each element, like actions inside states or guard conditions on transitions.

2.5.6.2 Cognitive Functional Size (CFS)

Shao & Wang [85] proposed a cognitive functional size. The CFS measure was based on internal architectural control-flows, output data and input data. The CFS metric is denoted as:

$$CFS = (N_i + N_o) * W_c$$

Where N_i and N_o are the sum of inputs and outputs of a software respectively. The W_c is the weight given to various control-flow structures.

CFS has drawbacks. For example, it does not consider essential factors such as inheritance and excludes the details found in operators and operands thus cannot be used to measure the cognitive complexity of UML behavioral diagrams [104].

2.5.6.3 Class Complexity

Misra & Akman [102] defined the class complexity metric using cognitive weights. The measure is used to assess the class complexity in object-oriented system. It works by connecting a certain weight with each method. After allocating weight it calculates the weights of all methods. Therefore, complexity of a whole class is determined. If the complexity of whole system is to be calculated then weights of classes are added in the same level while if there are child classes, then the weights are multiplied. The class complexity is denoted as:

$$CC = \prod_{j=1}^m \sum_{k=1}^n (W_{cjk})$$

Where, W_c is the weight of a class, M represents the depth in object-oriented code and n represent the number of classes.

The class complexity metric is easy to calculate, however it only considers the number of methods, classes and subclasses and excludes the internal structure of the method [103].

2.5.6.4 Cognitive Information Complexity Measure (CICM)

The Cognitive Information Complexity Measure (CICM) was proposed to aid in understanding the cognitive information complexity and the information coding

efficiency of a program [105]. The CICM relies on the cognitive weight of internal BCSs, identifiers, operators and lines of code (LOC) [105]. The measure is denoted as follows:

$$\text{CICM} = \text{WICS} * \text{Wc}$$

WICS refers to the sum of the weighted information across all lines of code, while Wc represents the cognitive load assigned to each basic control structure.

The measure has been criticized for being difficult to calculate if a software contains many lines of code [103].

2.5.6.5 Modified Cognitive Complexity Measure (MCCM)

Misra[106] proposed the modified cognitive complexity measure (MCCM) which was a modification of cognitive functional size (CFS). The measure was based on the cognitive weights caused by the basic control structures, number of operators and operands. The measure is denoted as follows:

$$\text{MCCM} = (\text{Ni1} + \text{Ni2}) * \text{Wc}$$

N_{i1} represents how many times operators appear, N_{i2} shows the total count of operands used, and Wc stands for the cognitive weight given to each basic control structure.

The measure is practicable as stated by the cognitive informatics works. However, the measure gives high complexity values [67], [103].

2.5.6.6 Cognitive Program Complexity Measure (CPCM)

Cognitive Program Complexity Measure (CPCM) was proposed by Misra [107]. The metrics considered the total occurrences of inputs, outputs and cognitive weights. It is denoted as follows:

$$\text{CPCM} = \text{Soo} + \text{Wc}.$$

Where S_{oo} represent the sum total of input and output variables and W_c is the cognitive weight of the basic control structure.

CPCM has limitations. For instance, it does not factor out complexity due to operators in its computation [67]. Moreover, it is unclear and misleading [103].

2.5.6.7 Slice-based Cognitive Complexity Metrics

Alqadi & Maletic [27] proposed a set of slice-based cognitive complexity metrics to measure the effort of understanding code segment:

Slice size. This measure provides an indicator of the cognitive effort required to comprehend a particular slice. Program slice includes program artefacts that are data and control dependent to the function or variable of interest. Slice size for a module x can be defined as:

$$X = \sum_{i=1}^k (S_i / k)$$

Where S is the number of statements in slice i, and k is the number of slices in the module

Slice Identifier: The metrics focuses on only parts that are relevant with respect to a particular feature or variable. For a module x, the slice identifier can be defined as:

$$X = \sum_{i=1}^k (SI_i / k)$$

Where S is the number of identifiers in slice I, and k is the number of slices in module x.

Slice Count: The count of slices focuses on the overall cognitive complexity of source code parts. Program segment that has a high number of slices will have high number of features leading to higher concentration of identifiers, method invocations, and relevant control and data dependencies. The metric is defined as:

$$\text{Slicecount}(x) = k,$$

Where k is the number of slices in x.

SliceSpatial: Spatial complexity metrics account for the difficulty of reading the source code of a program for understanding, in terms of the lexical distance that the maintainer is required. The metric is defined as:

$$SliceDistance(i) = Sm_i - \frac{Sni}{q},$$

Where Sm is the number of the first statement in slice I , sn is the line number of the last statement in slice i , and q is the module size in LOC.

These slicebased metrics use program slicing which is time consuming to calculate large systems. In addition, program slicing focus on a particular variable of interest hence they cannot be used to measure UML behavioral diagrams.

2.5.6.8 Cognitive Complexity for Business Process Model (CCBP)

According to Muketha [108], the Cognitive Complexity for Business Process models (CCBP) is a metric that uses the number of inputs, outputs, and the cognitive weights (Wc) of structured activities in a business process model. This measure is calculated as follows:

$$CCBP = (NOIA + NOOA) * Wc.$$

Where $NOIA$ is the number of input activities, $NOOA$ is the number of output activities and Wc is the total cognitive weight of all control-flow blocks within the model. The measure has been validated both theoretically and empirically. Even so, it is limited since it can only be used to assess the cognitive complexity of business process models.

2.5.6.9 Weighted Class Complexity (WCC)

Misra & Akman [109] modified the CC metric and defined a new metric as weighted class Complexity (WCC). Object oriented program structure rely upon the attributes and methods in a class. The complexity of a class is calculated by the sum total of weights of methods and attributes. The WCC is calculated as follows:

$$WCC = N_a + \sum_{p=1}^s MC_p$$

Where N_a is the number of attributes in a class and MC is the method complexity. The weighted Class Complexity incorporates the complexity due to internal structure of methods and attributes. The measure is limited since it fails to consider the critical concepts of Object-Oriented programming such as inheritance, encapsulation, overloading and Polymorphism [103].

2.5.6.10 Attribute weighted class Complexity (AWCC)

Arockiam & Aloysius[70] proposed a new metric called attribute weighted class complexity (AWCC). As cognitive load is distinct for individual attributes, so in AWCC different cognitive weight are assigned to attributes. These weights were formulated by considering the efforts needed to comprehend the data types. AWCC is calculated as follows different:

$$AWCC = \sum_{i=1}^n AC_i + \sum_{j=1}^m MC_j + \sum_{k=1}^m ICC$$

Attribute weighted class complexity is limited since it only adds the attribute complexity to the extended weighted class complexity and fails to consider other object-oriented features [67].

2.5.6.11 Improved Cognitive Complexity Metric (ICCM)

Esther et al.[110] proposed improved cognitive complexity metric (ICCM) for object-oriented code. According to the researchers, the naming of variables in a code is crucial in comprehending the code. The researchers found that arbitrarily named variables (ANV) rise the difficulty of comprehending three times greater than meaningfully named variables (MNV). ICCM is calculated as follows:

$$ICCM = \sum_{k=1}^{locs} \sum_{v=1}^{LOCS} (3ANV) * W_C$$

Where the first summation is the total lines of code. ANV and MNV show the arbitrarily and meaningfully named variables respectively, while W_C is the weight of BCS. ICCM fails to consider OO aspects such as class complexity, inheritance and coupling [67].

2.6 Metrics Tooling

Tools that support metrics are essential for gaining acceptance of any proposed measurement in the software field. These tools help in gathering data, performing calculations, and showing the results clearly [129].

2.6.1 SLOC Count

Source Lines of Code (SLOC) Count [137] was developed to count SLOC in a GNU/Linux distribution, but it can be used for counting the SLOC of arbitrary software systems. This tool has the capability of report generation, collecting the data generated and present it in several formats according to the requirement

However, this tool only calculates the lines of code (LOC) metric and relies on that to estimate the time, effort, and money required to develop the software. It is not a web-based application and hence, a user need to download and install it before using or editing the tool.

2.6.2 NDepend

NDepend tool [138] is an extension for Visual Studio for .net managed code. It currently supports 82 code metrics which includes Lines of Code, Cyclomatic Complexity, Coupling, Nesting Depth, and Rank. It has the ability to detect dependency cycles which can be eliminated to achieve higher code maintainability and can compare two versions of the code base to check the modifications and to warn the user if there are any incompatible changes in the code.

This is not a web-based application. The NDepend extension has to be downloaded and then installed for Visual Studio. In addition, the user cannot modify or customize the tool to add new modules that can calculate other metrics or statistics which are not originally calculated by the tool [21].

2.6.3 Code Counter Pro

Code Counter Pro [139] is a tool that counts all the programming lines, SLOC (source lines of code) and KLOC (thousands of lines of code) automatically, calculates the productivity of the software development team, measures Function Points through Backfiring and measures the percentage of comments in the source code. It can handle several types of source code including that of C, C++, Java, C#, HTML, XML, and Pascal etc.

The tool mainly concentrates on the lines of code metric, doesn't have the ability to calculate other metrics related to the source code and the users are not permitted to modify and customize the tool.

2.7 Metrics Validation Approaches

A software metric should be validated both theoretically and empirically to ensure its validity [18], [21], [130], [131], [132].

2.7.1 Theoretical Frameworks

2.7.1.1 Weyuker's nine properties

Weyuker's [73] proposed a framework consisting of nine properties designed to validate complexity metrics. These properties are outlined as follows:

Property 1 Noncoarseness: This property states that UML diagrams that are different from each other should not produce the same metric value. Each unique diagram should result in a distinct measurement.

Property 2 Granularity: This property states that there should be finite number of UML diagrams with similar metric value.

Property 3 Nonuniqueness: This property implies that two UML diagrams can have the same metric value but differ in the way they name some of their elements.

Property 4 Design Details are important: This property implies that design details are useful in determining the metric of a UML diagram. This means that two diagrams P and Q could look similar on the outside but have varying internal designs and implementations.

Property 5 Monotonicity: Monotonicity means that the metrics as a result of combining two UML diagrams can never be less than the complexity of the two initial UML diagrams.

Property 6 Nonequivalence of interaction: Nonequivalence suggests that if two equal UML diagrams, and are combined with the third same diagram, they yield distinct complexity value from the original complexities.

Property 7 Permutation: The property asserts that a measure is sensitive due to the permutation of elements.

Property 8 Renaming: Renaming implies that the metric should remain unchanged as a result of renaming elements in a diagram.

Property 9 Interaction increases complexity: Interaction implies that combining two UML diagrams increases the complexity metric value.

Weyuker did not consider other metric types such as size and coupling in his framework. To address this limitation, Briand et al. [39] introduced a framework with various properties to validate metrics across five categories namely, size, length, cohesion, coupling, and complexity.

Weyuker's properties are not well-suited for non-complexity metrics [108]. However, they remain widely used for metric validation. For example, Chidamber and Kemerer [74] employed Weyuker's properties to validate object-oriented design metrics. Additionally, Halstead's software science metrics and McCabe's complexity metrics were evaluated using these nine properties. Further, Ndia [74] and Muketha [108] applied Weyuker's properties to validate their respective metrics.

2.7.1.2 Kitchenham properties

Kitchenham et al. [69] developed a theory to validate metrics by confirming the category of software metrics which are attribute, unit, instrument, and protocol validity. The properties are:

Property 1: A measurable attribute must allow distinct entities to be differentiated from one another.

Property 2: A valid measure must conform to the original meaning.

Property 3: Every part of an attribute contributing to a valid measure should be equivalent.

Property 4: Distinct entities can have the same attribute value which is limited by a measurement error.

Property 5: Any metric must rely on a model of the relationship between certain attributes.

Property 6: A model dimensionality must be consistent.

Property 7: The metric must not display unexpected discontinuities.

Property 8: The metric must apply the units and scale types accurately.

Although the framework is based on the measurement theory, it does not relate to a specific measurement attribute.

2.7.1.3 Briands' Framework

Briand et al [39] suggested a property-based approach for software measurement validation by evaluating the size, length, complexity, cohesion, and coupling. According to Briand's framework, a system S is formally defined as a pair $\langle E, R \rangle$, where E represents the set of elements of S , and R is the relationship between elements. In Briand's framework, size is always equal to or greater than zero' and is characterized by the following properties;

Property 1: The size of the system is greater or equal to zero.

Property 2: The size of the system is null if there are no elements.

Property 3: The system size results from summing the sizes of the two modules.

Property 4: This property explains the way to calculate the size of a system.

Property 5: This property means that extending the number of elements to a system does not decrease its size.

Property Size 6: This property concerns adding modules together.

Briand et al. [39] proposed length properties as follows:

Property 1: Nonnegativity. The length of a diagram $X = \langle E, R \rangle$ is greater or equal to zero i.e. $\text{Length}(X) \geq 0$.

Property 2: Null value. The length of a diagram $X = \langle E, R \rangle$ is null if X is empty i.e. if X has no elements in it.

Property 3: Nonincreasing monotonicity for connected components. Let X be a diagram and m be a module of X . This property asserts that adding relationships between elements of m does not increase the length of X .

Property 4: Nondecreasing monotonicity for non-connected components. Let X be a diagram and m_1 and m_2 be two modules of X . This property suggests that totaling the relationships from elements of m_1 to elements of m_2 does not reduce the length of X .

Property 5: Disjoint modules. The length of a diagram $X = \langle E, R \rangle$ composed of two disjoint modules m_1, m_2 is equal to the maximum of the lengths of m_1 , and m_2 i.e. $(X = m_1 \cup m_2 \text{ and } m_1 \cap m_2 = \emptyset \text{ and } E = E_{m_1} \cup E_{m_2}) \Rightarrow \text{Length}(X) = \max \{ \text{Length}(m_1), \text{Length}(m_2) \}$.

Briand et.al [39] complexity are as follows:

Property 1: A systems complexity is above or equal to zero.

Property 2: The complexity is zero if a system is null.

Property 3: The complexity of the system does not depend on the method representing associations between elements.

Property 4: The complexity of the system is due to summation of the complexities of elements that compose it.

Property 5: The complexity of the system is as a result of adding two disjoint elements that compose it.

Briand et.al [39] cohesion properties are:

Property 1: The cohesion of a system is above or equal to zero.

Property 2: The cohesion is null if a system has no elements.

Property 3: The cohesion of a system is not decreased by the relationship between elements.

Property 4: Encapsulation of two elements does not reduce the cohesion of a system.

Briand et.al [39] coupling properties are as follows:

Property 1: A systems coupling is above or equal to zero.

Property 2: The coupling of a system is null if it has no elements.

Property 3: The coupling of a system increases due to adding intra-components associations.

Property 4: Combining two components does not increase the coupling.

Property 5: The coupling of a module as a result of combining two unrelated modules is equivalent to the sum of the couplings of the two original modules | coupling of the original modular system.

Although the framework is reasonable from measurement theory perspective, Briand's properties reveals that they lay too much emphasis on code and design phases and less on the run time environment, meaning they largely overlook the unique features of behavioral diagrams.

2.7.1.4 Kaner's Framework

Kaner's framework [35] provides a set of criteria for evaluating software metrics by analyzing factors such as the metric's scope, the property being assessed, and the type of scale applied to the attribute. It helps determine how practical a metric is by addressing ten key evaluation questions:

i. What is the goal of the metric?

The purpose of using the metric should be clearly described to ensure its relevance and validity.

ii. What area does this metric cover?

The specific context or boundary where the metric is applied must be identified.

iii. Which property is being measured?

The characteristic or attribute targeted by the metric must be clearly defined.

iv. What is the natural scale for the attribute?

The attribute must be measurable on a meaningful and appropriate scale.

v. How much does the attribute naturally vary?

Possible sources and ranges of variation in the attribute should be explained.

vi. How is the metric defined?

The metric should have a clear and unambiguous definition.

vii. What scale is used for the metric?

The chosen scale for the metric should suit the type of data being represented.

viii. How consistent are the readings from this tool?

Any potential measurement errors or inconsistencies should be noted.

ix. How is the attribute related to the metric value?

The connection between the property being measured and the metric result should be clearly shown.

x. What are the expected or possible side effects of applying this metric?

Unintended consequences or drawbacks of using the metric should be acknowledged.

Although this framework has been useful in assessing the practical value of software metrics, it does not fully support the evaluation of UML behavioral diagrams. The questions are general and do not address specific aspects such as detailed diagram elements or the structure of control flows, which are essential in analyzing the complexity of UML behavior models. Metric validation should be context-dependent by considering the nature of the models and their intended use [69].

2.7.2 Empirical Validation

Empirical validation is conducted to determine whether newly proposed metrics are significant in the software industry [132], [135], [136].

2.7.2.1 Experiments

Experimentation is a scientific research method designed to test a hypothesis through controlled conditions. It involves manipulating variables, precise measurement, and establishing cause-and-effect relationships [88]. There are various types of experiments, including laboratory experiments, field experiments, and natural experiments [88]. Researchers can adopt different experimental designs, such as between-subject and within-subject designs. A between-subject design involves dividing subjects into different groups and each subjected to different conditions. For instance, Ndia [74] employed a between-subject design in metrics validation. On the other hand, a within-subject design involves exposing all the subjects to similar conditions. Muketha [108] utilized this approach to validate the proposed metrics.

2.7.2.2 Case Studies

A case study is an in-depth investigation of a particular subject, process, or system. This method integrates multiple data collection techniques, including observations, interviews, and document analysis, to obtain comprehensive insights [88], [89]. However, a significant limitation of case studies is their susceptibility to researcher bias, which may influence the findings [88].

2.7.2.3 Surveys

A survey is a research methodology that involves collecting and analyzing opinions, perceptions, and experiences from a sample of respondents [88], [90]. Surveys involve questionnaire design, sampling technique, and data analysis. Surveys are utilized in software engineering to examine user satisfaction, software usability and industry practices.

2.8 Conceptual Framework

A conceptual framework brings together important ideas and terms that help explain the research problem. It shows how these ideas are connected, often through the use of independent and dependent variables in a research context [133], [134]. These concepts are usually drawn from both theoretical foundations and past empirical studies [74], [91].

The complexity metrics of UML behavioral diagrams serve as the independent variables. These include metrics such as weighted number of state activities (WNSA), adjusted weighted number of states (AWNS), total sequence complexity (TSC), and total interaction complexity (TIC). These complexity metrics influence the level of software analyzability, which is treated as the dependent variable. Analyzability in this study is evaluated through two main indicators: the ease of understanding a diagram and the time taken to analyze it.

In addition, two moderating variables, namely, education level and UML modeling experience are considered, as they may influence the relationship between complexity metrics and analyzability. Figure 2.7 represents these relationships

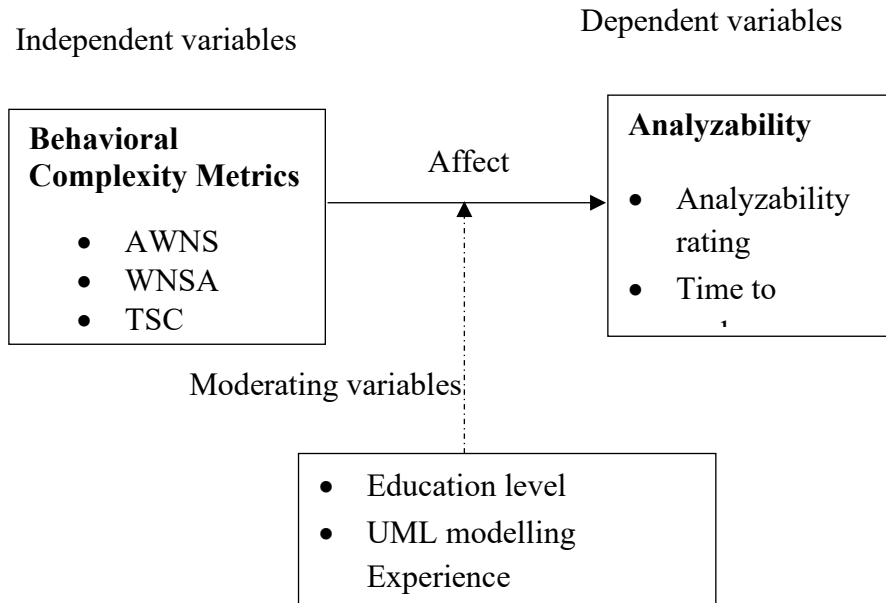


Figure 2.7: Conceptual Framework

2.9 Summary

This section recaps and highlights the gaps identified from the reviewed literature on various complexity measurement models, metrics, validation frameworks, and tools to understand their role in assessing software complexity.

The analysis focused on key theoretical models, including the Entity-Attribute-Metric (EAM) model, existing UML metrics, metrics tool, Weyuker's axioms, the ISO 25010 quality model, Briand's framework, and Kaner's framework. These models provided a background for defining, validating, and applying UML behavioral diagram metrics.

In the analysis of existing literature, several gaps were identified: first, the existing metrics measurement frameworks were found to be inadequate for defining analyzeability metrics as they do not account for all aspects of run-time system behavior; second, existing complexity metrics were found to be limited, as they fail to capture all features of internal system behavior, leading to an incomplete assessment of software complexity. This limitation makes it difficult to evaluate how easily a system can be analyzed.

Further, the chapter revealed that existing theoretical validation frameworks are not well-suited for evaluating UML behavioral diagram metrics. This makes it challenging

to confirm whether these metrics are both mathematically sound and practically useful. Also, it was observed that although various metrics tools are available, they are often inconsistent in their measurements. These tools calculate only a limited number of selected metrics and tend to produce varying results over time, making it challenging to identify a standard approach for measuring software complexity.

These findings emphasize the need for enhanced complexity measurement models, improved metrics validation frameworks, and standardized metrics tools to ensure that software complexity, particularly in UML behavioral diagrams, can be measured more accurately. The findings from this chapter formed the basis for the methodological approach in the study.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

This chapter presents the techniques used in the study namely, research philosophy, design, methods, data collection, and data analysis tools. The research design, outlines the steps followed to achieve the study's objectives. The research methods section describes the selected methodology and provides a rationale for its selection. In addition, the chapter describes the research strategy, target population, sampling techniques, data collection, and analysis tools, along with justifications for each choice. Ethical considerations in the study are also addressed.

3.2 Research Philosophy

This study applied pragmatism due to its flexibility in combining qualitative and quantitative methods [68], [69]. This study adopted pragmatism as its guiding philosophy because it allows for the use of both qualitative and quantitative methods, depending on what best suits the research purpose [70].

Ontologically, the study assumes that reality is both constructed and observable, meaning some aspects of complexity can be interpreted through human perception, while others can be objectively measured. Epistemologically, knowledge is drawn from practical inquiry using appropriate methods for each objective.

Positivism was unsuitable due to its emphasis on objectivity alone, limiting its use in exploratory phases. Interpretivism was also limited, as it lacks support for empirical validation.

In the early stages of the study, qualitative methods were useful for exploring and understanding complex aspects of UML behavioral diagrams. Later, as the study focused on evaluating and validating proposed metrics, quantitative methods were used to analyze data and confirm findings. Pragmatism was chosen because it supports this flexible approach, aiding the researcher in selecting methods that align with each research objective.

3.3 Research Design

This study employed both qualitative and quantitative research designs to address the research problem effectively [71], [72]. A qualitative exploratory design was used in the initial phase to identify perspective-based characteristics and guide the theoretical formulation of complexity metrics. An expert opinion survey was applied to collect insights from respondents on critical features, technical concerns, and diagram interpretation. A semi-structured questionnaire was used in the survey to enable the collection of both qualitative and quantitative. The validation of metrics using theoretical frameworks was also qualitative.

The quantitative phase adopted a true experimental design to investigate the usefulness of the proposed metrics in predicting the analyzability of UML behavioral diagrams. This design was appropriate for testing hypotheses about cause-and-effect relationships between diagram features and perceived complexity [74].

The validation process incorporated both subjective and objective approaches to ensure a comprehensive assessment of the proposed metrics. In the subjective phase, subjects were presented with a series of behavioral diagrams and asked to examine each one within a specified time frame. After reviewing each diagram, they rated its analyzability using a Likert scale, reflecting their judgment on how easy or difficult it was to analyze. In the objective phase, subjects were required to record the time they took to analyze each diagram. Specifically, they noted both the starting and ending times, which allowed for the calculation of the actual duration spent on analysis. This combination of perceptual ratings and time-based measurements provided both qualitative and quantitative insights into the effectiveness of the metrics.

3.4 Research Process

A research process displays the steps followed to ensure that the research problem was efficiently handled. This study involved five main steps: identification of perspective-based information that can be used as metrics for UML behavioral diagrams; defining a suite of metrics; developing a metrics validation framework; developing and applying a static analyzer tool; and executing a controlled lab experiment to validate the metrics. Figure 3:1 shows the research process activities.

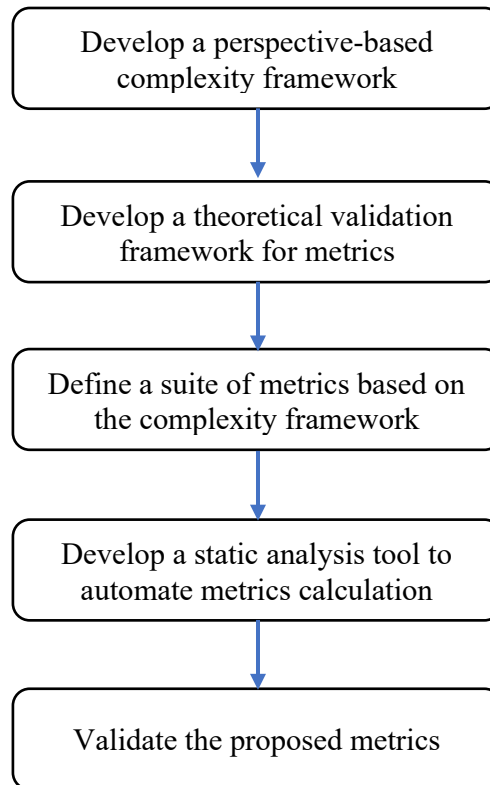


Figure 3.1: Research Process

3.4.1 Development of a Perspective-Based Complexity Framework

This study introduces a perspective-based complexity framework designed to outline the key dimensions that influence the complexity of UML behavioral diagrams. The framework serves as a foundation for identifying the various perspectives through which complexity can be understood and assessed within these diagrams.

The perspective-based complexity framework proposed in this study focuses on three main dimensions to aid in understanding the complexity of UML behavioral diagrams namely, the element perspective, the control-flow perspective, and the interaction perspective. These perspectives capture different aspects of what makes these diagrams complex and were used to guide the development of specific attributes and metrics.

The element perspective is concerned with the core components that make up each type of behavioral diagram. For example, states are essential elements in statechart diagrams, action states are key parts of activity diagrams, and messages are

components in sequence diagrams. These elements aid in measuring structural complexity and are associated with metrics such as the Weighted Number of States and Activities (WNSA) for statechart diagrams, and adjusted message metrics like Adjusted Weighted Message into Lifeline (AWMIL) and Adjusted Weighted Message out of Lifeline (AWMOL) for sequence diagrams, along with the Total Sequence Complexity (TSC).

The control-flow perspective, on the other hand, looks at the order and structure of how behaviors or actions occur in the system. This includes typical software flow structures such as sequential steps, decision, repeated behaviors, and parallel actions that happen at the same time. These types of flow are useful for understanding how the system executes behavior and are captured using the Aggregate Control Flow Complexity (ACFC) metric.

The interaction perspective is based on how elements or components in the diagram communicate with each other. This includes both incoming and outgoing flows of messages, links, edges, or transitions between objects or diagram parts. Incoming interactions refer to the number of connections directed toward an object, while outgoing interactions refer to those that move away from it. Metrics such as Incoming Interaction Complexity (IIC), Outgoing Interaction Complexity (OIC), and Total Sequence Complexity (TSC) were used to capture these interactions.

The framework was validated to assess its relevance and comprehensiveness in identifying the complete set of complexity perspectives of behavioral diagrams. Validation was achieved through an expert opinion survey. A copy of the questionnaire used in this study is shown in Appendix 1.

3.4.2 Definition of Metrics Based on the Perspective-Based Complexity Framework

The perspective based complexity framework was utilized to define a suite of measures for UML behavioral metrics under each category of perspective.

3.4.2.1 Element Complexity Metrics

Element perspective aids the designers to understand the extent to which the building blocks contribute to the complexity of UML behavioral diagram. The weighted

number of states and activities metric measures the complexity due to the presence of state element in a statechart diagram, while the adjusted weighted message into lifeline, weighted message out of lifeline and total sequence complexity measures the complexity contributed by the message component. The adjusted weighted number of state metric measures the complexity due to the presence of an action state in an activity diagram.

3.4.2.2 Control-flow Complexity Metric

Control flow metric measures complexity due to behavior execution. The motivation in measuring this attribute is to find out what kind of control flow exists in a diagram when behavior is being executed. The aggregate control-flow metrics aid the software designers to understand the extent to which the presence of a sequence, decision, parallel, and control-flow during the execution of behavior contributes to the complexity of a diagram.

3.4.2.3 Interaction Complexity Metrics

A diagram's interaction occurs when the elements of a diagram communicate with each other. The incoming interaction complexity measures the complexity due to the incoming interaction while the outgoing interaction measures the complexity due to outgoing interaction. The total sequence complexity is a composite metric derived from incoming interaction complexity and outgoing interaction complexity.

3.4.3 Theoretical Validation of the Metrics

To determine if the metrics were mathematically sound, a theoretical validation was conducted using Weyuker's properties [73]. Weyuker's properties are widely used for validating complexity metrics. Additionally, the metrics were analyzed using a perspective-based metrics validation framework (PBMVF) that was developed.

The research focused on the development of a perspective-based metrics validation framework (PBMVF) that is relevant to behavioral diagrams. Two existing software metrics validation frameworks, namely Kaner's and Briand's were modified to establish measurement attributes. Three measurement attributes, namely element, control flow, and interaction, were identified based on the perspective complexity of these diagrams.

The element attribute analyzes the building blocks of a behavioral diagram, the control flow analyzes the presence of sequence, decision, loop, and parallel in a diagram while the interaction attribute analyzes how objects communicate by sending messages to each other. The second step was to define properties for each of the measurement attributes. The properties of the element are the type of element, the extent of the element, and weighted element complexity. The control flow properties are the type of control flow, the extent of the control flow, depth of the edge, and weighted control flow complexity. The interaction properties are the type of interaction, total of incoming and outgoing messages, and weighted interaction complexity. Additionally, the perspective-based metrics validation framework was validated to test the applicability of the proposed framework using behavioral diagram metrics.

3.4.4 Development of Behavioral Metrics Complexity Tool

Development of the behavioral metrics complexity tool was necessary for the new metrics to be recognized by the software industry. A static analyzer tool was created to detect XML files, collect the required tokens and calculate various metrics. The BCMT tool was implemented using Visual Basics 2019. The tool was effective and efficient in the computation of metrics.

To ensure the tool's correctness, validation was carried out through manual computation. Selected UML behavioral diagrams were analyzed both manually and using the BMCT tool. The results from both methods were compared to verify accuracy. The tool's outputs consistently matched the manually derived metric values, confirming its reliability in processing and computing behavioral diagram metrics.

3.5 Population

A population includes all individuals or groups whose characteristics are relevant for drawing conclusions from the study. In this research, the target population consisted of UML modeling professionals working in the Kenyan software industry. These experts were selected to help validate the perspective-based complexity framework by providing informed feedback based on their experience.

To validate the metrics, the target population was 238 students enrolled in their final year of study at the School of Computing and Information Technology (SCIT) at Murang'a University of Technology. Students can be used as subjects for software

engineering experiments since the results obtained from them are similar to those of industry experts [75].

3.6 Sampling Technique and Sample Size

Sampling process is the method used to choose a smaller group from a larger population, making it possible to draw conclusions or predictions about the entire population. The researcher decides on a sample size by considering whether the population is homogeneous or heterogeneous [75] [76].

In this study, snowballing sampling was used to obtain a sample for validating the perspective-based complexity framework (PBCF). This method is advantageous when the study involves subjects who are hard to reach, and provides depth for uncovering rich and detailed insights [77] [78]. To initiate the snowball sampling process for expert validation of the Perspective-Based Complexity Framework (PBCF), the first expert was identified through purposive selection. This individual was chosen based on their recognized experience in software modeling. A total of 11 experts were involved in the validation of the PBCF.

In addition, subjects for the three experiments were chosen using systematic random sampling, which ensured that the selection was both structured and unbiased. Systematic random sampling is easy and allows uniform presentation of the population [111]. In this research, subjects who had been trained on modelling UML behavioral diagrams volunteered to participate in the experiment for validation of behavioral metrics. A total of 58 participants took part in the experiments, contributing to both the subjective evaluations and the objective measurements. In experiment one, 19 subjects participated, 19 subjects for experiment two, and 20 subjects for experiment three. This number of subjects is acceptable as found in other researchers' work [74].

3.7 Pilot Study

The expert survey questionnaire was initially tested with five experts experienced in UML modeling. This phase was conducted before the main study to verify the instrument's validity and reliability, and to ensure that each item accurately measured the intended variables. Ten subjects, all trained in UML modeling, participated in the pilot study, and none of these subjects were included during the final experimental stage.

3.8 Data Collection Instruments

The study used suitable tools to collect the required data. The study utilized a semi-structured questionnaire as the primary tool for gathering expert opinions. Expert opinion surveys have been used in software engineering to validate new metrics and frameworks [79] [80]. This questionnaire featured closed-ended questions formatted on a five-point Likert scale to ensure consistent responses (see Appendix 1).

A static analyzer tool (BCMT) was utilized to collect the metrics from the UML behavioral diagrams. Additionally, to validate the new metrics, a Likert scale was adopted to rate the analyzability of behavioral diagrams (see Appendix 2).

3.8.1 Validity of the Research Instruments

The validity and reliability of the instruments is crucial in determining the credibility of the research findings. Therefore, efforts were made to ensure that the data obtained was credible.

3.8.1.1 Content Validity

To make sure the questionnaire covered all the necessary areas, it was scrutinized by the supervisors and experts in the field of UML modeling. They assessed each question to find out if it matched the goals of the study. Their feedback aided improve the quality of the questionnaire.

3.8.1.2 Face Validity

This type of validity is about how the questionnaire looks. The supervisors and experts who were familiar with the area reviewed the questionnaire to see if the questions were clear, relevant, and easy to understand. Their feedback assisted in making the questionnaire clear.

3.8.1.3 Internal Validity

This means how well the study avoids errors that could affect the results. In this study, we included both subjective questions (based on personal opinions) and objective questions (based on facts). This aided to eliminate personal bias.

3.8.1.4 External Validity

This is about whether the results of the study can apply to other situations. The experts who answered the questionnaire came from different areas of software engineering. Because of this, the results are more likely to be useful in other similar settings.

3.8.2 Reliability of the Research Instruments

Reliability can be defined as the degree to which an instrument produces consistent results over time [81]. To ensure the reliability of the instrument developed, pretesting was executed. The pretest was carried out using Cronbach's alpha, a method commonly applied to assess the reliability of measurement instruments [81]. The acceptable Cronbach's alpha coefficient should not be below 0.07. The reliability test shown in Table 3.1 indicates that the subjective part obtained a Cronbach alpha of 0.776 while the objective part obtained 0.792. Both values exceeded the recommended threshold of 0.70 [82] [83], confirming the reliability of the data collection instrument.

Table 3.1: Reliability of the Questionnaire

Scale	Cronbach's Alpha
Subjective questionnaire part	0.776
Objective questionnaire part	0.792

3.9 Experimental Materials

The experimental phase of the study was conducted to validate the proposed complexity metrics by examining their ability to reflect the analyzability of UML behavioral diagrams. The materials used included the Behavioral Complexity Metrics Tool (BCMT) and a collection of XML files for UML diagrams, specifically sequence, statechart, and activity diagrams.

The diagrams were initially created using PlantUML, a tool for generating UML models. After modeling, the diagrams were exported in XML format, which served as input files for the experiment. The BCMT tool, developed in Visual Basic 2019, was installed and run on the researcher's computer to extract metric values from each XML file. During the experiment, subjects were presented with a set of UML behavioral diagrams and asked to analyze them. Each subject was required to record the time taken to analyze each diagram (objective measurement) and rate its analyzability using

a Likert scale (subjective measurement). The corresponding metric values for each diagram were computed using the BCMT tool.

The goal of this experiment was to determine whether there is a relationship between the computed metric values and the subjects' perceived effort in analyzing the diagrams. The results from the experiment were then used to validate the metrics, providing evidence of their relevance and predictive power in assessing diagram analyzability

3.10 Data Analysis

Data analysis was both qualitative and quantitative in nature. The statistical package for Social Sciences (SPSS) version 28 was used to analyze quantitative data. The quantitative data gathered from the expert opinion survey was analyzed through descriptive statistics, including the use of frequency, mean, and percentage.

Qualitative analysis focused on the theoretical validation of the proposed metrics. This involved evaluating each metric against Weyuker's nine properties, which test characteristics such as non-coarseness, monotonicity, and interaction sensitivity. In addition, metrics were assessed using the Perspective-Based Metrics Validation Framework (PBMVF) developed in the study. The PBMVF guided the structured analysis of metric relevance, alignment with complexity perspectives.

Data from the experiment was quantitative in nature. Before analysis, a Kolmogorov-Smirnov test was conducted on the data collected to establish whether it was normal or non-normal. Data was analyzed using descriptive statistics and inferential statistics. Descriptive statistics such as measures of central tendency (mean, max and min) and measure of dispersion (range and standard deviation) were used to explore the data and also for data visualization. Finally, inferential statistics (correlation and regression) was used to aid the researcher make inferences on the data.

3.11 Ethical Issues

Ethical considerations by the researcher first involved obtaining an introductory letter from the Board of Postgraduate Studies at Murang'a University of Technology (refer to Appendix 3). Approval was then sought and granted by the National Commission for Science, Technology, and Innovation (NACOSTI), as shown in Appendices 5 and

6. The researcher conducted training sessions on UML behavioral diagram modeling for the participants before involving them in the experiments. Participation in both the training and experimental activities was voluntary. The researcher also assured all participants that any data collected would be kept confidential and used solely for academic purposes.

3.12 Summary

This chapter outlines the research methodology, detailing the overall approach used in the study. It covers the guiding research philosophy, design, strategy, sampling procedures, data collection tools, analysis methods, and ethical practices. A five-phase process was followed to meet the study's objectives, anchored in a pragmatist research philosophy. The research adopted an explanatory approach, using both surveys and experimental methods as core strategies. For the first and fifth objectives, purposive sampling was applied. The experiments made use of the BCMT metrics tool and a collection of XML files. Data analysis involved both descriptive and inferential statistical techniques. All ethical requirements were observed, and relevant permissions were obtained before conducting the expert survey and experimental phases.

CHAPTER FOUR

A PERSPECTIVE-BASED COMPLEXITY FRAMEWORK (PBCF) FOR UML BEHAVIORAL DIAGRAMS

4.1 Introduction

This chapter introduces a Perspective-Based Complexity Framework (PBCF) intended to classify the attributes used in measuring the complexity of UML behavioral diagrams. The development of this framework addresses the first research objective, which focuses on identifying relevant perspective-based characteristics for defining complexity framework for these diagrams. In addition, this section outlines the findings derived from an expert opinion survey conducted to validate the proposed framework.

4.2 Overview

The Perspective-Based Complexity Framework (PBCF) is developed to aid software modelers to analyze UML behavioral diagrams. Behavioral diagrams, such as statechart, activity, and sequence diagrams, are commonly used in software engineering to depict the runtime behavior of a system. However, as systems grow in size and functionality, these diagrams can become complex to analyze.

The PBCF framework addresses this gap by examining the diagrams from different aspects, known as perspectives. Each perspective highlights a different source of complexity, such as the number of elements in the diagram, how control flows are executed in the system, or how different objects in the system interact with each other.

The framework classifies these perspectives making it easier to examine complex diagrams. This structured approach aids developers, designers, and analysts to better analyze, measure, and the quality of UML behavioral diagrams.

4.3 PBCF Framework Requirements

The requirements for the PBCF were derived from the findings of the literature review, which identified limitations in existing complexity frameworks, especially their narrow focus on single diagrams and structural metrics.

Based on these findings, the PBCF is expected to meet the following criteria:

- i. The framework should identify the different perspectives that contribute to complexity of UML behavioral diagrams.
- ii. The framework should sub-categorize the perspectives to analyze the complexity of different diagrams.
- iii. The framework should be scalable to apply across all the behavioral diagrams
- iv. The framework should support integration with existing UML tools to make it practical in real- world of software engineering.

4.4 Architectural Design

The various parts of the perspective-based complexity framework for UML behavioral diagrams which are the element perspective, control flow perspective and interaction perspective are described into details.

The framework identifies different measurable perspectives that behavioral diagrams display during execution of behavior. In addition, the framework classifies the identified perspectives into different categories. For example, the element perspective includes observable attributes such as action states, states, and messages. The control flow perspective is broken down into types like sequential, decision, repetitive, and parallel control flows. Similarly, the interaction perspective is classified into incoming interactions and outgoing interactions. Figure 4.1 illustrates the details of the PBCF framework.

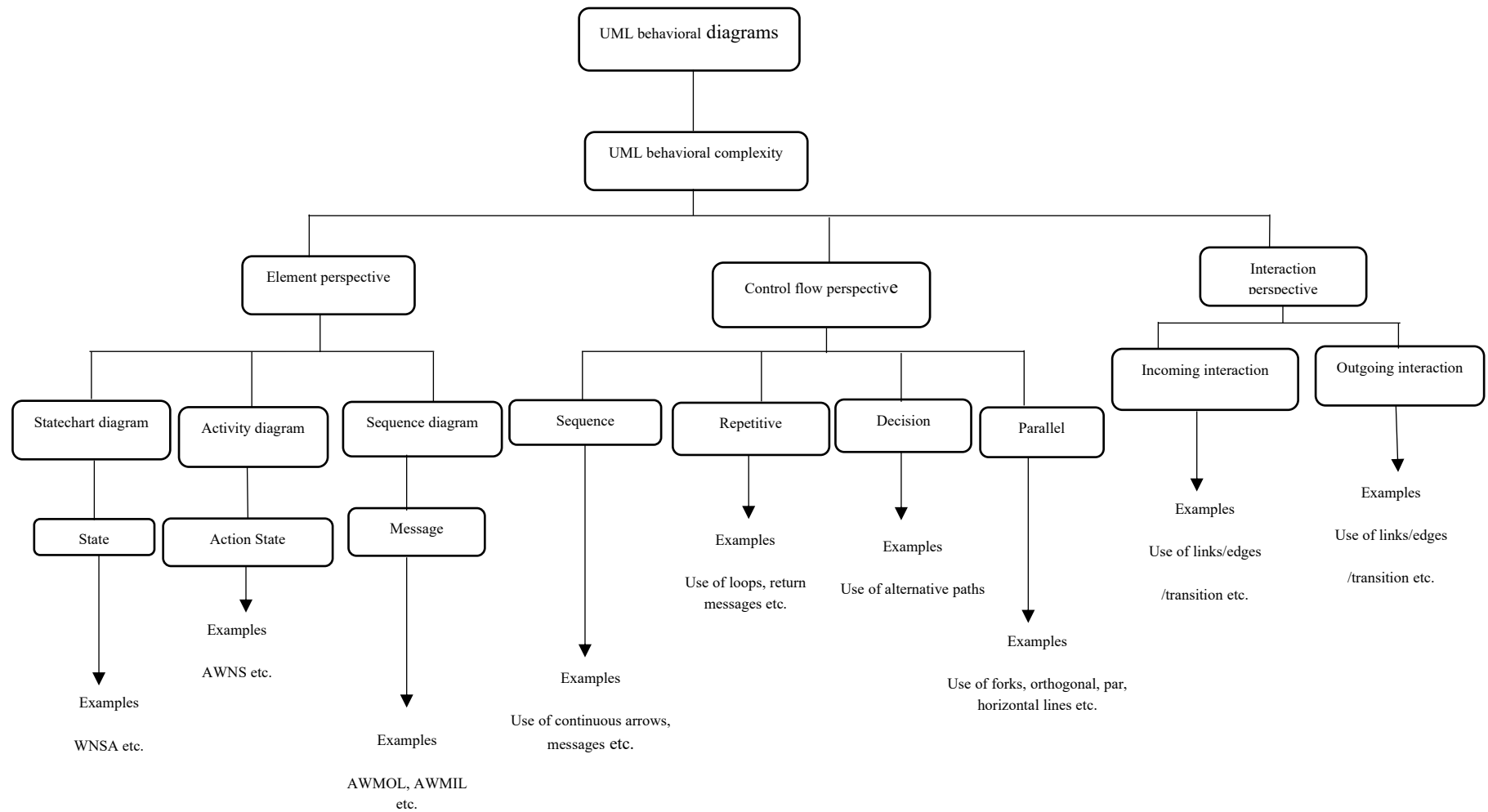


Figure 4.1: Perspective-based Complexity Framework

4.4.1 Element Perspective

The element perspective focuses on the fundamental structural components that make up a behavioral diagram. Each type of UML behavioral diagram is composed of distinct elements, and an increase in the number of these components typically leads to greater diagram complexity. For instance, in a statechart diagram, the primary structural elements include states, events, and transitions. Figure 4.2 provides a visual representation of the elements within a statechart diagram.

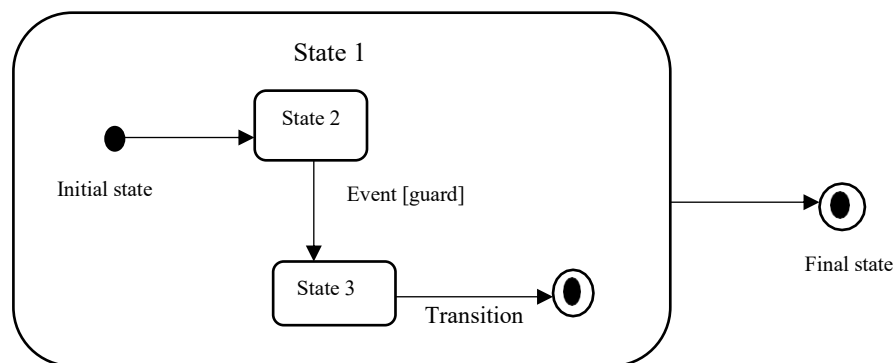


Figure 4.2: Statechart Elements

A sequence diagram consists of multiple objects that communicate by exchanging messages. These objects are depicted using lifelines, while the messages exchanged between them are illustrated using arrows. The messages indicate the interactions or relationships among the objects. Figure 4.3 illustrates a sequence diagram structure.

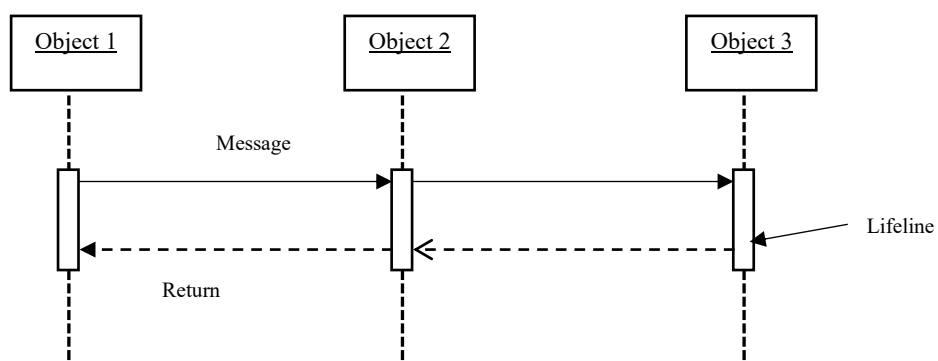


Figure 4.3: Sequence Diagram Structure

An activity diagram serves as a flowchart that captures the progression from one activity to the next. It comprises elements such as action states, edges, and both initial and final states. The action state reflects the specific behavior executed by an object,

while an activity edge indicates the directional link connecting two action states. The components that make up an activity diagram are illustrated in Figure 4.4.

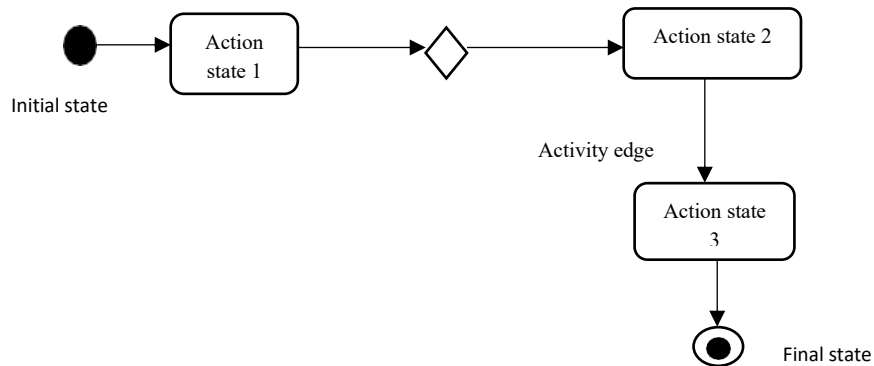


Figure 4.4: Activity Diagram Structure

4.4.2 Control Flow Perspective

In software systems, control flow refers to the sequence in which operations or instructions are executed. In UML behavioral diagrams, the control flow perspective represents how behavior transitions occur between different objects over time. This perspective is derived from traditional software control structures and helps to illustrate how behavioral processes unfold within a system.

Several distinct forms of control flow can be observed in behavioral modeling. Sequential control flow describes the linear execution of tasks, where one operation follows another in a specific order. Decision control flow, on the other hand, captures scenarios in which the system must choose between alternative execution paths based on defined conditions or triggers. Repetitive control flow focuses on behaviors that recur within the system, analyzing how certain actions are repeated a given number of times. Lastly, parallel control flow represents situations where multiple activities or operations occur concurrently, reflecting simultaneous execution within the system. Figure 4.5 visually presents how each of these flow types is depicted within behavioral diagrams.

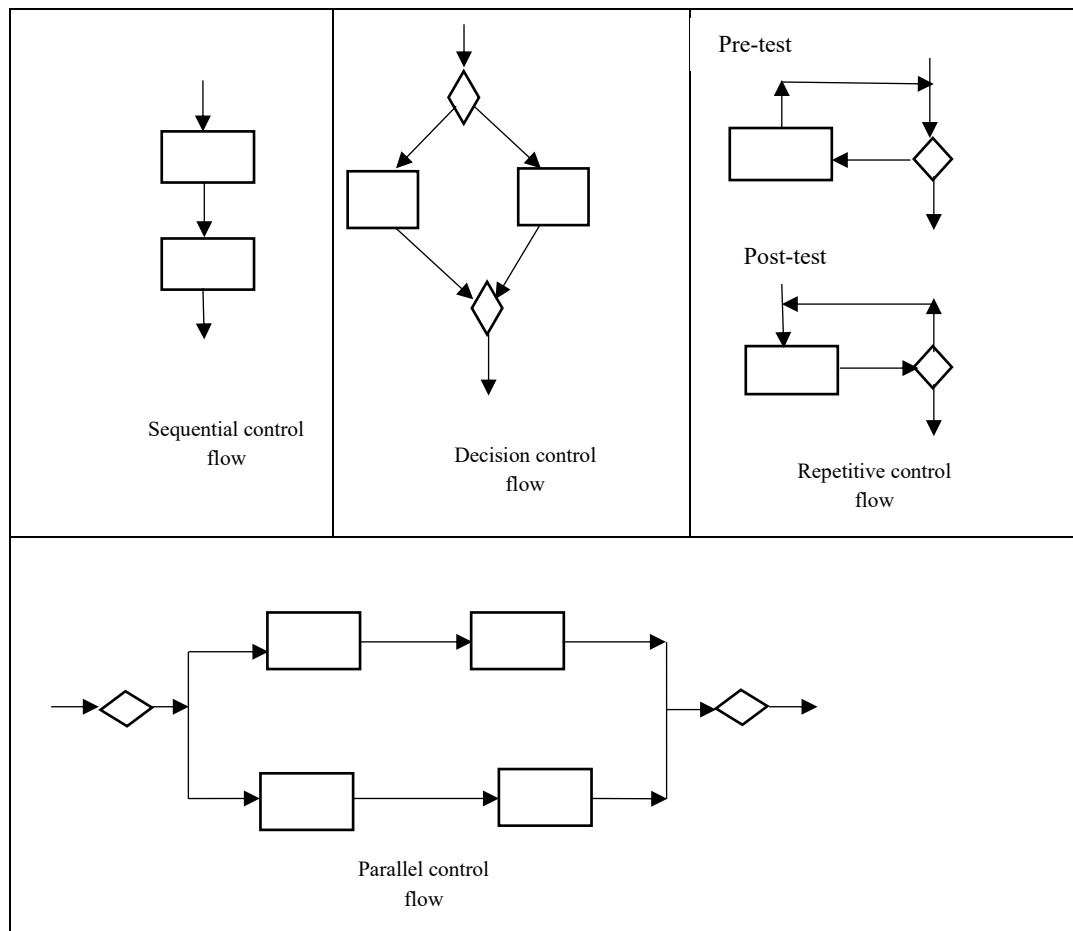


Figure 4.5: Control Flows in Behavioral Diagrams

4.4.3 Interaction Perspective

In UML behavioral diagrams, interaction refers to the exchange of messages or signals between various diagram elements, including states, action states, and lifelines. These interactions are depicted using different modeling notations, such as transitions, edges, messages, or links, depending on the type of behavioral diagram involved. These constructs serve to demonstrate how different parts of the system communicate and influence one another during execution.

An element that is involved in a higher number of transitions, messages, links, or edges is interpreted as having greater interaction intensity. A higher degree of interaction is often associated with increased diagram complexity. To better analyze these relationships, the interaction perspective is categorized into two subtypes namely incoming interaction, which accounts for all transitions or connections directed toward

an element, and outgoing interaction, which captures the transitions or links initiated from an element to others.

To illustrate, consider the statechart diagram in Figure 4.6. The state labeled initiated has three outgoing transitions, while the preserved state has two. In contrast, the confirmed, in progress, and done states each have only one outgoing transition. This suggests that the initiated state has the highest level of interaction among the states, thereby contributing more to the behavioral complexity of the diagram.

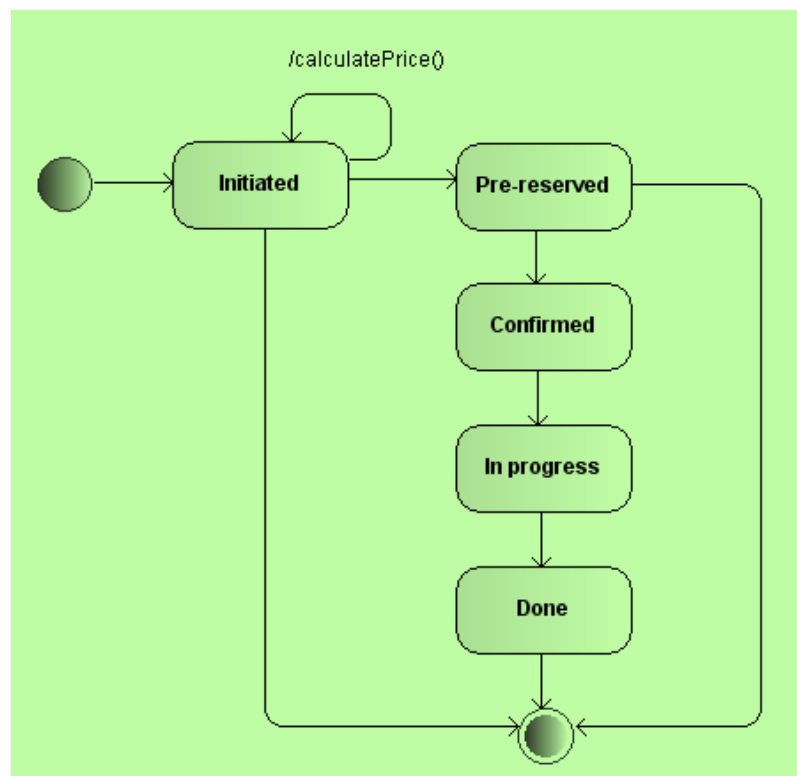


Figure 4.6: Reservation Statechart Diagram

4.5 Expert Opinion Validation Survey

This section outlines the process and purpose of a structured expert survey conducted to evaluate the validity of the proposed Perspective-Based Complexity Framework (PBCF).

4.5.1 Goal of the Study

The primary aim of the study was to evaluate the significance, comprehensiveness, and practical applicability of the proposed Perspective-Based Complexity Framework (PBCF) through feedback gathered from UML modeling experts. Specifically, the

study sought to determine whether the framework successfully identifies the key factors that influence the complexity of behavioral diagrams and whether its categorization aligns with practices commonly used in real-world modeling scenarios. In addition, the experts were asked to consider the framework's suitability for integration into software modeling.

4.5.2 Context Definition

UML experts were invited to take part in the survey, whereby a total of 11 experts participated in the survey, and these individuals were identified through a snowball sampling. The technique was employed to reach individuals with specialized knowledge by leveraging referrals from initial experts.

The researcher recruited 11 experts, considering that this number is sufficient given the inherent difficulty in locating UML experts. The relatively small pool was deemed appropriate because, in this specialized field, additional participants were unlikely to provide significantly new insights. Consequently, the researcher concluded that this sample size is acceptable, as supported by prior studies [74] [84]. This approach ensured that the survey captured a comprehensive range of expert opinions while remaining both practical and reflective of the available expert population.

4.5.3 Survey Operation

The experts were provided with a detailed Perspective-Based Complexity Framework and a semi-structured questionnaire designed to gather their feedback. The framework itself was designed to delineate the various behavioral perspective complexities. Additionally, the survey questionnaire was carefully constructed to capture detailed feedback and expert opinions, ensuring that experts had all the necessary resources to fully understand and engage with the material presented.

4.5.4 Reliability of the Research Instrument

A reliability assessment was carried out on the questionnaire utilized in the validation process to verify the dependability and consistency of the data obtained through the study. The analysis specifically aimed to evaluate the framework's inclusiveness and relevance, ensuring that the instrument would produce stable and consistent results regardless of the group of experts applying it. This step was essential to confirm the tool's effectiveness in accurately capturing the intended perspectives.

Cronbach's alpha was used as the statistical measure for reliability. The results demonstrated that the set of items linked to the element perspective achieved a Cronbach's alpha of 0.794, indicating a strong correlation among the items and high reliability for this construct. In the case of the control flow perspective, the coefficient was 0.714, suggesting that the items possess acceptable internal consistency. Notably, the interaction perspective recorded a perfect alpha value of 1.0. Table 4.1 illustrates these results.

All reported Cronbach's alpha values exceeding the generally accepted benchmark of 0.7 [82]. Therefore, the questionnaire is considered to be a reliable instrument. These findings confirm that the tool is capable of consistently measuring the relevant perspectives of the proposed framework, even when applied across different expert groups.

Table 4.1: Framework Inclusiveness Reliability

Perspective	Cronbach's Alpha
Element	0.794
Control flow	0.714
Interaction	1.0

4.6 Results

Once the data collection phase was completed, all the questionnaires received were carefully checked for completeness. It was observed that every respondent had duly filled out their questionnaire, and as a result, no responses were discarded. This complete data set was then accepted for further data analysis, providing a robust foundation for the study's conclusions.

4.6.1 Respondent Demographics and Characteristics

The researcher first gathered essential background information on the respondents before delving into the core results related to the framework's relevance and inclusiveness. The collected background information covered each respondent's educational qualifications, their years of professional experience in the software industry, and their proficiency in working with UML modeling techniques. Gathering such demographic and professional background data is crucial as it aids contextualize the responses and ensures that the sample comprises individuals with sufficient expertise to evaluate the framework.

4.6.2 Academic Qualifications

Respondents were requested to indicate their highest academic qualification. Among the total number of respondents, 8 experts (72.7%) held a bachelor's degree, while the remaining 3 respondents (27.3%) had obtained a master's degree. A summary of these findings is presented in Table 4.2.

In this study, a minimum of a bachelor's degree in computer science, software engineering, or a related field was considered an essential criterion for expert participation. This threshold ensured that participants had the academic foundation required to understand and evaluate the proposed metrics and modeling framework. Those with postgraduate qualifications contributed additional depth through advanced academic exposure and research experience.

Table 4.2: Level of Education

Level of Education	Frequency	Percent (%)
Bachelors Degree	8	72.7
Masters Degree	3	27.3

4.6.3 Industry Experience

The questionnaire also gathered information on the duration of each respondent's professional experience in the software industry. The responses reflected a broad range of experience levels. Specifically, 2 respondents (18.2%) indicated they had worked for 4 to 5 years, 3 respondents (27.3%) reported having 6 to 7 years of experience, and 6 respondents (54.5%) had accumulated 8 or more years in the field. A breakdown of this data is shown in Table 4.3.

Table 4.3: Experience in the Software Industry

Years Served in the Industry	Frequency	Percent (%)
4-5	2	18.2
6-7	3	27.3
8 and above	6	54.5

4.6.4 Expertise in UML Modeling

The researcher aimed to assess the respondents' foundational knowledge in UML diagram modeling. The findings showed that 1 respondent (9.1%) possessed a moderate understanding of UML modeling, while 7 respondents (63.6%) were identified as having a high level of expertise. Additionally, 3 respondents (27.3%) were classified as having extremely high proficiency in UML modeling. The details of their expertise levels are summarized in Table 4.4.

Table 4.4: Knowledge in UML Modeling

Knowledge in UML Modeling	Frequency	Percent (%)
Moderate	1	9.1
High	7	63.6
Extremely High	3	27.3

4.6.5 Relevance of the Framework

The main aim of this part of the survey was to find out how useful the proposed perspective-based framework is for analyzing UML behavioral diagrams. Respondents were asked to rate the framework using a Likert scale. The average score from their ratings was used to show how relevant they thought the framework was.

The results showed that the framework is considered very useful, with an average score of 4.55. This score is between “agree” and “strongly agree,” meaning most respondents believed the framework is practical and helpful for analyzing UML diagram complexity. Also, the standard deviation was 0.688, which is less than 1. This means there wasn’t much difference in the responses from experts. The results are shown in Table 4.5.

Table 4.5: Relevance of the Framework

	Relevance of the Framework
Mean	4.55
Standard deviation	0.688

Additionally, Table 4.6 illustrates that the majority (63.6%) of respondents strongly agreed that the framework is relevant. An additional 27.3% agreed with its relevance,

while only a small fraction (9.1%) neither agreed nor disagreed. This distribution strengthens the conclusion that the proposed framework is perceived as a valuable tool for analyzing the complexity inherent in UML behavioral.

Table 4.6: Relevance of the Perspective-Based Framework

Likert Scale	Frequency	Percent (%)
Strongly agree	7	63.6
Neither agree nor disagree	1	9.1
Agree	3	27.3
Strongly disagree	0	0.0
Disagree	0	0.0

4.6.6 Inclusiveness of the Framework

The survey also established the inclusiveness of various perspectives within the framework. This was done by assessing three main components namely, the element perspective, the control flow perspective, and the interaction perspective. The element perspective of the framework was designed to capture key attributes that contribute to the complexity of UML behavioral diagrams. The respondents rated several specific attributes: state, action state, and message.

State attribute was rated with an average score of 4.45, and the standard deviation was 0.522, indicating strong consensus among respondents regarding its contribution to diagram complexity. Similar to the state attribute, the action state attribute also achieved a mean of 4.45 and a standard deviation of 0.522, reflecting consistency in expert opinions. The message attribute obtained a slightly lower mean of 4.27 with a standard deviation of 0.65, suggesting that while it is also regarded as an important factor, there is marginally more variation in responses regarding this attribute. These summary of the results are presented in Table 4.7.

Table 4.7: Inclusiveness of the Element Perspective

Element Attributes	Mean	Standard Deviation
State	4.45	0.522
Action state	4.45	0.522
Message	4.27	0.65

Additionally, respondents rated the inclusiveness of key attributes under the element perspective, which includes state, action state, and message. For both the state and action state attributes, 6 respondents (54.4%) agreed that these elements play a significant role in contributing to the complexity of UML behavioral diagrams, while 5 respondents (45.5%) strongly agreed. For the message attribute, 6 respondents (54.5%) agreed on its importance, 4 respondents (36.4%) strongly agreed, and 1 respondent (9.1%) was neutral, selecting neither agree nor disagree. There were no responses in the disagree or strongly disagree categories for any of the attributes. The distribution of responses is shown in Table 4.8.

Table 4.8: Rating of the Inclusiveness of Element Perspective

	State	Action state	Message
Likert Scale	Percent (%)	Percent (%)	Percent (%)
Strongly disagree	0	0	—
Disagree	0	0	0
Neither agree nor disagree	0	0	9.1
Agree	54.4	54.4	54.5
Strongly agree	45.5	45.5	36.4

In addition to the element perspective, the control flow perspective was also examined to determine its contribution to the overall complexity of UML behavioral diagrams. This perspective was evaluated through attributes, namely sequence, decision, parallel, and repetitive flows.

The sequence control flow attribute recorded a mean score of 4.55 with a standard deviation of 0.69. The decision attribute was rated slightly higher, achieving a mean

of 4.64 and a standard deviation of 0.67. Similarly, the repetitive control flow also received a mean score of 4.64, though with a lower standard deviation of 0.50, indicating greater agreement among respondents. The parallel control flow was rated the highest, with a mean score of 4.73 and a standard deviation of 0.65. Table 4.9 shows the results.

Table 4.9: Inclusiveness of the Control-flow Perspective

Control-flow Attributes	Mean	Standard deviation
Sequence	4.55	0.69
Decision	4.64	0.67
Parallel	4.73	0.65
Repetitive	4.64	0.50

Further analysis of the control flow perspective reveals that 72.7% of respondents (7 out of 11) strongly agreed that both the sequence and decision control flow attributes significantly influence complexity, while 18.2% and 9.1% respectively selected agree and neither agree nor disagree for the sequence attribute. Similarly, the decision, control flow received 8 responses (72.7%) for strongly agree, 2 (18.2%) for agree, and 1 (9.1%) for neither agree nor disagree.

The repetitive control flow had the highest level of agreement, with 9 respondents (81.8%) selecting strongly agree and 1 respondent (9.1%) choosing agree, while 1 respondent (9.1%) remained neutral. For the parallel control flow, 7 respondents (63.6%) strongly agreed with its relevance, while 4 respondents (36.4%) selected agree. The distribution of ratings is presented in Table 4.10.

Table 4.10: Rating of the Inclusiveness of Element Perspective

Likert Scale	Sequence (%)	Decision (%)	Repetitive (%)	Repetitive (%)
Strongly disagree	0	0	0	0
Disagree	0	0	0	0
Neither agree nor disagree	9.1	9.1	9.1	0
Agree	18.2	18.2	9.1	36.4
Strongly agree	72.7	72.7	81.8	63.6

The interaction perspective was evaluated to determine its influence on the overall complexity of UML behavioral diagrams. This assessment concentrated on two specific attributes, namely, incoming interaction and outgoing interaction. The incoming interaction achieved a mean of 4.09 with a standard deviation of 0.700, while the outgoing interaction achieved a mean of 4.09 with a standard deviation of 0.700. Table 4.11 represents these results.

Table 4.11: Inclusiveness of the Interaction Perspective

Interaction Attributes	Mean	Standard deviation
Incoming interaction	4.09	0.700
Outgoing interaction	4.09	0.700

The interaction perspective findings reveal that 6 respondents (54.5%) agreed that both types of interaction significantly affect diagram complexity, while 3 respondents (27.3%) strongly agreed with this view. Additionally, 2 respondents (18.2%) remained neutral, selecting neither agree nor disagree for both attributes. Notably, none of the respondents expressed disagreement regarding the relevance of the interaction perspective in measuring behavioral complexity. The distribution of responses is presented in Table 4.12.

Table 4.12: Rating of the Inclusiveness of Interaction Perspective

Likert Scale	Incoming interaction (%)	Outgoing interaction (%)
Strongly disagree	0	0
Disagree	0	0
Neither agree nor disagree	18.2	18.2
Agree	54.5	54.5
Strongly agree	27.3	27.3

4.6.7 Adoption of the Framework

Researchers evaluated whether the perspective-based complexity framework could be practically adopted by software engineers as a useful taxonomy for analyzing the complexity of UML behavioral diagrams. To achieve this, they indicated their level of agreement regarding the framework's applicability and usefulness in real-world settings. The results revealed a mean score of 4.55 and a standard deviation of 0.69. These findings are summarized in Table 4.13.

Table 4.13. Adoption of the Perspective Based Framework

	Relevance of the Framework
Mean	4.55
Standard deviation	0.69

In addition, analysis of the response distribution concerning the adoption of the framework showed that 63.6% of the respondents strongly agreed that the proposed framework could be effectively utilized by software engineers as a practical complexity taxonomy for UML behavioral diagrams. A further 27.3% agreed with this viewpoint, while only 9.1% selected neither agree nor disagree, indicating minimal uncertainty among the experts regarding its applicability. These results are presented in Table 4.14.

Table 4.14: Rating on the Adoption of the Framework

Likert Scale	Frequency	Percent (%)
Neither agree	1	9.1
Agree	3	27.3
Strongly agree	7	63.6
Disagree	0	0
Strongly disagree	0	0

4.6.8 Qualitative Results

The questionnaire included an open-ended item under Part D (Appendix 1), which invited respondents to suggest any additional perspectives they felt had not been captured or adequately represented in the proposed framework for analyzing the complexity of UML behavioral diagrams. However, no responses were recorded for this section of the survey.

As a result, no qualitative data was available for analysis. The study therefore relied exclusively on the quantitative responses provided through the structured Likert-scale questions in assessing expert views on the relevance, inclusiveness, and adoption potential of the framework.

4.7 Discussion

Reliability analysis together with the evaluation results from the questionnaire, provides adequate evidence that the developed perspective-based complexity framework (PBCF) is both robust and applicable. The high Cronbach's alpha values of the framework indicate that the instrument used to measure the inclusiveness and relevance of the framework is highly consistent and reliable. The fact that each perspective namely, element, control flow, and interaction achieved mean scores that range from "agree" to "strongly agree" further strengthens the framework's reliability.

The internal consistency of the instrument is crucial for ensuring that the results are reliable. In this study, Cronbach's alpha values for the different perspectives all exceed the threshold of 0.7. In particular, the interaction perspective's score of 1.0 suggests that the attributes are well designed to measure this aspect. Reliability results from multiple experts assure software engineers that the findings reflect a true consensus among experienced professionals.

The demographic data collected from the respondents indicate that the sample was composed of well-educated and experienced professionals. The majority hold a bachelor's degree and a number of experts have eight or more years of experience in the software industry which is the expertise required to evaluate UML behavioral diagrams. Also, the high levels of knowledge in UML modeling among the experts further strengthen the validation process.

The survey results indicate that the perspective-based framework is highly relevant for diagram analyzability. A mean rating of 4.55, supported by a low standard deviation, reflects a strong agreement among respondents. The components of the framework namely the element, control flow, and interaction perspectives were assessed separately, and all the perspectives were deemed inclusive and reflective of the key attributes that contribute to behavioral diagram complexity. For example, the element perspective focuses on attributes such as state, action state, and message, received high ratings, indicating that these elements are critical.

Significantly, the control flow perspective, which entails the sequential, decision, repetitive, and parallel aspects of diagram complexity, was strongly endorsed by the experts. The high percentages of respondents who "strongly agree" with the inclusiveness of these attributes support the framework's potential as a comprehensive tool for complexity analysis of behavioral diagrams. The interaction perspective, while slightly lower in mean ratings compared to the other perspectives reveal a significant contribution to the overall complexity of UML behavioral diagrams. The experts acknowledge that these interactions are critical in analyzing the complexity of these diagrams.

The experts strongly agree on adoption of the framework with a mean adoption rating of 4.55 and 63.6% implying that the framework is not only theoretically sound but also practically valuable. Software engineers and industry practitioners agree that the framework as a useful guide for analyzing and managing complexity in UML diagrams, which is an essential aspect of software design and development.

The findings from this study have significant implications for both practice and research. First, the high reliability scores and strong expert consensus indicate that the framework can be adopted as a standardized approach for assessing complexity in

UML behavioral diagrams. Its structured perspectives provide a consistent basis for evaluating model analyzability, which is essential in software design documentation and review. Second, the framework has practical utility in real-world development environments, where it can guide teams in identifying and managing complexity early in the modeling phase, thereby enhancing system maintainability and reducing the risk of design-related errors. Third, the integration of the framework into modeling tools and software engineering workflows can promote objectivity in complexity analysis and improve communication among stakeholders by offering a common language for interpreting behavioral diagrams. Finally, it lays the foundation for further academic inquiry into the relationship between diagram complexity and software quality attributes.

4.8 Summary

This chapter introduced the Perspective-Based Complexity Framework (PBCF), developed to identify and classify factors that influence the complexity of UML behavioral diagrams. The framework was structured around three main perspectives namely, element, control flow, and interaction, each with specific measurable attributes. An expert opinion survey was conducted to validate the framework, focusing on its relevance, inclusiveness, and potential for practical adoption. The results demonstrated strong expert support, with high average ratings and minimal variation in responses across all attributes and perspectives.

CHAPTER FIVE

A PERSPECTIVE-BASED METRICS VALIDATION FRAMEWORK (PBMVF)

5.1 Introduction

This chapter develops a new framework called the Perspective-Based Metrics Validation Framework (PBMVF). It is designed to aid in assessing whether the complexity measures for UML behavioral diagrams are valid. The main goal of this chapter is to fulfill the second research objective, which is to develop a theoretical validation framework for UML metrics based on perspectives. The chapter also includes findings from applying the proposed framework to current UML behavioral metrics. These results are then compared with those from other existing validation frameworks.

5.2 Requirements for the PBMVF

Requirements for the PBMVF were derived through a combination of theoretical analysis, literature review, and expert input obtained during the qualitative phase of the study. Foundational theories in software measurement, particularly measurement theory, Weyuker's properties, and validation frameworks namely Briand and Kaner, provided the basis for determining what constitutes a valid and meaningful metric. In addition, insights from the systematic review of existing UML behavioral diagram metrics and expert evaluations helped refine what aspects of complexity should be captured and how they should be categorized and assessed. These sources informed the development of a framework that not only ensures mathematical rigor but also considers the practical relevance of metrics within software modeling contexts.

Based on this process, the PBMVF was designed to meet the following requirements:

The framework should:

- i. Identify and define the key perspectives that contribute to the complexity of UML behavioral diagrams.
- ii. It should establish a structured methodology for assessing complexity by categorizing relevant attributes within each perspective.

- iii. The framework should provide well-defined measurement properties for each perspective, ensuring a quantifiable and objective evaluation of behavioral metrics.
- iv. These measurement properties should serve as validation criteria, verifying the accuracy, consistency, and applicability of the derived behavioral metrics.
- v. The framework should ensure that complexity assessments in UML behavioral diagrams are reliable and aligned with formal complexity measurement standards.

5.3 Theoretical Foundations

A structured approach that ensures both theoretical rigor and practical applicability of newly defined metrics is needed. The framework presented in this chapter is designed with the aim of validating complexity metrics of behavioral diagrams by defining measurement attributes together with their properties. This framework builds on objective one while addressing gaps in existing metrics validation frameworks.

5.3.1 Definitions

To define the complexity attributes of a behavioral diagram, the framework established by Briand et al. [39] in their study of systems and modules was adopted. Behavioral diagrams depict a system's dynamic aspects, including interactions, state transitions, and message exchanges among various system components. These diagrams are characterized by building elements, namely, objects (such as states, action states, lifelines), and the directed edges that establish relationships between these objects. Additionally, messages within the diagram illustrate communication between components, thereby forming the basis of system interactions. Therefore, a behavioral diagram is identified by its objects, edges and messages.

Formally, a behavioral diagram can be mathematically defined as a 3-tuple $\langle O, E, M \rangle$, where:

- i. O – represents the set of objects or elements within the behavioral diagram. These elements can include states in a statechart diagram, lifelines in a sequence diagram, or action states in an activity diagram. Each object plays a distinct role in the system's behavior and interaction flow.

- ii. E – represents the set of directed edges that establish relationships between the objects. These edges define the possible transitions between states in statecharts, the flow of control in activity diagrams, or the sequence of message exchanges in sequence diagrams. They provide the structural framework necessary for modeling dynamic behavior.
- iii. M – represents the set of messages exchanged between objects. Messages define the communication pathways and interactions that occur within the system. In sequence diagrams, for example, messages indicate the flow of information between lifelines, while in state diagrams, they represent events triggering state transitions.

This formal representation of behavioral diagrams offers a structured and mathematical foundation for analyzing their complexity. By defining the diagram in terms of its constituent objects, relationships, and message exchanges, it becomes possible to quantify, compare, and validate the complexity of different behavioral models systematically.

5.3.2 Complexity Perspective of Behavioral Diagrams

Behavioral diagram complexity can be analyzed from elements used in the diagram, how control flows through it, and the way different parts interact with each other. Each of these perspectives represents a different facet of complexity, contributing to a holistic understanding of a system's behavioral aspects. This structured approach ensures that all relevant aspects of complexity are considered, improving the accuracy of behavioral metric validation.

The element perspective looks at the basic parts that make up a UML behavioral diagram, such as states in a statechart, messages in a sequence diagram, and actions in an activity diagram. Complexity within this perspective is assessed based on the number, types, and interaction of these elements, influencing software analyzability.

The control flow perspective examines the execution of behavior within a UML behavioral diagram, taking into considerations of elements such as sequential execution, decision-making paths, loops, and parallel executions. The control flow complexity is crucial for evaluating system analyzability.

The interaction perspective focuses on how elements within a UML diagram communicate. The degree of interaction influences software performance and system analyzability. High levels of interaction can indicate a complex communication structure, impacting system efficiency.

5.3.3 Kaner's Framework and Briand's Metrics Framework

Kaner & Bond [35] introduced a practical framework consisting of ten evaluation questions aimed at assessing metric reliability, applicability, and relevance. This framework provides valuable guidelines for validating software metrics from a practical perspective, ensuring that metrics used in software engineering are meaningful, and applicable in real-world scenarios. The key strengths of Kaner's framework include its emphasis on practical usability, cost-effectiveness, and real-world validation, making it particularly useful for applied software metric evaluation.

However, despite its practical strengths, the framework lacks specificity in evaluating complexity attributes, particularly in the context of UML behavioral diagrams. The framework does not explicitly define how behavioral complexities should be measured, nor take into considerations for the structural or dynamic aspects of software systems. This makes it challenging to apply Kaner's framework in scenarios where behavioral interactions play a significant role in complexity assessment.

On the other hand, Briand et al. [39] developed a theoretical validation framework focused on fundamental software design properties such as size, length, complexity, cohesion, and coupling. This framework provides a structured approach to metric validation, ensuring that metrics are grounded in formal theoretical principles. The strengths of Briand's framework lie in its ability to rigorously define and classify software complexity attributes, making it an effective tool for validating design-level software metrics.

Despite its theoretical robustness, Briand's framework has limitations when applied to behavioral metrics. The framework addresses structural software attributes, such as object-oriented design metrics, and does not provide explicit validation criteria for runtime behavioral complexity in UML diagrams. Behavioral aspects such as state transitions, event-driven interactions, and message exchanges are not explicitly accounted for, making the framework less applicable for dynamic system analysis.

5.4 Measurement Attributes and their Properties

5.4.1 Element

Element complexity refers to the structural components of a UML behavioral diagram, including the different types of elements present, their quantity, and their relative complexity. To assess the complexity introduced by these elements, the following properties are defined:

Property 1 (Type of element): Is the metric measuring the complexity due to the presence of different types of elements in the diagram?

The first property examines whether the metric accounts for the complexity introduced by the presence of different types of elements within the diagram. Each UML behavioral diagram consists of unique structural components that define its purpose and behavior. The variety and nature of these elements significantly impact the diagram's complexity.

For instance, in a statechart diagram, the primary elements include different types of states such as simple states, orthogonal states, composite states, initial states, and final states. The presence of composite and orthogonal states adds complexity since they introduce nested and concurrent execution paths. Similarly, in a sequence diagram, the complexity is influenced by the types of messages exchanged between objects, including synchronous, asynchronous, return, and delayed messages. A synchronous message, which requires a response before execution continues, inherently increases complexity due to its blocking nature, whereas an asynchronous message does not have the same constraint. Additionally, activity diagrams include components such as initial states, action states, and final states, all of which contribute differently to the complexity of the system.

A well-structured complexity metric should recognize the different types of elements in a diagram and assess their impact on overall system complexity. Diagrams that contain a wider variety of elements typically require more effort to analyze due to their intricate structure.

Property 2 (Extent of the element): Is the metric measuring the extent of elements in the diagram?

The second property measures the extent to which elements appear within a diagram. The extent of an element refers to its frequency within the diagram, which directly correlates with complexity. A behavioral diagram containing a large number of elements naturally increases the effort required to analyze a diagram.

For instance, a statechart diagram with 30 states will be significantly more complex than one with only 10 states. Additionally, a sequence diagram with 40 lifelines and 170 messages will present more complexity than one with 3 lifelines and 10 messages. The number of components in a behavioral diagram plays a crucial role in determining its complexity, as a larger number of elements typically increases system interdependencies and the likelihood of execution errors.

A robust complexity metric should account for the total number of elements present in the diagram and use this count to quantify the extent of its complexity. This ensures that diagrams with high number of elements are identified as more complex than simpler diagrams with fewer components.

Property 3 (Weighted element complexity): Does the metric measure the complexity derived from the extent and complexity factor of an element?

The third property evaluates whether the metric incorporates the weighted complexity of each element by counting how many times each element appears and how difficult each one is considered to be. Some elements inherently introduce more complexity than others, and this should be factored into the measurement.

A statechart diagram with a composite state (which contains multiple nested states) is inherently more complex than a simple state, as it requires additional rules to manage entry, exit, and transitions. Significantly, in a sequence diagram, asynchronous message adds more complexity than an asynchronous message.

A valid complexity metric should assign weights to elements based on their inherent complexity and calculate the overall weighted complexity by multiplying the extent of an element by its weight. This approach provides a more accurate measure of diagram complexity rather than simply counting elements without considering their individual contributions to system complexity.

5.4.2 Control Flow

Control flow in UML behavioral diagrams defines how behavior is executed at runtime. It represents the logical sequencing of actions, decisions, and interactions, making it a critical factor in determining the complexity of a system. The complexity of control flow is influenced by the number of execution paths, conditional branches, loops, and parallel processes. To systematically assess control flow complexity, the following properties are defined:

Property 1 (Type of control flow): Does the metric measure the complexity due to the existence of categories of control flows in a diagram?

The first property assesses whether the metric considers the complexity introduced by different categories of control flow within a diagram. UML behavioral diagrams exhibit various forms of control flow, each of which contributes to complexity in different ways.

A behavioral diagram that contains multiple decision points, loops, and parallel execution flows will be significantly more complex than one that follows a simple sequential execution pattern. A reliable complexity metric should account for the different types of control flows present in the diagram and assess their impact on the system.

Property 2 (Extent of the control flow): Does the metric assess the extent of guards on a control flow?

The second property evaluates whether the metric assesses the extent of control flow based on the number of guards (Boolean conditions) present in the diagram. Guards define conditional execution paths by specifying constraints that must be satisfied for a transition or message exchange to occur.

As an example, in a statechart diagram, guarded transitions introduce additional complexity by requiring specific conditions to be evaluated before a state change can occur. Similarly, in an activity diagram, guard conditions within decision nodes dictate execution paths. The presence of multiple guards increases complexity as it introduces decision-making logic into the system, requiring additional effort to analyze and validate.

A meaningful complexity metric should count the number of guards and use this as an indicator of control flow complexity.

Property 3 (Depth of the edge): Is the metric measuring the complexity due to the distance covered by an edge?

The third property measures whether the metric considers the depth of execution paths in a diagram. The depth of an edge refers to the distance between the initial execution point and the final execution state within a diagram. Diagrams with deep execution paths, requiring multiple transitions or messages to reach completion, are inherently more complex.

A sequence diagram with a message that travels across multiple lifelines introduces more complexity than a local message between adjacent objects. Similarly, in a statechart diagram, deeply nested states require additional execution steps before reaching the final state, increasing the complexity of behavior execution.

A well-defined complexity metric should incorporate the depth of execution paths to reflect the real complexity introduced by long or deeply nested control flows.

Property 4 (Weighted control flow complexity): Does the metric measure the complexity derived from the extent, the distance covered and the complexity factor of the control flow?

The final property considers the overall weighted complexity of control flow by integrating three key factors namely, the weight of control flow types (sequential, decision, repetitive, parallel), the total depth of control flow edges (length of execution paths) and the extent of control flow (number of guard conditions and transitions).

By computing the product of these factors, a complexity metric can provide a comprehensive measure of control flow complexity.

5.4.3 Interaction

Interaction as a concept has been used based on how objects or elements communicate with each other. Also, it assesses the level of interaction of an object. According to our framework, an object communicates with another object via a message or link or transition, or edge. An object with more outgoing and incoming messages or links or

transition, or edges has a high level of interaction. Therefore, we have incoming and outgoing interactions. The interaction measurement concept has the following properties:

Property 1 (Type of interaction): Is the metric measuring the complexity due to the presence of incoming and outgoing interactions?

The type of interaction refers to incoming and outgoing interaction as depicted by incoming and outgoing messages or links or edge or transitions. A good metric should evaluate the different types of interaction in a diagram.

Property 2 (Total of incoming and outgoing messages): Does the metric count the total number of incoming and outgoing messages?

The extent of the interaction refers to how large is the interaction in terms of the presence of incoming and outgoing messages. A good metric should be based on the total count of incoming and outgoing messages.

Property 3 (Weighted interaction complexity): Does the metric measure the complexity derived from the incoming and outgoing messages and their complexity factor?

The complexity of an interaction is a product of the weight of an interaction and the total number of incoming and outgoing messages. A good metric should be based on the presence of incoming and outgoing messages together with their assigned weights.

5.5. Results

In this section, the proposed framework is validated with existing behavioral metrics and comparison with existing validation frameworks. The aim is to determine the comprehensive and effectiveness of the proposed framework in assessing behavioral complexity metrics.

5.5.1 Evaluation with Selected Behavioral Metrics

To evaluate the intuition of the proposed framework, several existing metrics were selected as experimental materials [23], [86], [87]. In [23], the metrics namely weighted number of states (WNS), weighted number of events (WNE) and weighted number of transitions (WNT) to measure the complexity of statechart diagrams are

defined. The WNS is a function of a category of state and the weight assigned to the state, the WNT is a function of types of transitions in the statechart diagram and the cognitive weights assigned to a sequence, branch, and loop while the WNE is a function of types of events in the statechart diagram and the weight assigned to the type of event.

In [86], three measures are introduced to help evaluate the complexity of behavioral diagrams: weighted message into lifeline (MIL), weighted message out of lifeline (MOL), and weighted number of lifelines (NOL). MIL refers to the number of message flows that end at a lifeline or object. MOL counts the message flows that start from a lifeline or object. NOL represents the total number of lifelines present in a sequence diagram.

In [87], defined complexity metrics for statechart diagrams namely, the state complexity ($State_{Comp}$) and transition complexity ($Trans_{Comp}$). The $State_{Comp}$ assesses complexity due to the presence of activities in a state while $Trans_{Comp}$ assesses complexity due to the presence of guards in a transition.

These metrics were grouped into element and control flow measurement attributes. The weighted number of states, the weighted number of events, state complexity, weighted messages into the lifeline, weighted messages out of the lifeline, and the weighted number of lifelines were categorized under element metrics. The weighted number of transitions and the transition complexity were placed under the control flow metrics.

5.5.1.1 Element Metrics

WNS and WNE metrics in [23] measure the presence of states and transitions in a statechart diagram. They are evaluated as follows

Property 1 (Type of element): Is the metric measuring the complexity due to the presence of different types of elements in the diagram?

A good measure should evaluate the elements composing a diagram. For instance, the weighted number of states (WNS) evaluates complexity due to the presence of states

while the WNE evaluates complexity due to the presence of events in a statechart diagram.

Property 2 (Extent of the element): Is the metric measuring the extent of elements in the diagram?

A good element metric shows the size of a diagram by counting how many elements it has. WNS measures the number of state types in a statechart, and WNE measures the number of event types.

Property 3 (Weighted element complexity): Does the metric measure the complexity derived from the extent and complexity factor of an element?

The weighted complexity of a UML diagram is the product of the weight and extent of an element. Therefore, a good metric should be based on the presence of elements together with their weights. The WNS, and WNSE return complexity values based on the category of elements and their complexity coefficients.

The WNE and WNT metrics satisfied the element properties as specified in the proposed framework. The summary results are illustrated in Table 5.1.

Table 5.1: Summary of Validation Results

Property	WNSA	WNE
Property 1	Yes	Yes
Property 2	Yes	Yes
Property 3	Yes	Yes

MOL, MOL, and NOL metrics measure messages and lifelines in a sequence diagram [86]. They are validated as follows:

Property 1 (Type of element): Is the metric measuring the complexity due to the presence of different types of elements in the diagram?

The MOL and MIL evaluate complexity due to the presence of messages, while the NOL evaluates complexity due to the presence of lifelines in a sequence diagram.

Property 2 (Extent of the element): Is the metric measuring the extent of elements in the diagram?

A good element metric should show the size of a diagram by counting the number of elements in it. MOL shows how many messages go out from a lifeline, MIL shows how many messages come into a lifeline, and NOL shows how many lifelines are in a sequence diagram. This means all these metrics meet this requirement.

Property 3 (Weighted element complexity): Does the metric measure the complexity derived from the extent and complexity factor of an element?

The weighted complexity of a UML diagram is the product of the weight of and the extent of an element. Therefore, a good metric should be based on the presence of elements together with their weights.

The MOL, MIL and NOL metrics did not return complexity values based on the category of elements and their assigned weights. This is because no weights have been assigned to the types of messages and lifelines. This information is presented in Table 5.2.

Table 5.2: Validation Results.

Property	MOL	MIL	NOL
Property 1	Yes	Yes	Yes
Property 2	Yes	Yes	Yes
Property 3	No	No	No

The State_{Comp} metric in [87] measures states in a statechart diagram. It is evaluated as follows:

Property 1 (Type of element): Is the metric measuring the complexity due to the presence of different types of elements in the diagram?

The State_{Comp} measures how complex a statechart diagram is based on the number of states and activities it includes.

Property 2 (Extent of the element): Is the metric measuring the extent of elements in the diagram?

A good element metric shows the size of a diagram by counting its elements. The StateComp metric counts the number of states and activities in a statechart, so it meets this condition.

Property 3 (Weighted element complexity): Does the metric measure the complexity derived from the extent and complexity factor of an element?

The weighted complexity of a UML diagram is the product of the weight and extent of an element. Therefore, a good metric should be based on the presence of elements together with their weights. The StateComp returns complexity values based on the category of elements and their assigned weights.

The StateComp satisfied all the properties of the element attribute as presented in Table 5.3.

Table 5.3: Validation of StateComp metric

Property	StateComp
Property 1	Yes
Property 2	Yes
Property 3	Yes

5.5.1.2 Control Flow

The WNT metric in [23] assesses the presence of transitions in a statechart diagram. It is validated as follows:

Property 1 (Type of control flow): Does the metric measure the complexity due to the existence of categories of control flows in a diagram?

A good metric should evaluate the different types of control flows in a diagram. The weighted number of transitions (WNT) metric can assess complexity due to the presence of a sequence, decision, and loop.

Property 2 (Extent of the control flow): Does the metric assess the extent of guards on a control flow?

A good metric should return the size of the control flow based on the number of guard conditions. The WNT metric does not evaluate the complexity of a diagram based on the total number of guards in a control flow. Therefore it does not satisfy this property.

Property 3 (Depth of the edge): Is the metric measuring the complexity due to the distance covered by an edge?

A good metric should return the distance covered by an edge. The WNT does not evaluate the complexity due to the distance covered by an edge.

Property 4 (Weighted control flow complexity): Does the metric measure the complexity derived from the extent, the distance covered, and the complexity factor of the control flow?

A good metric should be based on the presence of control flow, together with its assigned weights. WNT can evaluate complexity due to the presence of control flows together with their assigned complexity weights. Table 5.4 shows the summary of the validation of the weighted number of transitions metric.

Table 5.4: Validation of WNT Metric

Property	WNT
Property 1	Yes
Property 2	No
Property 3	No
Property 4	Yes

The $\text{Trans}_{\text{Comp}}$ metric in [87] measures transitions in a statechart diagram. It is evaluated as follows:

Property 1 (Type of control flow): Does the metric measure the complexity due to the existence of categories of control flows in a diagram?

A good metric should evaluate the different types of control flows in a diagram. The $\text{Trans}_{\text{Comp}}$ metric can assess complexity due to the presence of a transition and guard in a statechart diagram.

Property 2 (Extent of the control flow): Does the metric assess the extent of guards on a control flow?

A good metric should return the size of the control flow based on the number of guard conditions. The $\text{Trans}_{\text{Comp}}$ metric evaluates the complexity of a diagram based on the total number of guards in a control flow. Therefore it satisfies this property.

Property 3 (Depth of the edge): Is the metric measuring the complexity due to the distance covered by an edge?

A good metric should return the distance covered by an edge. The $\text{Trans}_{\text{Comp}}$ does not evaluate the complexity due to the distance covered by an edge.

Property 4 (Weighted control flow complexity): Does the metric measure the complexity derived from the extent, the distance covered, and the complexity factor of the control flow?

A good metric should be based on the presence of control flow, together with its assigned weights. $\text{Trans}_{\text{Comp}}$ can evaluate complexity due to the presence of control flows together with their assigned complexity weights. Table 5.5 shows the summary of the validation of the $\text{Trans}_{\text{Comp}}$ metric.

Table 5.5: Validation of $\text{Trans}_{\text{Comp}}$ metric

Property	WNT
Property 1	Yes
Property 2	Yes
Property 3	No
Property 4	Yes

5.5.2 Comparison with Existing Validation Frameworks

Whenever new frameworks are proposed, it is always a good idea to compare it with existing alternatives. In this section, the proposed framework is compared with three existing metrics validation frameworks to assess its completeness in terms of coverage of different aspects and perspectives of complexity. This exercise exposes the strengths and limitations of compared frameworks.

In the evaluation of each framework, three dimensions were considered. First, assess whether the framework considers multiple metrics, which is essential in ensuring a broad evaluation of software complexity. Second, examine whether the framework incorporates different aspects of complexity, recognizing that software complexity extends beyond structural properties to include behavioral features. Third, analyze whether the framework accounts for behavioral perspectives, an important factor in the assessment of UML behavioral diagrams and dynamic interactions within software systems.

The comparison reveals notable differences in how each framework approaches complexity validation. Weyuker's [73] framework, while effective in considering multiple metrics, does not incorporate different aspects of complexity or behavioral perspectives. Briand et al. [57] offer a more comprehensive approach by considering multiple metrics and different aspects of complexity, but they do not address behavioral perspectives, which are essential for evaluating the dynamic behavior of software systems. Comparably, Kaner & Bond [35] focus on multiple metrics but do not take into account different aspects of complexity or behavioral perspectives. In contrast, our proposed perspective-based framework incorporates all three dimensions, making it more robust in validating software complexity holistically. Table 5.6 represents these findings.

Table 5.6: Comparison with Existing Validation Frameworks.

Framework	Considers multiple metrics	Considers different aspects of complexity	Considers behavioral perspectives
Weyuker's	Yes	No	No
Briand's et al.	Yes	Yes	No
Kaner & Bond	Yes	No	No
Proposed perspective-based framework	Yes	Yes	Yes

5.6 Discussion

Comparison results show that while some metrics met the framework's validation properties, others failed to account for weighted complexity and specific behavioral

interactions. These results confirm that PBMVF provides a stronger and more detailed approach to complexity validation compared to traditional frameworks.

Existing validation frameworks, such as those proposed by Weyuker, Briand, and Kaner, primarily focus on structural properties and general complexity attributes but lack specific provisions for analyzing dynamic interactions and runtime behavioral complexities within UML diagrams. The PBMVF fills this gap by incorporating behavioral perspectives, making it a more approach for validating UML behavioral complexity metrics.

PBMVF enhances both theoretical and practical validation by introducing measurement attributes and validation properties that assess complexity across different perspectives. A comparative validation of the PBMVF against existing validation frameworks further reveals its exclusivity. While Weyuker's, Briand's, and Kaner's frameworks address certain aspects of complexity validation, none of them fully incorporate all three critical dimensions namely multiple metrics, different aspects of complexity, and behavioral perspectives.

Weyuker's framework, for instance, effectively considers multiple metrics but does not incorporate different aspects of complexity or behavioral perspectives. Similarly, Briand's framework provides a more structured approach by considering different aspects of complexity, but it does not account for dynamic interactions in behavioral diagrams. Kaner's framework focuses on practical usability but lacks theoretical depth. In contrast, the PBMVF effectively integrates all three dimensions, making it more comprehensive and applicable for evaluating software complexity.

PBMVF has significant implications for software engineering, particularly in evaluating the analyzability of UML behavioral diagrams. It provides a systematic and structured approach to assessing behavioral complexity. This framework can aid software engineers in improving design decisions, optimizing software architecture, and enhancing model clarity. These improvements, in turn, contribute to higher software quality, reduced debugging time, and better system performance. Furthermore, the PBMVF ensures standardization in complexity validation, making it easier to compare and analyze different UML behavioral diagrams.

5.7 Summary

In this chapter, the PBMVF framework was developed by integrating and modifying Kaner's practical validation framework and Briand's theoretical validation framework. The framework provides a more structured, comprehensive, and systematic approach to evaluating complexity in UML behavioral diagrams. The framework introduces a perspective-based approach that categorizes complexity into element complexity, control flow complexity, and interaction complexity. This classification allows for a more detailed and holistic assessment of UML behavioral diagrams, ensuring that all critical aspects of behavioral complexity are taken into consideration.

CHAPTER SIX

A METRICS SUITE FOR UML BEHAVIORAL DIAGRAMS BASED ON COMPLEXITY PERSPECTIVES

6.1 Introduction

In this chapter, a suite of metrics for measuring the complexity of UML behavioral diagrams are defined. The chapter solves the third research objective as described in chapter one, which is to define a suite of theoretically sound metrics based on the perspective-based complexity framework.

6.2 Determination of Measurable Attributes

To understand and measure the complexity of UML behavioral diagrams, the study used a perspective-based complexity framework. This approach aided in identifying specific attributes within the diagrams that can be measured to show how complex they are. These measurable attributes are grouped into three main categories:

- i. Attributes related to diagram elements

These are based on the core parts that make up UML behavioral diagrams. For instance, a state in a statechart diagram, an action state in an activity diagram, and a message in a sequence diagram are considered measurable elements in this category.

- ii. Attributes related to control flow

These describe how the process moves through the diagram. They include patterns such as simple sequences, repeated loops, decision-making paths, and activities that occur at the same time (parallel flows).

- iii. Attributes related to interaction

These focus on how different parts of the diagram connect and exchange information. Interaction is tracked by counting the number of transitions, links, or messages that go into or come out from a specific point in the diagram. These are referred to as incoming and outgoing interactions.

6.3 Metrics Definition

This section introduces a set of complexity measures based on how UML behavioral diagrams are viewed in terms of structure and behavior. Some of the metrics are newly defined while others are modified from existing complexity metrics to suit behavioral diagrams.

6.3.1 Element Complexity Metrics

UML behavioral diagrams are made up of elements, which define how the diagram is structured. Each diagram type such as sequence, statechart, or activity has its own unique elements. These elements directly affect how complex a diagram is. In this study, the element-based perspective includes measurable components such as action states, states, and messages. These are further illustrated in Figure 6.1.

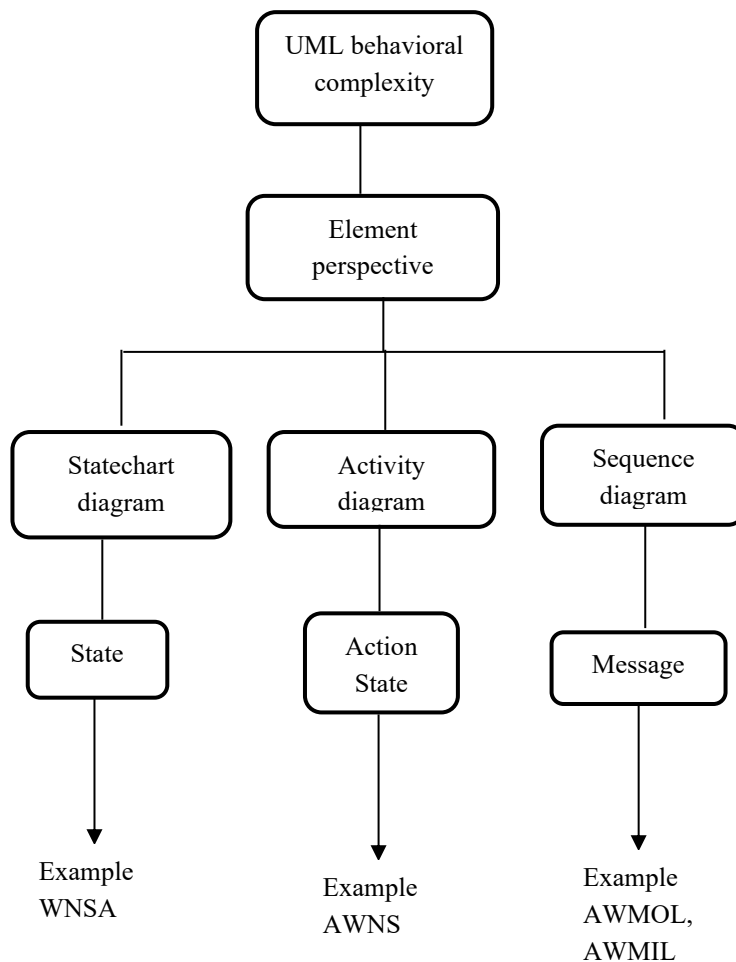


Figure 6.1: The Element Perspective

6.3.1.1 Element Complexity Metric for Statechart

A statechart diagram shows the different stages (states) an object goes through in its life [92]. The main parts of a statechart include states, transitions, events, activities, and guard conditions. These are shown in Figure 6.2.

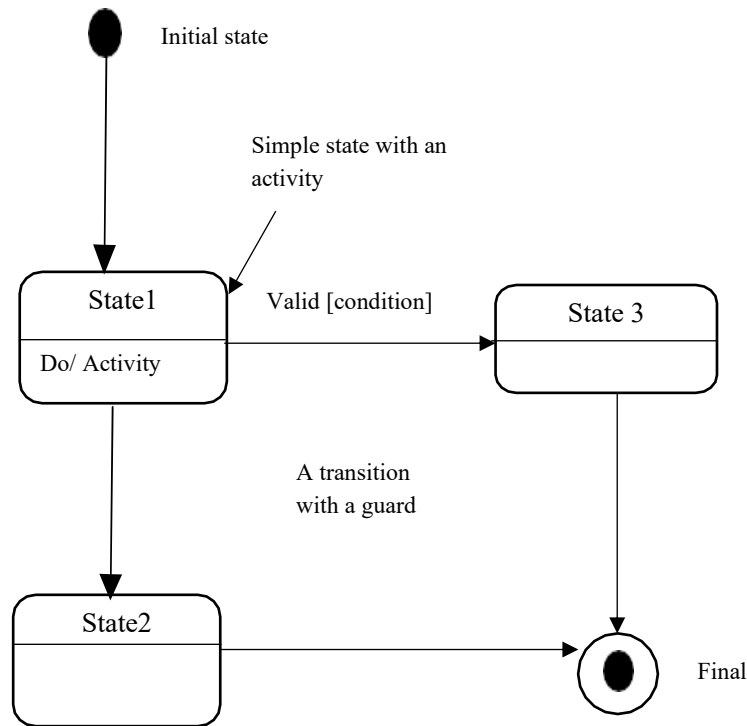


Figure 6.2: Statechart Diagram

Base Metrics for Statechart:

The base metrics for statechart diagrams are as follows: the Number of States (NS) refers to the total different types of states in the diagram. The Number of Entry Actions (NEA) measures the total actions performed when entering states, while the Number of Activities (NA) counts the total actions happening within the states. The Number of Transitions (NT) records all the transitions between states, Number of Guards (NG) identifies all conditions used to control transitions, and Number of Events (NE) is the total number of events that trigger transitions [93].

Derivation of the Weighted Number of States and Activities (WNSA):

An advanced metric called Weighted Number of States and Activities (WNSA) was developed to capture more detail. This metric is an improvement over previous

versions, such as the one proposed in [23], which did not consider internal activities inside a state, and in [87], which did not factor in different types of states in a statechart diagrammed.

WNSA uses weights to represent complexity of simple states, composite states, submachine states, and orthogonal states are all given different weights based on how complex they are. The weights used in the Weighted Number of States approach (WNSA) were derived through a combination of theoretical reasoning and intuitional support to accurately reflect the varying levels of complexity associated with different state types and activities in UML state machine diagrams. For example, an initial and final states carry a weight of 1, simple states are weighted at 1.5, and more complex states such as composite and submachine states carry a weight of 2.5. Orthogonal states are assigned a weight of 3, and history states a weight of 2. Furthermore, the number of activities within a state adds to its weight. A state with no activity equals 1.0, one activity equals 1.2, two equals 1.4, and three activities equal 1.6. The use of 0.2 increments factor is grounded in cognitive scaling theory, which suggests that human perception of complexity tends to grow in small, manageable steps rather than abrupt jumps. The WNSA is calculated by multiplying the type weight and activity weight for each state and combining the results. Table 6.1 represents the weights assigned to states with activities.

WNSA is defined as:

$$WNSA = \prod_{i=1}^n (W_i X_i A_w)$$

X represents the specific category of a state, *i* marks the index of the first state being considered, W refers to the weight given to each state type based on its complexity, n is the total count of states, and A stands for the weight given to states that include internal activities.

Table 6.1: Complexity of State Activities

Number of Activities inside a state	Weight (A_w)
State with no activity	1.0
1	1.2
2	1.4
3	1.6

6.3.1.2 Element Complexity Metric for Sequence diagrams

A sequence diagram shows how different objects interact with one another by exchanging messages. The main parts of this diagram are the objects themselves and the messages they send. Messages come in four forms: asynchronous, synchronous, return, and delayed. In an asynchronous message, the sender must pause and wait until the receiver finishes processing. In a synchronous message, the sender delivers the message and keeps working without waiting. A return message is the response sent back after an asynchronous message. A delayed message takes longer to reach its destination. The different parts of a sequence diagram are illustrated in Figure 6.3.

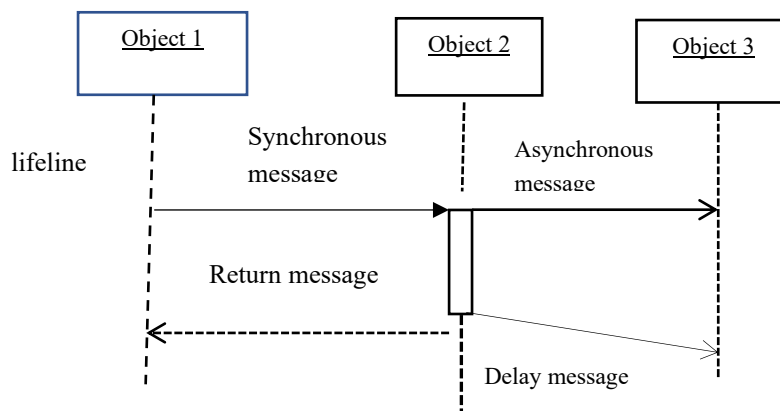


Figure 6.3: Elements of a Sequence Base Metrics for Sequence diagram

In a sequence diagram, three primary metrics are used to measure its complexity. These are: Message into Lifeline (MIL), Message out of Lifeline (MOL), and Number of Lifelines (NOL) [94]. MIL refers to the total number of messages received by each lifeline, while MOL counts the number of messages sent from it. NOL represents the number of different lifelines, or objects, that appear in the sequence diagram.

Definition of Sequence Metrics

The definition of sequence metrics in this study is based on the observation that different types of messages in UML sequence diagrams represent varying levels of behavioral complexity, depending on the nature of their execution and interaction. For example, delay messages are given a weight of 1.0 because they require time before reaching the receiver. Synchronous messages are considered slightly more complex and are assigned a weight of 1.2, as they involve a single event. Asynchronous messages and their corresponding return messages are more complex due to the possibility of extra events during communication, so they are each given a weight of 1.4. The adoption of a 0.2 increment factor is based on principles of cognitive scaling, which propose that people generally perceive increases in complexity in gradual, manageable steps rather than through sudden, large changes. The different complexity levels of these message types are summarized in Table 6.2.

Table 6.2: Complexity of Messages

Message	Weight (W_j)
Delay message	1.0
Synchronous message	1.2
Asynchronous and return message	1.4

Also, the complexity of a message increases when it travels across more lifelines. To reflect this, weights are used based on the distance a message covers. If a message moves through just one lifeline, it is given a weight of 1.0. If it passes through two lifelines, the weight increases to 1.2, and for three lifelines, it is raised to 1.4. These distance-based weights are shown in Table 6.3.

Table 6.3: Complexity of Length of Message

Lifelines Covered	Weight (L_w)
1	1.0
2	1.2
3	1.4

To measure the complexity of a sequence diagram, a set of new metrics has been introduced. These are modified forms of the MIL and MOL metrics in [86]. The approach by Maina et al. [86] did not include the complexity caused by different types of messages and the distances they travel across lifelines.

i) Adjusted Weighted Message into Lifeline (AWMIL)

The Adjusted Weighted Message into Lifeline (AWMIL) is calculated based on the type of message received by a lifeline and the weights given to both the message type and how far it travels. It is expressed using the following formula:

$$AWMIL = \prod_{j=1}^n (W_j I_j L_w)$$

In this formula, I represents the message type, while j indicates the starting point for counting. W refers to the complexity weight given to each message type, n is the total number of messages, and L stands for the weight assigned based on how far the message travels.

ii) Adjusted Weighted Message out of lifeline (AWMOL)

The Adjusted Weighted Message out of Lifeline (AWMOL) is calculated using the type of message sent from a lifeline, along with the weights given to its type and the distance it covers. The formula for AWMOL is:

$$AWMOL = \prod_{k=1}^n (W_k O_k L_w)$$

In this equation, O stands for the type of message being sent, and k marks the starting point for counting those messages. W is the weight that reflects how complex each message type is, n is the total number of messages, and L represents the weight based on the message's travel distance.

iii) Total Sequence complexity (TSC)

Total Sequence Complexity (TSC) is determined using both AWMIL and AWMOL values. To compute it, AWMIL is first multiplied by AWMOL, which gives the total communication paths in the sequence diagram. This result is then squared to get the final TSC value. The formula for TSC is defined as follows:

$$TSC = ((\prod_{j=1}^n (W_j I_j L_w)) * (\prod_{k=1}^n (W_k O_k L_w)))^2$$

6.3.1.3 Activity Diagrams Metrics

An activity diagram works like a flowchart that shows how actions move from one step to the next [95], [96]. It includes components such as action states, edges, and start and end states. Action states describe what the system does at a particular point, while edges link one action to another. Sometimes, an edge includes a guard, which is a condition that must be true for the action to proceed. The initial state marks where the process begins, and the final state indicates where it ends [95]. The components of an activity diagram are illustrated in Figure 6.4.

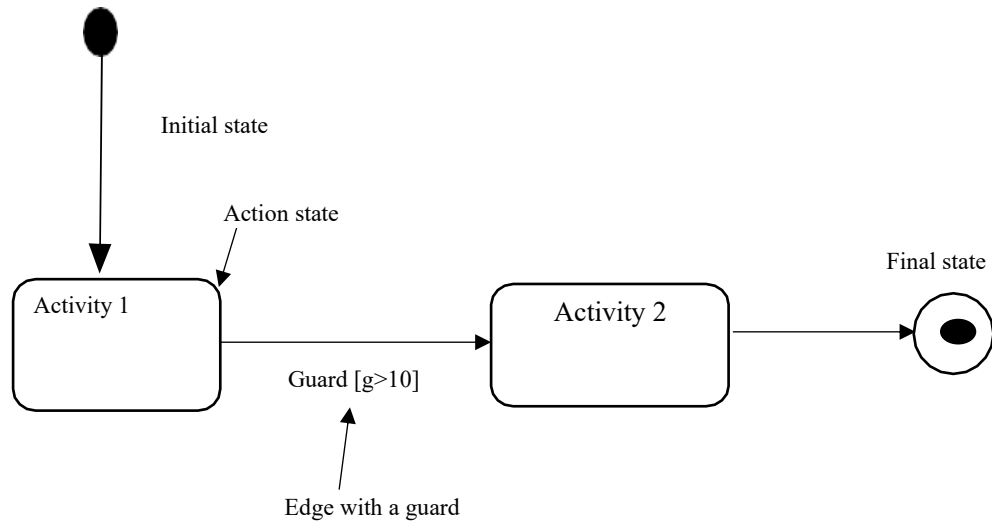


Figure 6.4: Elements of an Activity Diagram

Base Metrics for Activity diagram

The basic measures used in an activity diagram are the number of nodes and number of linearly independent paths through the diagram, as defined in [95]. These metrics

help to estimate how complex the diagram is by considering both the number of states and the connections (edges) between them.

Definition of Activity Metrics

Adjusted Weighted Number of State (AWNS) is as a result of the modification of Weighted Number of States (WNS) defined in [23]. This metric is based on the different types of states found in an activity diagram. A state that performs a function (like an action state) is given a weight of 2, while the initial and final states, which do not carry out any behavior, are each assigned a weight of 1. The AWNS value is calculated using the following formula:

$$AWNS = \sum_{U=1}^n (W_U S_U)$$

In this formula, S represents the total number of states grouped by type, u indicates the starting point for counting these states, W is the weight that reflects the complexity of each state type, and n is the total number of states in the activity diagram.

6.3.2 Aggregate Control Flow Complexity Metric

At runtime, a system's behavior is expressed through different control structures that guide the flow of operations. These structures include sequence flows, decisions, repeated loops, and parallel or simultaneous processes. Figure 6.5 highlights the attributes used to measure complexity from the control flow perspective.

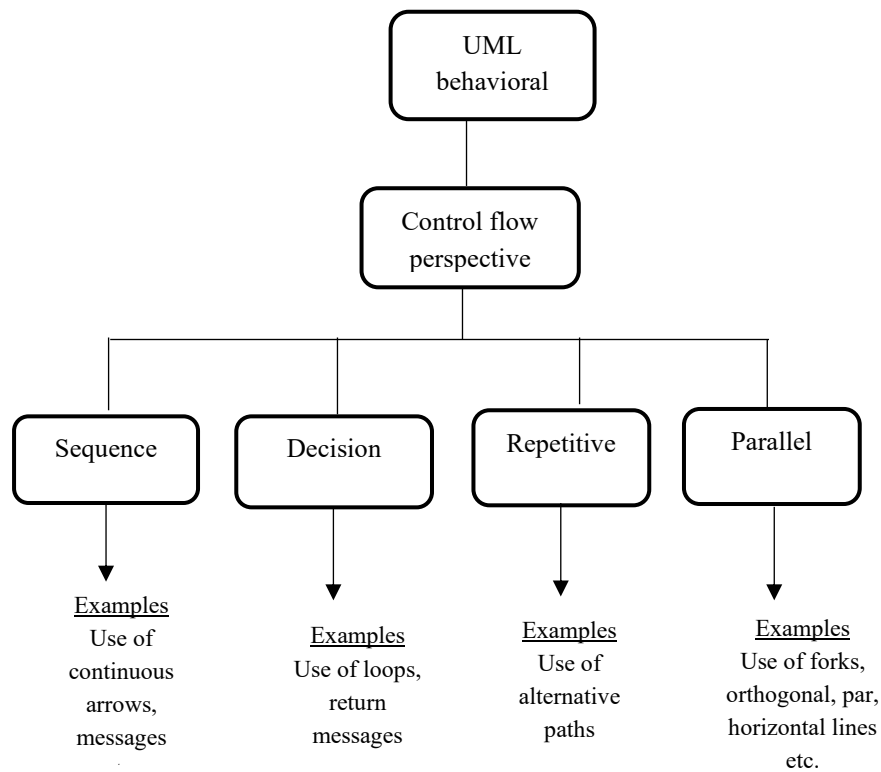


Figure 6.5: The Control Flow Perspective

Control structures are an important component of UML behavioral diagrams like statechart, sequence, and activity diagrams. These structures influence how complex a diagram is. To reflect this, each type of control structure is given a specific weight. For example, a sequence flow weights 1, a decision point is given a weight of 2, a loop is assigned 3, and a parallel process is assigned a weight of 4, as outlined in Table 6.4.

Table 6.4: Control Flow Complexity

Type	Activity	Wc
Sequence	One path arrow	1
Decision	Choice, if, else	2
Loop	Arrow to and back to component/object	3
Parallel	Forks, joins, orthogonal regions, par, horizontal lines	4

Sequences that include guard conditions are considered more complex than those without them. To reflect this added complexity, weights are assigned based on how many guards are present in a sequence. A sequence without any guard is given a

weight of 1.0. If there are 1, 2, or 3 guards, the weights increase to 1.2, 1.4, and 1.6 respectively. For sequences with 4, 5, or 6 guards, the weights are 1.8, 2.0, and 2.2. These values are listed in Table 6.5.

Table 6.5: Complexity of Sequence Guards

Number of guards inside a sequence	Weight (G_w)
Sequence with no guard	1.0
1	1.2
2	1.4
3	1.6
4	1.8
5	2.0
6	2.2

The Aggregate Control Flow Complexity (ACFC) metric is introduced to measure how control flow structures contribute to the overall complexity of a diagram. This metric builds upon and extends the Cognitive Functional Size (CFS) method developed in [85]. To compute ACFC, the complexity weights assigned to each type of control flow are combined to give a total value. The formula for calculating ACFC is shown in subsequent sections.

$$ACFC = \prod_{i=1}^n G_w \sum_{j=1}^m \sum_{k=1}^p \sum_{q=1}^r W_c(ijkq)$$

In this equation, W_c represents the complexity weight given to each control structure type, while G_w stands for the weight linked to the number of guards within a sequence. The variable n refers to the total number of sequences, m is the number of decision points, p is the count of loops, and r indicates the number of parallel structures included in the behavioral diagram.

6.3.3 Interaction Metrics

Interaction in UML behavioral diagrams takes place when elements like states, action states, or lifelines exchange messages, links, edges, or transitions. This type of communication between objects adds to the overall complexity of the diagram. The

measurable features used to assess interaction complexity are the incoming and outgoing messages as illustrated in Figure 6.6.

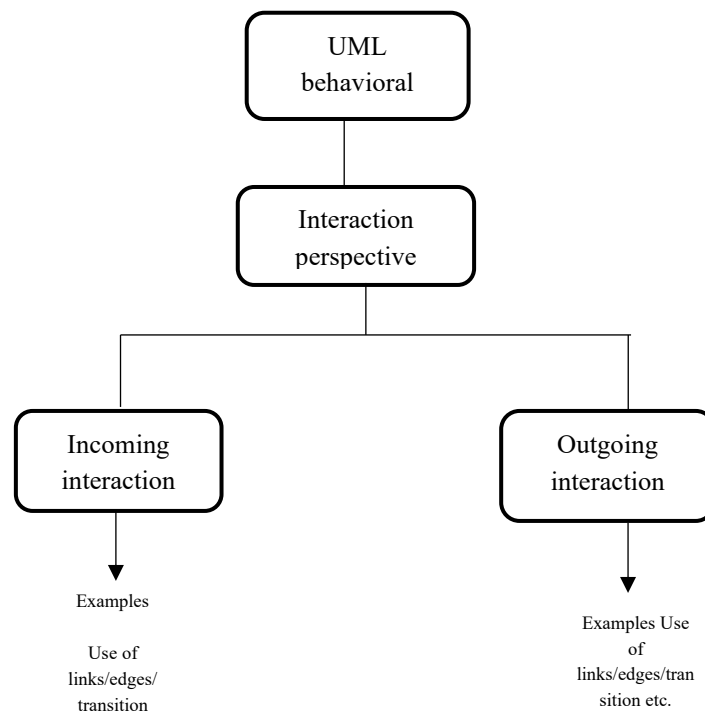


Figure 6.6: The Interaction Perspective

Objects that have more incoming or outgoing messages, links, edges, or transitions are seen as more interactive than those with fewer connections. Because of this, interaction complexity is measured by assigning weights based on how many incoming and outgoing connections an object has.

An object with only one incoming connection gets a weight of 1.0. If there are two incoming messages or links, the weight increases to 1.2. When there are three and four incoming connections, the weights are 1.4 and 1.6, respectively. These interaction weights based on the number of incoming messages are shown in Table 6.6.

Table 6.6: Complexity of Incoming Messages

Number of incoming messages or edges or transitions	Weight (A_i)
1	1.0
2	1.2
3	1.4
4	1.6

An object that sends out more messages, links, edges, or transitions is considered to have a higher level of interaction, which increases the diagram's complexity. To reflect this, different weights are given based on the number of outgoing connections. An object with one outgoing connection is given a weight of 1.5, while two outgoing connections raise the weight to 2.0. If there are three or four outgoing messages or links, the assigned weights are 2.5 and 3.0, respectively. For objects with five, six, or seven outgoing connections, the weights increase to 3.5, 4.0, and 4.5. Table 6.7 displays these weights based on the number of outgoing messages.

Table 6.7: Complexity of Outgoing Messages

Number of outgoing links or edges or messages	Weight (A_i)
1	1.5
2	2.0
3	2.5
4	3.0
5	3.5
6	4.0
7	4.5

To measure how complex the interactions are within a diagram, the following metrics have been defined:

i) Incoming Interaction Complexity (IIC)

This metric calculates interaction complexity based on how many incoming messages, links, edges, or transitions an object receives, along with the corresponding weights assigned to each. The formula used to compute IIC is:

$$IIC = \sum_{t=1}^n (W_t I_t)$$

In this equation, I represents the total count of incoming messages, links, edges, or transitions, t marks the starting point for counting these values, while W is the weight that reflects the complexity based on how many incoming messages the object has.

ii) Outgoing Interaction Complexity (OIC)

The Outgoing Interaction Complexity (OIC) measures how much an object contributes to interaction complexity based on the number of outgoing messages, links, edges, or transitions it has, along with their assigned weights. The OIC is calculated using the following formula:

$$OIC = \sum_{q=1}^n (W_q O_q)$$

In this formula, O stands for the total number of outgoing messages, links, edges, or transitions, q represents the starting point for counting these values, and W refers to the complexity weight given based on how many outgoing messages the object has.

iii) Total Interaction Complexity (TIC)

Total Interaction Complexity (TIC) is calculated by combining both the Incoming Interaction Complexity (IIC) and the Outgoing Interaction Complexity (OIC). TIC formula is:

$$TIC = \sum_{t=1}^n (W_t I_t) + \sum_{q=1}^n (W_q O_q)$$

In this formula, the first summation measures the complexity from incoming interactions, and the second summation accounts for the complexity from outgoing interactions.

6.4 Results

This section presents validation results of complexity metrics through calculating real life scenarios, comparison with existing metrics and Theoretical validations.

6.4.1 Calculating Real Life Scenarios

This section presents real-life scenarios to demonstrate the intuitiveness and applicability of the proposed element, control-flow, and interaction metrics. The diagrams selected for analysis intentionally vary in complexity to ensure a

comprehensive evaluation of the metrics across different modeling situations. The variation in complexity is reflected in the number and types of elements used, the depth of control structures, and the intricacy of interactions among system components

6.4.1.1 Scenario I: Statechart Diagrams

Computer Gaming Statechart Diagram

To provide a clear demonstration of how the proposed metrics suite can be effectively applied to evaluate UML statechart diagrams, an illustrative example is shown in Figure 6.7. This figure represents a statechart model derived from a computer game scenario. The statechart captures and visualizes the behavioral dynamics of a player as they interact with different components of the game environment. Each state in the diagram corresponds to a distinct stage within the game. Transitions between these states are triggered by specific player actions or game events, such as pressing a key, defeating an enemy, or entering a new area.

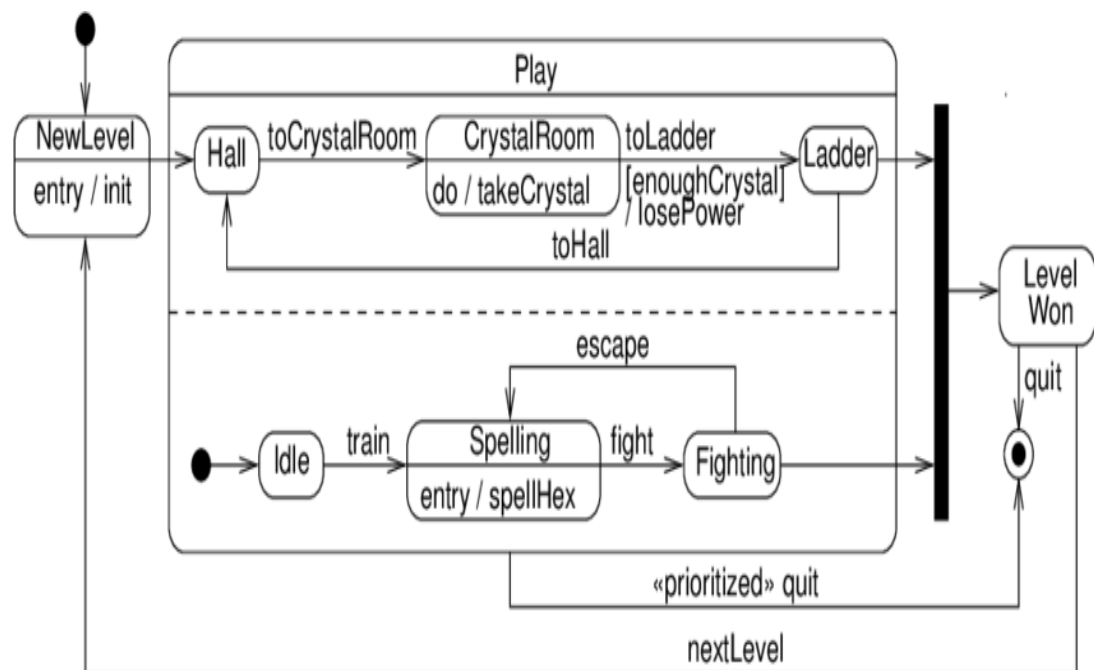


Figure 6.7: Computer Gaming Statechart Diagram [114]

i) Calculating Weighted Number of State Activities (WNSA)

Weighted Number of States and Activities (WNSA) is calculated using the following steps:

1. Note the number of each category of state.
2. For each category of state, count the number of activities inside it.
3. Get the WNSA complexity value by multiplying the type of state with its corresponding weight and weight of activities inside the state.

$$WNSA = \prod_{i=1}^n (W_i X_i A_w)$$

As depicted in Figure 6.7, the statechart diagram modeling the computer gaming scenario comprises a total of 11 basic (simple) states. Among these, 8 states do not contain any associated actions, whereas 3 states are assigned a single activity to be performed. In addition, the diagram includes one final state, two initial states, and a single orthogonal state. The metric calculation proceeds by applying the values to the respective formula as shown below:

- 8 (simple states with no activities) * 1.5 (weight of simple states) * 1 (weight of state with no activity) = $8 * 1.5 * 1.0 = 12$
- 3 (simple states with 1 activity) * 1.5 (weight of simple states) * 1.2 (weight of state with one activity) = $3 * 1.5 * 1.2 = 5.4$
- 2(initial states) * 1.0 (weight of an initial state) = $2 * 1 = 2$
- 1(initial state) * 1.0 (weight of an initial state) = $1 * 1 = 1$
- 1(orthogonal state) * 3.0 (weight of an orthogonal state) = $1 * 3 = 3$

$$WNSA = 12 + 5.4 + 2 + 1 + 3 = 23.4$$

ii) Calculating Aggregate Control flow complexity (ACFC)

To calculate the ACFC of the diagram in Figure 6.7, the following steps are followed:

1. Note the number of each category of control flow i.e. the sequence, decision, loop, and parallel.
2. For each category of sequence, count the number of guards inside it.
3. Get the ACFC complexity value by multiplying the type of each control flow with its corresponding weight.

$$ACFC = \prod_{i=1}^n G_w \sum_{j=1}^m \sum_{k=1}^p \sum_{q=1}^r W_c(ijkq)$$

The statechart diagram for the computer gaming system features a total of 13 transitions that do not include any guard conditions, alongside a single transition that incorporates one guard. Moreover, the diagram exhibits 3 looping transitions and 2 instances of parallel control flow paths. Substituting the formula as follows:

- 13 (sequences) * 1(weight of the sequence)* 1(weight of sequence with no guard) = 13*1*1=13
- 1 (sequence) * 1(weight of the sequence)* 1.2 (weight of sequence with one guard) =1*1*1.2=1.2
- 3 (loops) * 3 (weight of a loop) = 3*3 = 9
- 2 (parallels) * 4 (weight of a parallel) = 2*4=8

$$ACFC = 13 + 1.2 + 9 + 8 = 31.2$$

iii) Calculating Total Interaction Complexity (TIC)

To calculate the TIC, the following steps are followed:

1. For each object or state, count the number of incoming transitions and multiply by its corresponding weight to get the Incoming Interaction Complexity (IIC).
2. For each object or state, count the number of outgoing transitions and multiply by its corresponding weight to get the Outgoing Interaction Complexity (OIC).
3. Get the TIC through summation of IIC and OIC

$$TIC = \sum_{t=1}^n (W_t I_t) + \sum_{q=1}^n (W_q O_q)$$

In the context of the computer gaming system model, the statechart consists of 11 distinct states. Of these, 4 states receive exactly one incoming transition, another 4 states are linked by two incoming transitions each, while the remaining 3 states do not have any incoming transitions. Using these values, the metrics can be computed by applying them to the corresponding formulas as outlined below:

i) Calculating Incoming Interaction Complexity (IIC)

- 4 (4 states with one incoming transition) * 1 (Weight of one incoming transition) = $4 * 1.0 = 4.0$
- 4 (4 states with two incoming transitions) * 1.2 (Weight of two incoming transitions) = $4 * 1.2 = 4.8$
- 3 (3 states with no incoming transition) * 0 (Weight of zero transition) = $3 * 0 = 0$

$$\text{IIC} = 4 + 4.8 + 0 = 8.8$$

ii) Outgoing Interaction Complexity (OIC)

Additionally, the diagram shows that 3 of the states each have two outgoing transitions, while 6 states are connected to just one outgoing transition. There is also one state that does not lead to any subsequent state. These values are then applied to the relevant formulas for metric computation as demonstrated below:

- 6 (6 states with one outgoing transition) * 1.5 (Weight of one outgoing transition) = $6 * 1.5 = 9$
- 3 (3 states with two outgoing transitions) * 2.0 (Weight of two outgoing transitions) = $3 * 2 = 6$
- 1 (1 state with no outgoing transition) * 0 (Weight of zero transition) = $1 * 0 = 0$

$$\text{OIC} = 9 + 6 = 15$$

iii) The Total Interaction Complexity (TIC)

$$\text{TIC} = 8.8 + 15 = 23.8$$

Order Processing Statechart Diagram

The statechart diagram in Figure 6.8 illustrates how the system handles the various steps involved in processing and delivering an order. It captures the sequence of actions taken, from placing the order to completing delivery, showing how the system moves between different states based on specific events or triggers.

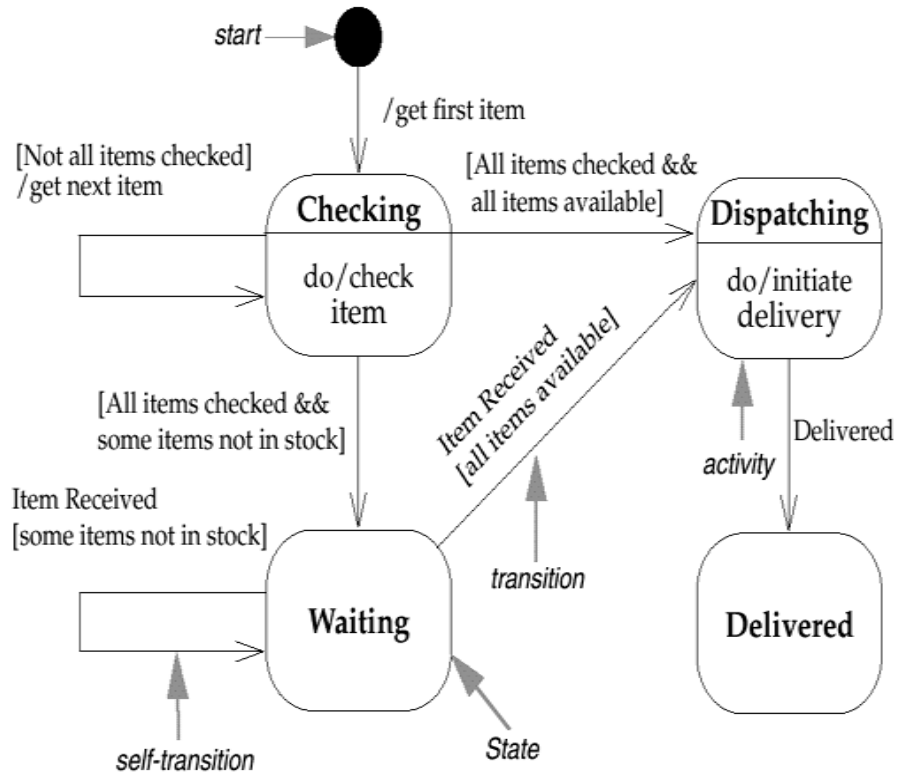


Figure 6.8: Order Processing Statechart Diagram [114]

i) Calculating WNSA

The statechart diagram presented in Figure 6.8, the order processing model includes four basic states. Among them, two states do not perform any activities, while the remaining two each execute a single activity. Additionally, the diagram includes one final state. As a result, the value for the Weighted Number of States and Activities (WNSA) is calculated as follows:

$$WNSA = \prod_{i=1}^n (W_i X_i A_w)$$

Substituting the formulae as follows:

- 2 (simple states with no activities) * 1.5 (weight of simple states) * 1 (weight of state with no activity) = 2 * 1.5 * 1.0 = 3
- 2 (simple states with 1 activity) * 1.5 (weight of simple states) * 1.2 (weight of state with one activity) = 2 * 1.5 * 1.2 = 3.6
- 1 (initial state) * 1.0 (weight of an initial state) = 1 * 1 = 1

$$WNSA = 3 + 3.6 + 1 = 7.6$$

ii) Calculating ACFC

In the order processing model, three transitions do not have any guard conditions, three transitions each with a single guard, and two transitions that include two guard conditions. Furthermore, the diagram contains two looping transitions. Accordingly, the value for ACFC metric is determined as follows:

$$ACFC = \prod_{i=1}^n G_w \sum_{j=1}^m \sum_{k=1}^p \sum_{q=1}^r W_c(ijkq)$$

Substituting the formula as follows:

- 3 (sequences) * 1(weight of the sequence)* 1(weight of sequence with no guard) = 3*1*1= 3
- 3 (sequences) * 1(weight of the sequence)* 1.2 (weight of sequence with one guard) = 3*1*1.2= 3.6
- 2 (sequences) * 1(weight of the sequence)* 1.4 (weight of sequence with two guards) = 2*1*1.4= 2.8
- 2 (loops) * 3 (weight of a loop) = 2*3 = 6

$$ACFC = 3 + 3.6 + 2.8 + 6 = 15.4$$

iii) Calculating TIC

In the context of the order processing system, the diagram contains four states. One of these states is connected by a single incoming transition, while the other three each receive two incoming transitions. Based on these values, TIC can be computed as shown below:

$$TIC = \sum_{t=1}^n (W_t I_t) + \sum_{q=1}^n (W_q O_q)$$

Substituting the formulae as follows:

- i) Calculating Incoming Interaction Complexity (IIC)

- 1 (1 state with one incoming transition) * 1 (Weight of one incoming transition) = $1 * 1.0 = 1.0$
- 3 (3 states with two incoming transitions) * 1.2 (Weight of two incoming transitions) = $3 * 1.2 = 3.6$

$$IIC = 1.0 + 3.6 = 4.6$$

ii) Calculating Outgoing Interaction Complexity (OIC)

In addition, the diagram includes one state that has three outgoing transitions, another state with two outgoing transitions, one state connected by a single outgoing transition, and one state that does not lead to any other state..

- 1 (1 state with one outgoing transition) * 1.5 (Weight of one outgoing transition) = $1 * 1.5 = 1.5$
- 1 (1 state with two outgoing transitions) * 2.0 (Weight of two outgoing transitions) = $1 * 2 = 2.0$
- 1 (1 state with three outgoing transitions) * 2.5 (Weight of three outgoing transitions) = $1 * 2.5 = 2.5$

$$OIC = 1.5 + 2 + 2.5 = 6$$

iii) Calculating Total Interaction Complexity (TIC)

$$TIC = 4.6 + 6 = 10.6$$

6.4.1.2 Scenario II: Activity Diagrams

Questionnaire Processing Activity Diagram

The diagram outlines the steps involved in processing a questionnaire. It starts with user registration, followed by sending the questionnaire. Depending on whether a response is received, the flow either moves through different waiting stages for processing and verification or handles complaints if issues arise. Once the necessary checks are completed, the process ends with archiving the data.

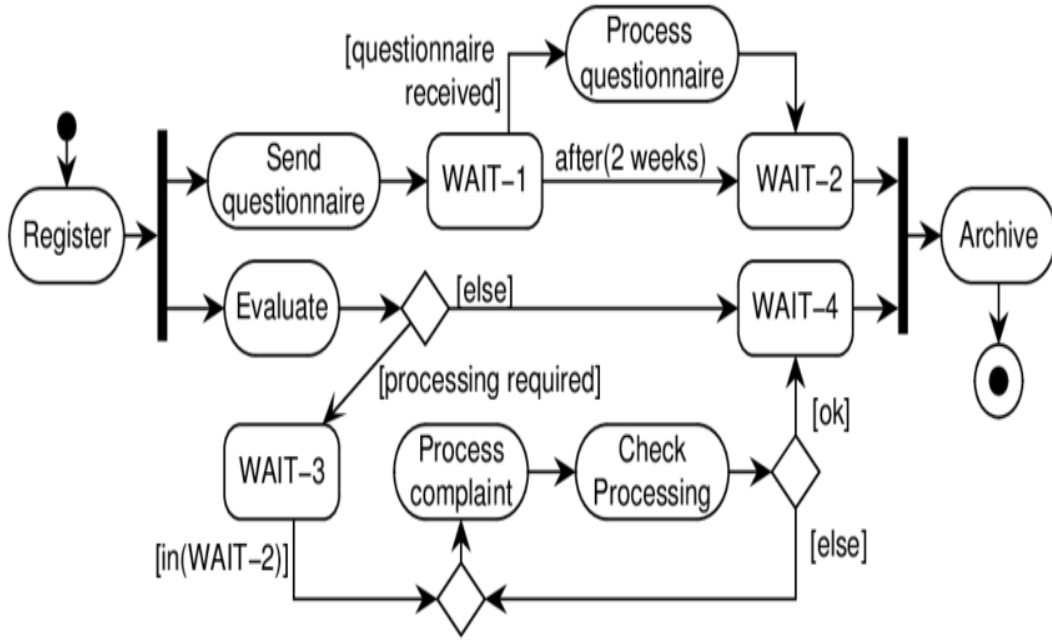


Figure 6.9: Questionnaire Processing Activity Diagram [114]

i) Calculating Adjusted Weighted Number of States (AWNS)

The AWNS metric is calculated as follows:

1. Note the number each category of state i.e. action state, final state and initial state.
2. Get the AWNS by multiply each category of state by its corresponding weight.

$$AWNS = \sum_{U=1}^n (W_U S_U)$$

In the questionnaire processing activity diagram, there are a total of 11 action states, along with one initial state and one final state. These values are then applied to the relevant formula as shown below:

- 1 (initial state) * 1 (weight of an initial state) = 1*1 = 1
- 1 (final state) * 1 (weight of a final state) = 1*1 = 1
- 11 (action states) * 2 (weight of an action state) = 11*2 = 22

$$AWNS=22+1+1=24$$

ii) Calculating Aggregate Control flow complexity (ACFC)

The questionnaire processing system, ten transitions occur without any guard conditions, and six transitions that include one guard. The diagram also features three decision points, one loop structure, and two paths that execute in parallel. The following steps are used to compute the ACFC:

1. Note the number of each category of control flow i.e. the sequence, decision, loop, and parallel.
2. For each category of sequence, count the number of guards inside it.
3. Get the ACFC complexity value by multiplying the type of each control flow with its corresponding weight.

$$ACFC = \prod_{i=1}^n G_w \sum_{j=1}^m \sum_{k=1}^p \sum_{q=1}^r W_c(ijkq)$$

Substituting the formula as follows:

- 10 (sequences) * 1(weight of the sequence)* 1(weight of sequence with no guard) = 10*1*1= 10
- 6 (sequences) * 1(weight of the sequence)* 1.2 (weight of sequence with one guard) =6*1*1.2= 7.2
- 3 (decisions) * 2 (weight of a decision) = 3*2 = 6
- 1 (loop) * 3 (weight of a loop) = 1*3 = 3
- 2 (parallels) * 4 (weight of a parallel) = 2*4 = 8

$$ACFC= 10+7.2+6+3+8=34.2$$

iii) Calculating Total Interaction Complexity (TIC)

To calculate the TIC metric, the following steps are followed:

1. For each object or action state, count the number of incoming transitions and multiply by its corresponding weight to get the Incoming Interaction Complexity (IIC).

2. For each object or action state, count the number of outgoing transitions and multiply by its corresponding weight to get the Outgoing Interaction Complexity (OIC).
3. Get the TIC through summation of IIC and OIC

$$TIC = \sum_{t=1}^n (W_t I_t) + \sum_{q=1}^n (W_q O_q)$$

In the questionnaire processing system, a total of 11 action states are present. Among these, 4 states each receive one incoming transition, while the remaining 7 states have two incoming transitions each. These values are then applied to the appropriate formula as shown below:

- i) Calculating Incoming Interaction Complexity (IIC)
 - 4 (4 action states with one incoming edge) * 1 (Weight of one incoming edge) = 4*1.0= 4.0
 - 7 (7 action states with two incoming edge) * 1.2 (Weight of two incoming edges) = 7 *1.2= 8.4

$$IIC = 4.0 + 8.4 = 12.4$$

Additionally, the diagram shows that 4 action states each have two outgoing transitions, while the other 7 states are connected to only one outgoing transition.

- ii) Calculating Outgoing Interaction Complexity (OIC)
 - 7 (7 action states with one outgoing edges) * 1.5 (Weight of one outgoing transition) = 7*1.5= 10.5
 - 4 (4 action states with two outgoing edges) * 2.0 (Weight of two outgoing transitions) = 4 *2.0= 8.0

$$OIC = 10.5 + 8 = 18.5$$

- iii) The Total Interaction Complexity (TIC)

$$TIC = 12.4 + 18.5 = 30.9$$

Quadratic Roots Activity Diagram

The activity diagram presented in Figure 6.10 is used to represent the logical flow for solving a quadratic equation, showing the steps followed to compute its roots. The behavior and transitions between activities are structured to reflect the calculation process.

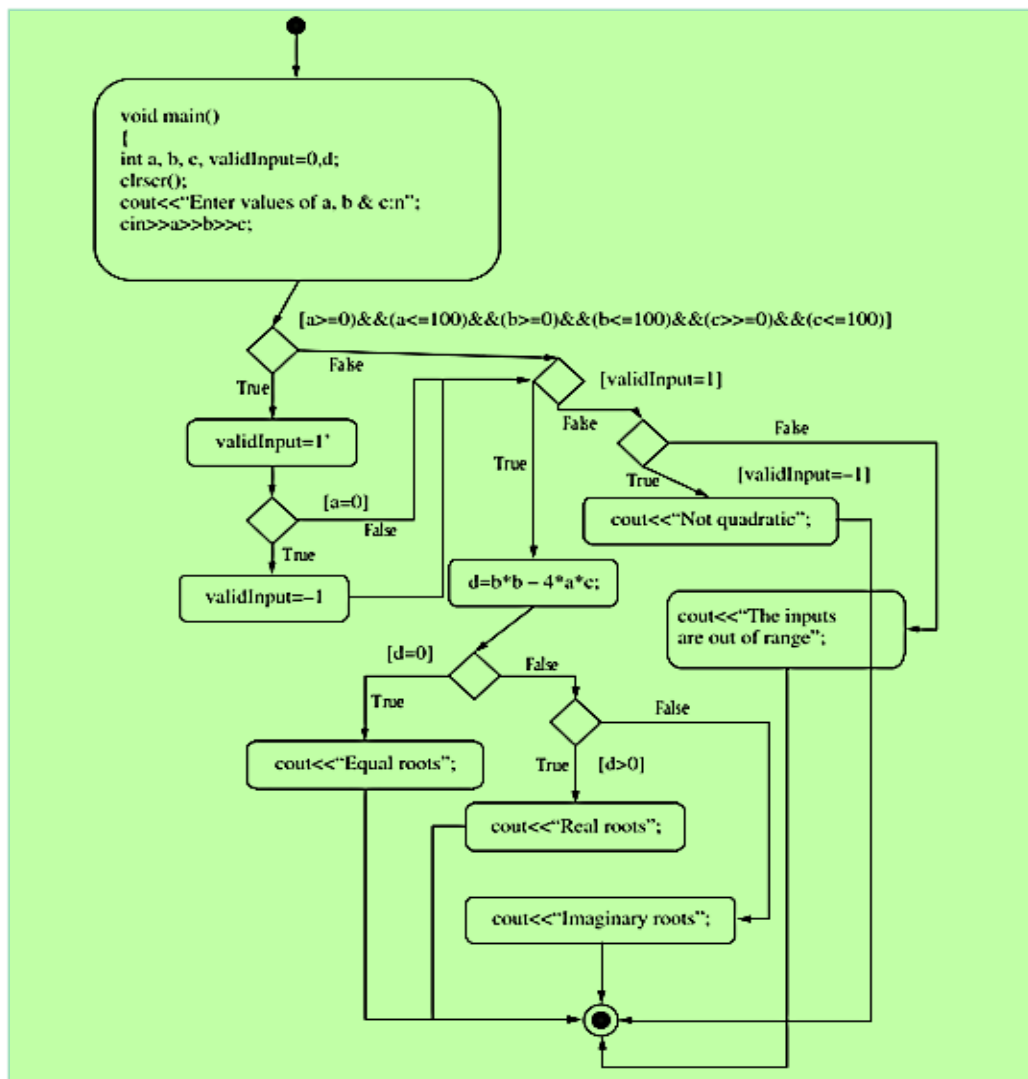


Figure 6.10: Quadratic Roots Activity Diagram [114]

i) Calculating Adjusted Weighted Number of States (AWNS)

The quadratic roots activity diagram consists of eight distinct action states, along with one initial state and one final state. Based on this structure, the value for the AWNS is computed as follows:

$$AWNS = \sum_{U=1}^n (W_U S_U)$$

Substituting the formula as follows:

- 1 (initial state) * 1 (weight of an initial state) = 1*1 = 1
- 1 (final state) * 1 (weight of a final state) = 1*1 = 1
- 8 (action states) * 2 (weight of an action state) = 8*2 = 16

$$AWNS=1+1+16=18$$

ii) Calculating Aggregate Control Flow Complexity (ACFC)

In the activity diagram used for solving a quadratic equation, nine edges do not include guard conditions, five edges each have one guard, and one edge that contains six guard conditions. Additionally, the diagram includes six decision nodes. These values are then applied to the appropriate formula as shown:

$$ACFC = \prod_{i=1}^n G_w \sum_{j=1}^m \sum_{k=1}^p \sum_{q=1}^r W_c(ijkq)$$

Substituting the formula as follows:

- 9 (9 sequences) * 1(weight of the sequence)* 1(weight of sequence with no guard) = 9*1*1= 9
- 5 (sequences) * 1(weight of the sequence)* 1.2 (weight of sequence with one guard) =5*1*1.2= 6.0
- 1 (sequence) * 1(weight of the sequence)* 2.2 (weight of sequence with six guards) =1*1*2.2= 2.2
- 6 (decisions) * 2 (weight of a decision) = 6*2 = 12

$$ACFC= 9+6+2.2+12=29.2$$

iii) Calculating Total Interaction Complexity (TIC)

In the quadratic root activity diagram, there are nine action states in total. Seven action states each have a single incoming edge, while the remaining two action states are

connected by three incoming edges. These figures are then used in the formula as outlined below:

$$TIC = \sum_{t=1}^n (W_t I_t) + \sum_{q=1}^n (W_q O_q)$$

i) Calculating Incoming Interaction Complexity (IIC)

- 7 (7 action states with one incoming edge) * 1 (Weight of one incoming edge) = 7*1.0= 7.0
- 2 (2 action states with three incoming edge) * 1.4(Weight of three incoming edges) = 2*1.4 = 2.8

$$IIC = 7 + 2.8 = 9.8$$

In addition, five action states have a single outgoing edge, two states are linked by three outgoing edges, and the remaining two states each have four outgoing edges. OIC is computed as follows:

ii) Calculating Outgoing Interaction Complexity (IIC)

- 5 (5 action states with one outgoing edges) * 1.5 (Weight of one outgoing transition) = 5*1.5= 7.5
- 2(2 action states with three outgoing edges) * 2.5 (Weight of three outgoing transitions) =2 *2.5= 5.0
- 2(2 action states with four outgoing edges) * 3.0 (Weight of three outgoing transitions) =2 *3.0= 6.0

$$OIC = 7.5 + 5.0 + 6.0 = 18.5$$

iii) The Total Interaction Complexity (TIC)

$$AIC = 9.8 + 18.5 = 28.3$$

6.4.1.3 Scenario III: Sequence Diagrams

Timer Functionality Sequence Diagram

The sequence diagram in Figure 6.11 represents how different parts of the system interact during a communication check. It starts with a request from the user, which goes through the RS232 interface to the Signal Generator. The system accepts the response if the Signal Generator replies within 250 milliseconds. If no reply is received in that time, a timer handles the timeout situation.

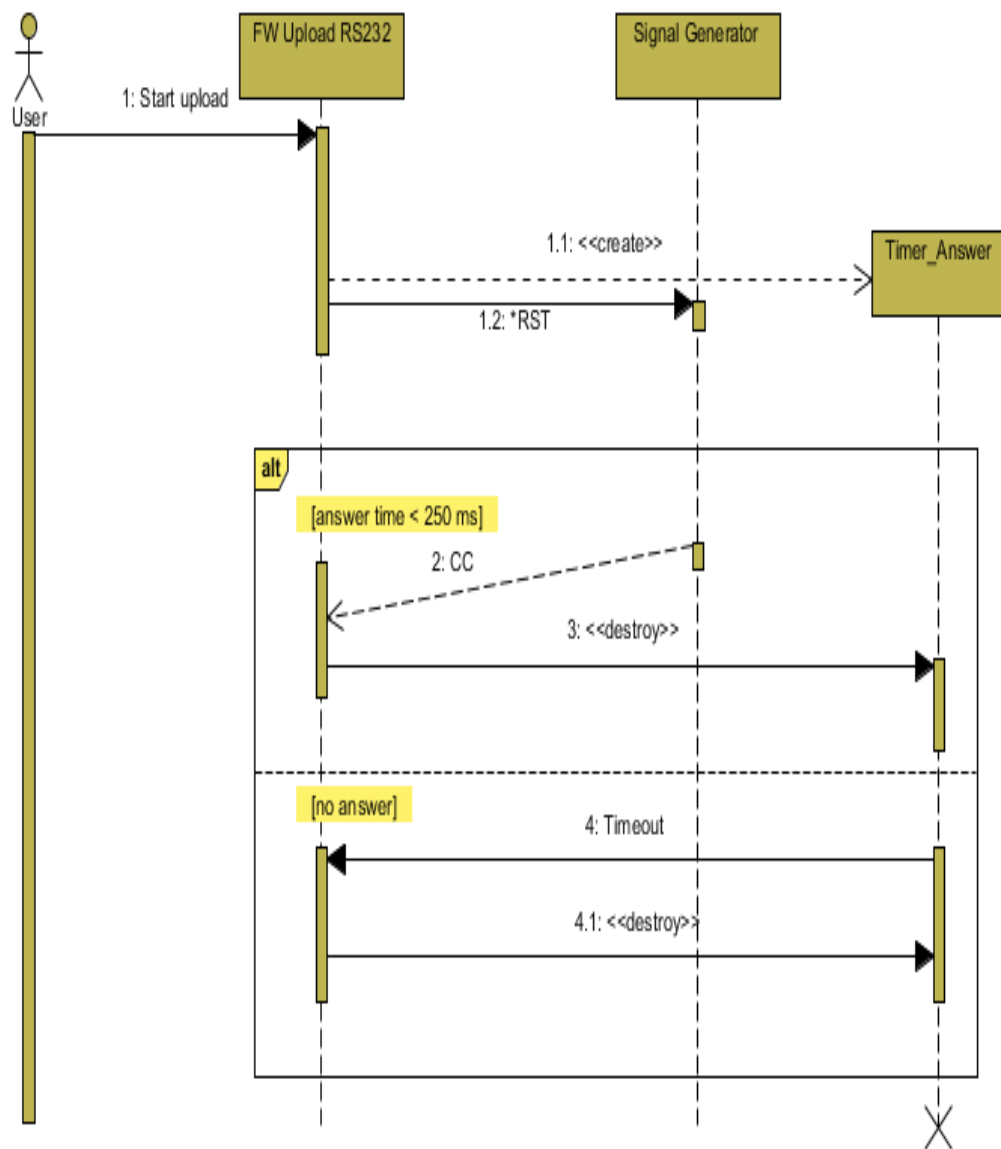


Figure 6.11: Timer Functionality Sequence Diagram [114]

i) Calculating Total Sequence Complexity (TSC)

Total sequence complexity metric is a function of AWMIL and AWMOL. AWMIL is defined as follows:

$$AWMIL = \prod_{j=1}^n (W_j I_j L_w)$$

AWMOL is defined as follows:

$$AWMOL = \prod_{k=1}^n (W_k O_k L_w)$$

Total Sequence Complexity is computed as follows:

$$TSC = \left(\prod_{j=1}^n (W_j I_j L_w) \right) * \left(\prod_{k=1}^n (W_k O_k L_w) \right)^2$$

The sequence diagram in Figure 6.11 shows three lifelines, including both asynchronous message calls and return message. These components are then used in the formula as shown below:

i) Calculating AWMIL

Lifeline 1

- 1(1 message) *1.2(weight of message)*1.0 (number of lifelines transmitted over) = 1*1.2*1.0= 1.2
- 1(1 message) *1.4(weight of message)*1.0 (number of lifelines transmitted over) =1*1.4*1.0=1.4
- 1(1 message) *1.4 (weight of message)*1.2 (number of lifelines transmitted over)=1*1.4*1.2=1.68

Lifeline 2

- $3 (3 \text{ messages}) * 1.4 (\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 3 * 1.4 * 1.0 = 4.2$

Lifeline 3

- $1 (1 \text{ message}) * 1.4 (\text{weight of message}) * 1.2 (\text{number of lifelines transmitted over}) = 1 * 1.4 * 1.2 = 5.04$

$$AWMIL = 1.2 + 1.4 + 1.68 + 4.2 + 5.04 = 13.52$$

ii) Calculating AWMOL

Lifeline 1

- $3 (3 \text{ messages}) * 1.4 (\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 3 * 1.4 * 1.0 = 4.2$
- $3 (3 \text{ messages}) * 1.4 (\text{weight of message}) * 1.2 (\text{number of lifelines transmitted over}) = 3 * 1.4 * 1.2 = 5.04$
- $1 (1 \text{ message}) * 1.4 (\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 1 * 1.4 * 1.0 = 1.4$

Lifeline 2

- $1 (1 \text{ message}) * 1.4 (\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 1 * 1.4 * 1.0 = 1.4$

Lifeline 3

- $1 (1 \text{ message}) * 1.4 (\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 1 * 1.4 * 1.0 = 1.4$

$$AWMOL = 4.2 + 5.04 + 1.2 + 1.4 + 1.4 = 13.24$$

iii) Calculating TSC

$$TSC = (13.52 * 13.24)^2 = 32042.72$$

ii) Calculating Aggregate Control Flow Complexity (ACFC)

The sequence diagram for timer functionality contains six messages without any guard conditions and three that each include a single guard. It also features one decision, a loop interaction, and one instance of parallel processing. These values are then applied to the formula as follows:

$$ACFC = \prod_{i=1}^n G_w \sum_{j=1}^m \sum_{k=1}^p \sum_{q=1}^r W_c(ijkq)$$

Substituting the formula as follows:

- 6 (sequences) * 1(weight of the sequence)* 1(weight of sequence with no guard) = 6*1*1=6
- 3 (sequences) * 1(weight of the sequence)* 1.2 (weight of sequence with one guard) =3*1*1.2=3.6
- 1 (decision) * 2 (weight of a decision) = 2
- 1 (loop) * 3 (weight of a loop) = 3
- 1(parallel) * 4 (weight of a parallel) = 1*4=4

$$ACFC = 6 + 3.6 + 2 + 3 + 4 = 18.6$$

iii) Calculating Total Interaction Complexity (TIC)

In the timer functionality system, the diagram includes three lifelines. The bidder lifeline sends out seven messages, while each of the supplier 1 and supplier 2 lifelines sends out one message. On the receiving end, the bidder lifeline gets three messages, and both supplier 1 and supplier 2 receive two messages each. The values are used to compute TIC as follows

$$TIC = \sum_{t=1}^n (W_t I_t) + \sum_{q=1}^n (W_q O_q)$$

Substituting the formulae as follows:

i) Calculating Incoming Interaction Complexity (IIC)

- 3 (three incoming messages) * 1.4 (Weight of one incoming message) = $3 * 1.4 = 4.2$
- 2 (2 incoming messages) * 1.2 (Weight of two incoming transitions) = $2 * 1.2 = 2.4$

$$IIC = 4.2 + 2.4 = 6.6$$

ii) Calculating Outgoing Interaction Complexity (OIC)

- 7 (7 outgoing messages) * 4.5 (Weight of one outgoing transition) = $7 * 4.5 = 31.5$
- 2 (two outgoing messages) * 2.0 (Weight of two outgoing transitions) = $2 * 2 = 4$

$$OIC = 31.5 + 4.0 = 35.5$$

iii) The Aggregate Interaction Complexity (AIC)

$$TIC = 6.6 + 35.5 = 42.1$$

Procurement sequence diagram

The sequence diagram in Figure 6.12 shows a scenario where a bidder sends out a request for proposals (RFP) to two suppliers. Both supplier1 and supplier2 respond with their proposals. This happens within a loop, which means the process can repeat if the received proposals are not acceptable.

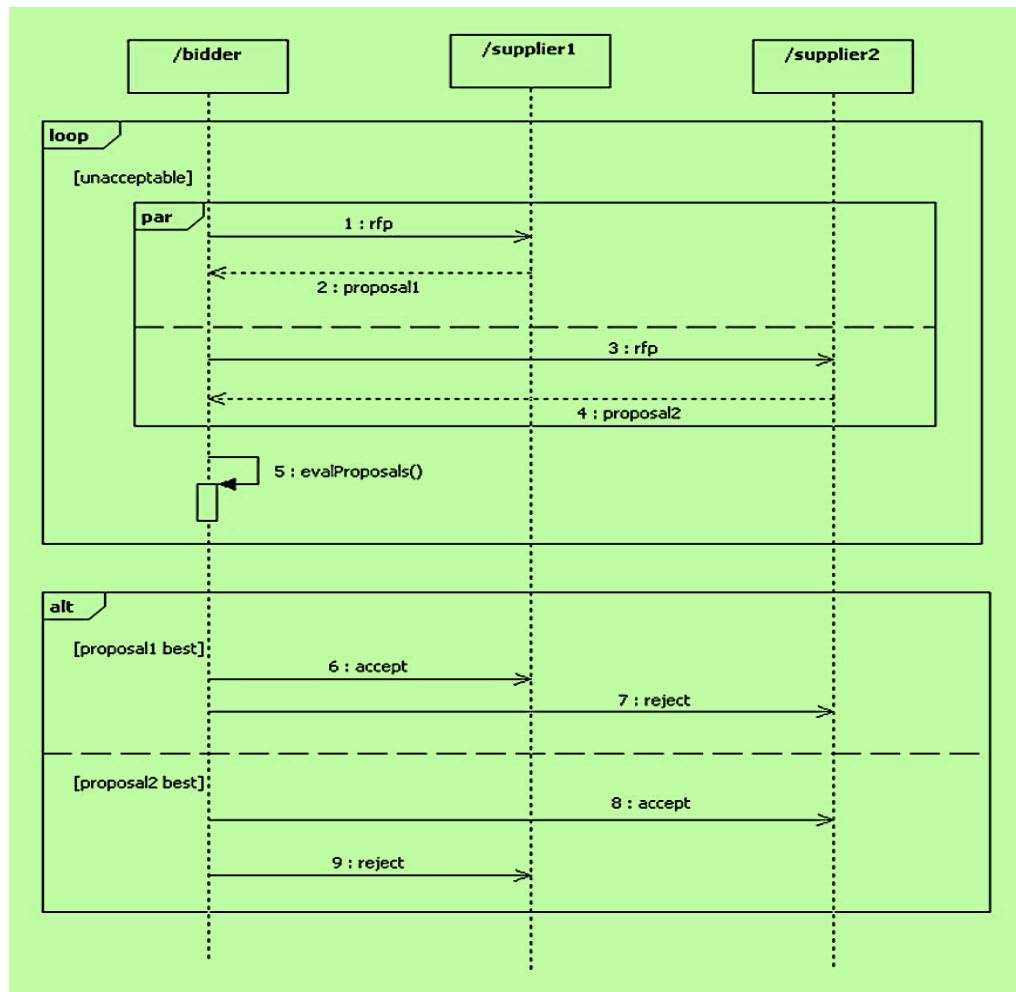


Figure 6.12: Procurement Process Sequence [114]

The complexity values are:

- i) Calculating AWMIL metric

Lifeline 1

- $1 (1 \text{ message1}) * 1.0 (\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 1 * 1.0 * 1.0 = 1.0$
- $1 (1 \text{ message1}) * 1.2 (\text{weight of message}) * 1.2 (\text{number of lifelines transmitted over}) = 1 * 1.2 * 1.2 = 1.44$

Lifeline 2

- $1 (1 \text{ message}) * 1.2(\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 1 * 1.2 * 1.1 = 1.2$

Lifeline 3

$$1 * 1.4 * 1.2 = 1.68$$

- $1 (1 \text{ message}) * 1.4(\text{weight of message}) * 1.2 (\text{number of lifelines transmitted over}) = 1 * 1.4 * 1.2 = 1.68$
- $2(2 \text{ messages}) * 1.2(\text{weight of message}) * 1.2 (\text{number of lifelines transmitted over}) = 2 * 1.2 * 1.2 = 2.88$

$$\text{AWMIL} = 1.0 + 1.44 + 1.2 + 1.68 + 2.88 = 8.2$$

ii) Calculating AWMOL metric

Lifeline 1

- $1 (1 \text{ message}) * 1.4(\text{weight of message}) * 1.2 (\text{number of lifelines transmitted over}) = 1 * 1.4 * 1.2 = 1.68$
- $1 (1 \text{ message}) * 1.2(\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 1 * 1.2 * 1.0 = 1.2$
- $2(2 \text{ messages}) * 1.2(\text{weight of message}) * 1.2 (\text{number of lifelines transmitted over}) = 2 * 1.2 * 1.2 = 2.88$

Lifeline 2

- $1 (1 \text{ message}) * 1.0(\text{weight of message}) * 1.0 (\text{number of lifelines transmitted over}) = 1 * 1.0 * 1.0 = 1.0$

Lifeline 3

$$1 * 1.2 * 1.2 = 1.44$$

- $1(1 \text{ message}) * 1.2(\text{weight of message}) * 1.2 (\text{number of lifelines transmitted over}) = 1 * 1.2 * 1.2 = 1.44$

$$AWMOL = 1.68 + 1.2 + 2.88 + 1.0 + 1.44 = 8.2$$

iii) Calculating TSC

$$TSC = (8.2 * 8.2)^2 = 4521.22$$

ii) **Calculating Aggregate Control Flow Complexity (ACFC)**

The procurement sequence diagram includes four message exchanges without any guard conditions and two that each involve one guard. It also features a single control flow used for decision-making. These information is used to calculate ACFC as follows:

$$ACFC = \prod_{i=1}^n G_w \sum_{j=1}^m \sum_{k=1}^p \sum_{q=1}^r W_c(ijkq)$$

Substituting the formula as follows:

- 4 (sequences) * 1(weight of the sequence)* 1(weight of sequence with no guard) = 4*1*1=4
- 2 (sequences) * 1(weight of the sequence)* 1.2 (weight of sequence with one guard) = 2*1*1.2=2.4
- 1 (decision) * 2 (weight of a decision) = 2

$$ACFC = 4 + 2.4 + 2 = 8.4$$

iii) **Calculating Total Interaction Complexity (TIC)**

In Figure 6.12, the diagram includes three lifelines representing different participants. The bidder lifeline sends four messages, while each of the supplier1 and supplier2 lifelines sends one message. On the receiving end, the bidder lifeline receives two messages, supplier1 receives one, and supplier2 receives three. Using these values, the Total Interaction Complexity (TIC) is calculated through the following steps:

$$AIC = \sum_{t=1}^n (W_t I_t) + \sum_{q=1}^n (W_q O_q)$$

Substituting the formulae as follows:

i) Calculating Incoming Interaction Complexity (IIC)

- 2 (2 incoming messages) * 1.2 (Weight of two incoming message)
= 2*1.2= 2.4
- 1 (1 incoming message) * 1.0 (Weight of one incoming messages)
= 1*1.0= 1.0
- 3 (3 incoming messages) * 1.4 (Weight of three messages) =
3*1.4=4.2

$$IIC = 2.4 + 1.0 + 4.2 = 7.6$$

ii) Calculating Outgoing Interaction Complexity (OIC)

- 4 (4 outgoing messages) * 3.0 (Weight of four outgoing transition) = 4*3.0= 12
- 1 (1 outgoing message) * 1.5 (Weight of one message) = 1*1.5 = 1.5
- 1 (1 outgoing message) * 1.5 (Weight of one message) = 1*1.5 = 1.5

$$OIC = 12 + 1.5 + 1.5 = 15$$

iii) The Total Interaction Complexity (TIC)

$$TIC = 7.6 + 15 = 22.6$$

6.4.2 Comparison with Existing Behavioral Metrics

Researchers have introduced metrics to evaluate the complexity of UML behavioral diagrams. However, many of these metrics were developed without following a structured framework, which can affect their reliability and limit their application across different contexts. Additionally, most of the available metrics focus on only one dimension of complexity, which does not provide a complete understanding of the overall complexity present in UML behavioral models. Also, some are tailored to

specific diagram types such as sequence or statechart diagrams making them less applicable to other UML behavioral representations.

The metrics proposed in this study were designed to address these limitations by targeting the overall complexity across multiple UML behavioral diagram types. They incorporate shared characteristics found in behavioral diagrams, which allows for broader applicability. This makes the proposed measures more versatile compared to existing ones.

In addition, unlike earlier metrics that assess only one aspect, such as size or structure, the proposed set accounts for several dimensions of complexity. These include element complexity, control flow structure, and interaction behavior. The combination of these aspects in metrics offers a more comprehensive and balanced assessment of behavioral diagram complexity.

Furthermore, the development of these metrics followed a formal validation process. Many existing metrics lack proper validation, which affects their acceptance and usefulness. In contrast, the metrics introduced in this study were evaluated using both Weyuker's properties and a perspective-based metrics validation framework. This dual validation approach strengthens their theoretical foundation and ensures they are practically useful. This comparison is presented in Table 6.8.

Table 6.8: Comparison Results

Metrics	Type of Metrics	Multiple Diagrams Measured	Multiple Perspectives Measured
Statechart metrics [93]	Size and complexity	No	No
Statechart metrics [23]	Complexity	No	No
Statechart metrics [87]	Complexity	No	No
Sequence metrics [86]	Complexity	No	No
Kim & Boldyreff [26] metrics	Size and complexity	No	No
Proposed perspective based Metrics	Complexity	Yes	Yes

6.4.3 Theoretical Validation Results

Software metrics should be based on a strong theoretical and mathematical foundation for acceptance by software engineers. Weyuker's properties have been used by a number of researchers to assess software complexity metrics. Weyuker's properties are widely recognized in software metrics research for assessing theoretical, while the PBMVF framework offers a more detailed and focused evaluation by considering the structural and behavioral elements specific to UML diagrams.

Weyuker's properties and the proposed perspective-based Metrics Validation Framework (PBMVF) were utilized to validate the proposed metrics. Weyuker's properties were applied to establish their theoretical soundness, while the PBCF provides a more comprehensive evaluation of UML behavioral metrics.

6.4.3.1 Validation with Weyuker's Properties

Property 1: Noncoarseness

This property requires that any valid metric should assign different complexity values to two behavioral diagrams that are structurally different. The proposed metrics return

unique values for diagrams that are not alike. As a result, all the metrics satisfy this condition.

Property 2: Granularity

Weyuker's second property indicates that a modification in a behavioral diagram should lead to a corresponding change in its complexity score. The proposed metrics namely WNSA, AWMIL, AWMOL, TSC, AWNS, ACFC, IIC, OIC, and TIC reflect changes in complexity values when variations are made to elements, control flow structures, or interactions. Therefore, this property is satisfied.

Property 3: Nonuniqueness

This property accepts that two diagrams, labeled P and Q, may be different in structure but yield the same complexity value, particularly when they contain similar types and numbers of elements but differ only in naming. The proposed metrics respect this property because they only return different values when the actual types and arrangements of elements, flows, or interactions are modified.

Property 4: Importance of Design Features

The property argues that two diagrams might contain an equal number of elements, but have varying complexity due to differences in the types of those elements, their control flow, or interaction logic. All the proposed metrics meet this property as they are sensitive to such changes and adjust their complexity scores accordingly.

Property 5: Monotonicity

This property argues that combining two diagrams should result in a higher overall complexity than the original diagrams considered separately. The proposed metrics produce numerical outputs and therefore support this logic. Hence, they meet this property.

Property 6: Interaction Nonequivalence

This property checks whether merging a third diagram with two others results in a new complexity level due to added interactions. The defined metrics do not satisfy this

property because they assign fixed weights to elements, interactions, and control structures, which prevents them from fully reflecting the interaction effect in such combinations.

Property 7: Permutation Sensitivity

The seventh property states that rearranging elements, control flows, or interactions can affect the complexity of a behavioral diagram. Since the proposed metrics use constant weights for their components regardless of order, this property is not satisfied.

Property 8: Naming Irrelevance

According to this property, diagrams that differ only in their labels or names should be treated as equal in complexity. The proposed metrics produce numeric values independent of diagram naming, thus satisfying this condition.

Property 9: Internal Interaction Impact

This property notes that internal interactions within a diagram can increase its complexity. When additional transitions, states, or events are added to a statechart diagram, all the proposed metrics return higher values, demonstrating an increase in complexity. Therefore, this property is satisfied by all metrics

Table 6.9: Evaluation using Weyuker's Properties

Property	1	2	3	4	5	6	7	8	9
WNSA	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
AWMIL	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
AWMOL	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
SC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
AWNS	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
ACFC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
IIC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
OIC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
AIC	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes

Key: Yes = satisfied property; No = dissatisfied property.

6.4.3.2 Validation with the PBMVF Framework

Element

Property 1 (Type of element): Is the metric measuring the complexity due to the presence of different types of elements in the diagram?

A reliable metric should be able to measure the components that make up a diagram. Element complexity metrics focus on the difficulty introduced by the presence of specific elements within UML behavioral diagrams. For example, the Weighted Number of States and Activities (WNSA) captures the complexity arising from states and activities in a statechart diagram. In sequence diagrams, Adjusted Weighted Message into Lifeline (AWMIL) and Adjusted Weighted Message out of Lifeline (AWMOL) reflect the complexity caused by incoming and outgoing messages. Similarly, in activity diagrams, Adjusted Weighted Number of States (AWNS) evaluates complexity based on the number of action states.

Property 2 (Extent of the element): Is the metric measuring the extent of elements in the diagram?

This property examines whether a metric accurately measures how much of each element is present in a diagram. A well-designed element metric should reflect the overall size of a diagram based on its structural components. The WNSA metric captures the total number of different types of states in a statechart diagram. AWMIL and AWMOL return the total number of categorized messages in a sequence diagram, while AWNS provides the count of action states in an activity diagram.

Property 3 (Weighted element complexity): Does the metric measure the complexity derived from the extent and complexity factor of an element?

The weighted complexity of a UML diagram is the product of the weight of and extent of an element. Therefore, a good metric should be based on the presence of elements together with their weights. The WNSA, AWMIL, AWMOL, and WNSA return complexity values based on the category of elements and their assigned weights.

As seen in Table 6.10, all the element complexity metrics satisfied the element properties as specified in the proposed framework.

Table 6.10: Summary of Validation Results of Element Metrics.

Property	WNSA	AWMIL	AWMOL	TSC	AWNS
Property 1	Yes	Yes	Yes	Yes	Yes
Property 2	Yes	Yes	Yes	Yes	Yes
Property 3	Yes	Yes	Yes	Yes	Yes

Control Flow

Property 1 (Type of control flow): Does the metric measure the complexity due to the existence of categories of control flows in a diagram?

A good metric should evaluate the different types of control flows in a diagram. The aggregate control flow complexity metric (ACFC) can assess complexity due to the presence of a sequence, decision, loop, and parallel in a diagram.

Property 2 (Extent of the control flow): Does the metric assess the extent of guards on a control flow?

A good metric should return the size of the control flow based on the number of guard conditions. The ACFC metric evaluates the complexity of a diagram based on the total number of guards in a control flow.

Property 3 (Depth of the edge): Is the metric measuring the complexity due to the distance covered by an edge?

A good metric should return the distance covered by an edge. The ACFC can evaluate the complexity due to the distance covered by an edge.

Property 4 (Weighted control flow complexity): Does the metric measure the complexity derived from the extent, the distance covered and the complexity factor of the control flow?

A good metric should be based on the presence of control flow together with their assigned weights. ACFC can evaluate complexity due to the presence of control flows together with their assigned complexity weights.

Table 6.11 presents the validation summary of the aggregate control flow complexity metric based on control flow properties.

Table 6.11: Validation Results of Aggregate Control Flow Metric.

Property	ACFC
Property 1	Yes
Property 2	Yes
Property 3	Yes
Property 4	Yes

6.4.3.3 Interaction

Property 1 (Type of interaction): Is the metric measuring the complexity due to the presence of incoming and outgoing interactions?

A valid metric should be able to assess the various forms of interaction present within a diagram. The interaction complexity metrics, namely Incoming Interaction Complexity (IIC), Outgoing Interaction Complexity (OIC), and Total Interaction Complexity (TIC) are designed to measure the complexity resulting from both incoming and outgoing interactions.

Property 2 (Total of incoming and outgoing messages): Does the metric count the total number of incoming and outgoing messages?

A good metric should be based on the total count of incoming and outgoing messages. The IIC, OIC, and TIC metrics evaluate the complexity of a diagram based on the total number of incoming and outgoing messages.

Property 3 (Weighted interaction complexity): Does the metric measure the complexity derived from the incoming and outgoing messages and their complexity factor?

A good metric should be based on the presence of incoming and outgoing messages together with their assigned complexity weights. The IIC, OIC, and TIC can assess

the complexity of a diagram based on the total number of incoming and outgoing messages together with their assigned weights.

As shown in Table 6.12, all the interaction complexity metrics satisfied the interaction properties as specified in the proposed framework.

Table 6.12: Summary of Validation Results of Interaction Metrics

Property	IIC	OIC	TIC
Property 1	Yes	Yes	Yes
Property 2	Yes	Yes	Yes
Property 3	Yes	Yes	Yes

6.5 Discussion

6.5.1 Implications of Results from Real-Life Scenario Calculations

Calculating metric values from different UML behavioral diagrams shows that the proposed metrics are intuitive and align well with expected levels of complexity. The results confirm that the metrics effectively represent the actual complexity found in various types of behavioral diagrams. For example, in statechart diagrams, the metric values derived from the computer gaming diagram are noticeably higher than those from the order processing diagram. This is expected, as the gaming diagram includes more states and control flows, as well as additional features like orthogonal states and loop transitions, which have higher complexity weights compared to other elements.

A similar trend appears in activity diagrams. The questionnaire processing diagram returns higher complexity values than the diagram representing the solution of a quadratic equation. This is due to the larger number of actions, more control flow paths, and higher interaction between components that increase overall complexity and are accurately measured by the proposed metrics.

In the case of sequence diagrams, the Total Sequence Complexity (TSC) is greater in the procurement diagram. This is because of the number and length of message exchanges, which contribute to increased complexity. Likewise, the Aggregate Control Flow Complexity (ACFC) is higher in the timer functionality sequence diagram due to the use of decision nodes, guards, and loops, all of which are assigned higher weights.

The Total Interaction Complexity (TIC) is also higher in the timer functionality sequence diagram, reflecting more intense interaction among objects. These outcomes support the ability of the defined metrics to capture the complexity introduced by structural and behavioral differences across UML diagrams.

6.5.2 Implications of Results from Comparison with Existing Metrics

A review of existing UML metrics shows that the proposed perspective-based metrics offer broader coverage. Unlike traditional metrics, which are usually limited to one type of behavioral diagram, the proposed metrics are designed to apply across all UML behavioral diagram types. This broader applicability is made possible by the perspective-based attribute structuring framework, which takes into account several aspects of complexity. Specifically, it includes element complexity, control flow complexity, and interaction complexity, making the overall evaluation more comprehensive.

6.5.3 Implications of Results from Theoretical Validations

Theoretical validation based on Weyuker's properties shows that the proposed metrics fulfill seven out of the nine criteria, which supports their mathematical validity and reliability. However, they do not meet Property 6 (Interaction Non-Equivalence) and Property 7 (Permutation Sensitivity). This is due to the constant weighting scheme applied to elements, control flows, and interactions, meaning that complexity calculations remain consistent irrespective of their positional rearrangement. While these properties are relevant in certain complexity assessments, their exclusion does not diminish the effectiveness of the proposed metrics for behavioral diagrams. It can be concluded that the proposed metrics are theoretically sound and well-suited for assessing UML behavioral complexity from a perspective-based approach.

The validation of the proposed metrics against Weyuker's properties provides strong theoretical support for their application in UML behavioral diagram complexity assessment. Additionally, the proposed perspective-based complexity metrics offer software engineers, system designers, and researchers a structured and reliable method for evaluating UML behavioral diagram complexity unlike traditional metrics, which often focus on single diagram types.

6.6 Summary

This chapter focused on the development and validation of perspective-based complexity metrics for UML behavioral diagrams, addressing the limitations of existing complexity measures. The proposed metrics incorporate multiple perspectives to provide a comprehensive evaluation of behavioral diagrams. The structured methodology used in defining and validating these metrics ensures both theoretical soundness and practical applicability. The metrics were validated using Weyuke's properties and perspective-based metrics validation framework to test their theoretical soundness and their practical applicability. By solving the gaps in existing metrics, the proposed measures contribute to more accurate, reliable, and comprehensive UML complexity assessments, which lead to improvement of software quality, analyzability, and standardization in behavioral diagram analysis.

CHAPTER SEVEN

DEVELOPMENT OF A BEHAVIORAL COMPLEXITY METRICS TOOL (BCMT)

7.1 Introduction

This section introduces the Behavioral Complexity Metrics Tool (BCMT), which was designed to automatically calculate the complexity values of various behavioral diagrams. The discussion addresses the fourth research objective, which focused on developing a static analysis tool to automate computation of metrics.

7.2 Requirements of BCMT Tool

This section presents both the functional and non-functional requirements needed for the design of the metrics tool.

Functional requirements for the BCMT tool were as follows:

- i. The BCMT should load all extensible markup language (XML) files.
- ii. The tool should recognize all correct XML files required.
- iii. The user should be able to select an option to calculate the metric values.
- iv. The BCMT should deliver the calculated metrics to the end user using various output channels.
- v. The tool should allow the user save the metrics for future reference.
- vi. The tool should allow the user to clear unwanted output from textboxes.
- vii. The tool should allow the user print the calculated metrics.
- viii. Users should be able to exit from the tool
- ix. Users should be able to display the source file and summary of calculated metrics from the tool.

The non-functional requirements of the BCMT tool were as follows:

- i. The BCMT tool should be portable. Moving from one OS to the other should not create any problem.
- ii. The BCMT tool should not have more than 10 bugs/100KLOC.
- iii. 95% of all users shall be satisfied with the usability of the BCMT tool.
- iv. The BCMT tool shall perform metrics computations efficiently.

- v. The BCMT tool shall load and parse XML files smoothly.

7.3 Implemented Metrics

Both the base and the derived metrics need to be implemented in the BCMT tool. For statechart diagrams, the primary metrics include: number simple states (NSS), number of initial states (NIS), number of final states (NFS), number of composite states (NCS), number of orthogonal states (NOS), number of history states (NHS), and number of submachine states (NSMS). Regarding sequence diagrams, the fundamental metrics encompass: number of delay messages (NDM), number of synchronous messages (NSM), number of asynchronous messages (NAM), number of return messages (NRM), and the number of lifelines involved (NLTO). For activity diagrams, the base metrics consist of: the number of action states (NAS), total initial states (TCIS), and total final states (TCFS). The base metrics for control flow complexity were the number of sequence (NS), number of decisions (ND), number of loops (NL), number of parallels (NP) and number of guards (NG). The interaction complexity base metrics were the number of incoming messages (NIM) and number of outgoing messages (NOM).

The derived metrics calculated by the tool were weighted number of states activities (WNSA), total sequence complexity (TSC), adjusted weighted number of state (AWNS), aggregate control flow complexity (ACFC) and total interaction complexity (TIC).

7.4 Structure of Input File

The input file is formatted in XML and represents the underlying structure of UML behavioral diagrams, which are generated using modeling tools like Visual Paradigm or Plant UML. These tools enhance developer efficiency by eliminating the need to manually write XML code for diagram structures.

The structure of the UML-XML source begins with a root node that identifies the diagram type, such as sequence, statechart, or activity. It then includes a declaration block that defines diagram components like states, transitions, messages, events, and actions, depending on the diagram type. This file also contains metadata relevant for behavioral interpretation, such as message exchanges in sequence diagrams, state transitions in statecharts, or action flows in activity diagrams. This structured input

format is processed by the BCMT tool to calculate corresponding complexity metrics. Figure 7.1 represents a UML XML file.

```
<xml version="1.0" encoding="UTF-8"?>
<activityDiagram xmlns="http://www.omg.org/spec/UML/20090901">
<startEvent id="start" name="Start"/>
<activity id="readData" name="Read Data" />
<activity id="generateDiagrams" name="Generate Diagrams" />
<flow id="flow1" source="start" target="readData" />
<flow id="flow2" source="readData" target=" <activity id
generateDiagrams"/>
    <flow id="flow3" source="generateDiagrams" target=" <activity id
        readData"/>
<flow id="flow4" source="generateDiagrams" target="end"/>
<endEvent id="end" name="End"/>
</activityDiagram>
```

Figure 7.1: XML File

7.5 Architectural Design

The software architectural design explains the relationship of different software components. The BCMT tool is composed of three modules namely, the input, analyzer and the output module.

7.5.1 Input Component

The function of the input module is to collect and load an XML from the file repository. The input component can load the XML files hence the user can see the loaded file on the source tab on the interface. The XML file has several features such as the elements of a diagram and control flow structures. Figure 7.4 represents an XML file.

7.5.2 Analyzer Component

Analyzer component plays a crucial role in generating the complexity metrics for UML behavioral diagrams. It operates through a two-phase approach comprising lexical analysis and parsing. In the first phase, the lexical analyzer scans the XML-based input files and identifies relevant tokens necessary for metric computation.

These tokens represent structural elements of the diagrams such as states, messages, or actions. Once extracted, the parser receives these tokens and processes them by counting occurrences based on predefined metric definitions. The parsing procedure is triggered when a user selects the "Calculate" option on the graphical user interface, initiating the automated analysis workflow that results in both base and derived metric values.

7.5.3 Output Component

Output component is the last stage in the metrics computation. It delivers the calculated metric values in various output channels. The output component has functionalities namely, view metrics, print metrics, save metrics and the data storage.

The view metrics enables the user to view the calculated behavioral metrics on the interface while the print metrics allow users to generate hard copies of their calculated values for documentation. The save metrics enables users to save the calculated values either in text or XML format for referencing purposes. The data storage is a repository where behavioral metrics are stored for future references. The architecture of the BCMT tool is represented in Figure 7.2.

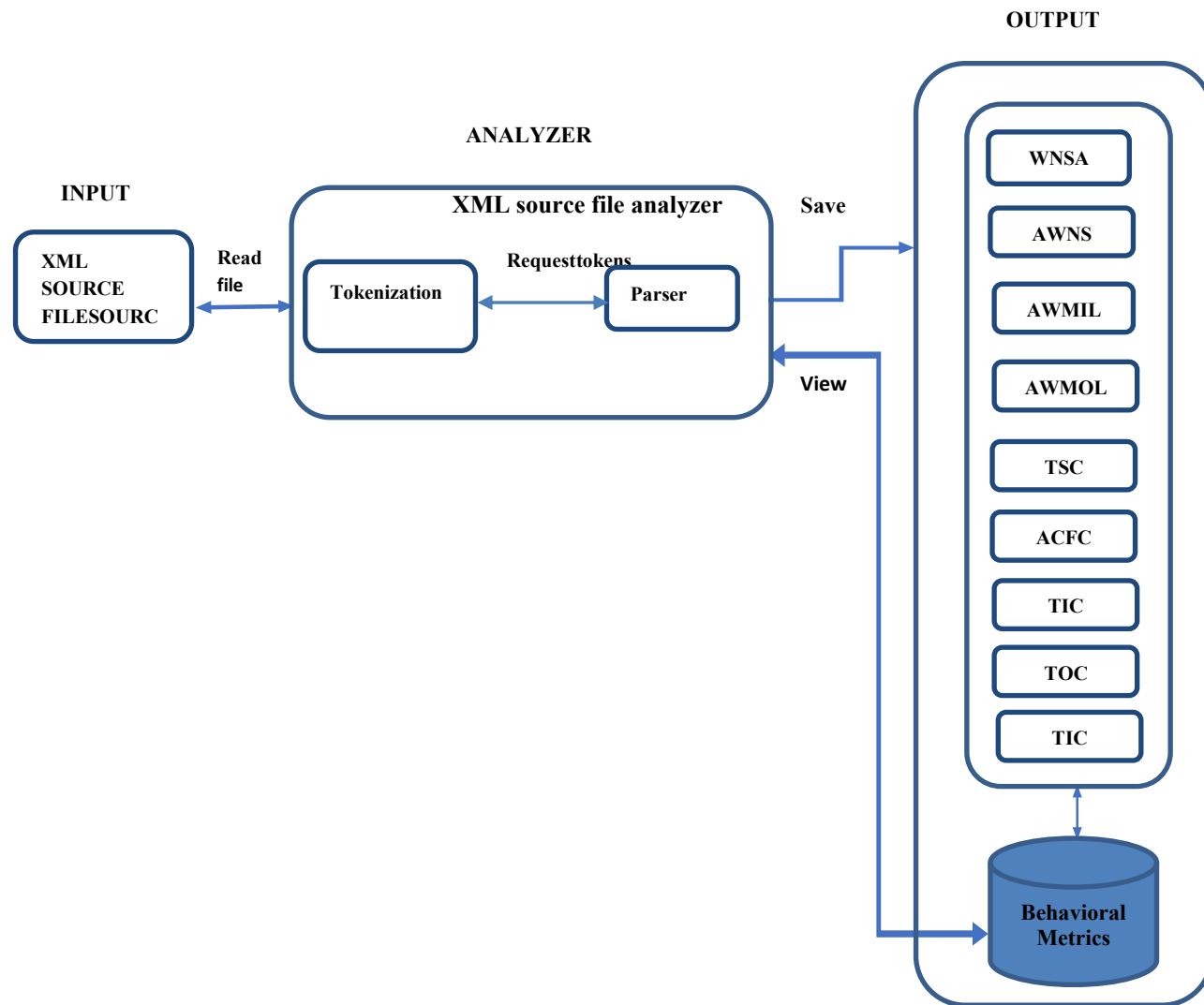


Figure 7.2: Architecture of the BCMT Tool

Once the tool’s architecture has been defined, the subsequent step involves developing the system’s design.

System design outlines the various components that make up the system that facilitate the loading, analysis, and processing of XML files. The Primary component initiates the data flow by reading from the XML source file, sending information to the analyzer, and processing the behavioral metrics. The calculated values are then passed to the output components. The XML Data Source component stores and provides access to XML files containing data for analysis. It interacts with the primary module and analyzer, providing the required tokenization and metric computation data. The analyzer breaks down the XML data into meaningful strings for behavioral metric computation. The behavioral metrics module stores, processes, and manages the calculated behavioral metrics. A structure chart outlining all the system modules, their interactions and functionality is shown in Figure 7.3.

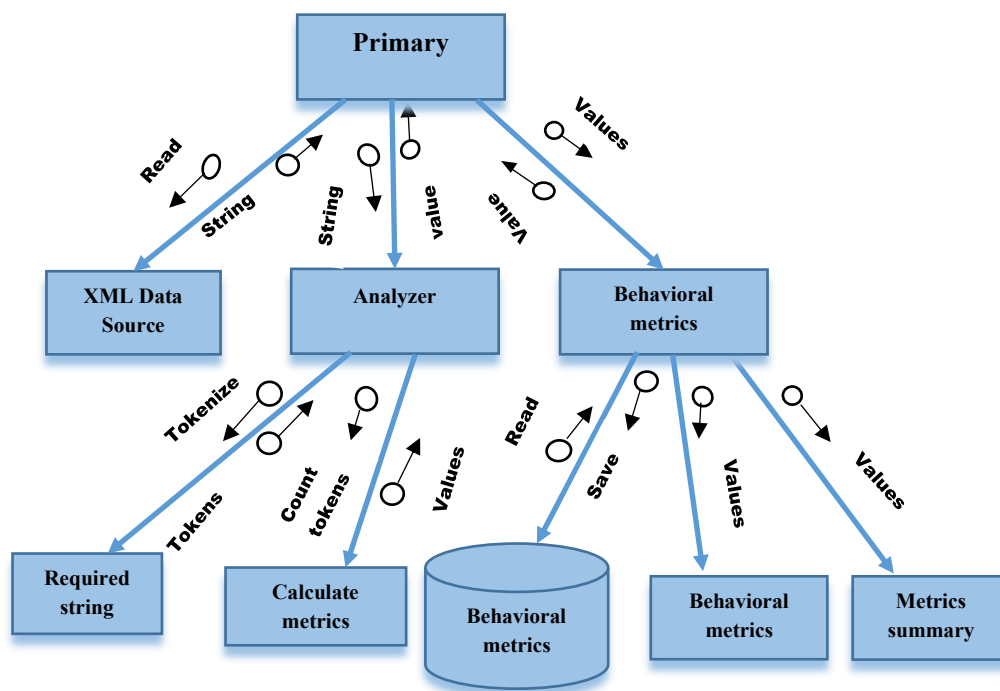


Figure 7.3: BCMT Structure Chart

7.6 User Interface Design

The user interface enables users to load UML diagrams, compute metrics, and view results through options like Open, Calculate, and Metrics. It also allows saving, printing, and exiting the application via menu and toolbar controls.

Form layout design, depicted in Figure 7.4, represents the graphical user interface tool. The menu bar includes File, and Exit options. The File menu provides access to Open, Calculate, Save, Clear, and Print, The Exit menu allows users to quit the application. The standard tool bar has view menu to display the summary of the calculated metrics and the source file while metrics displays the base and derived metrics after computation.

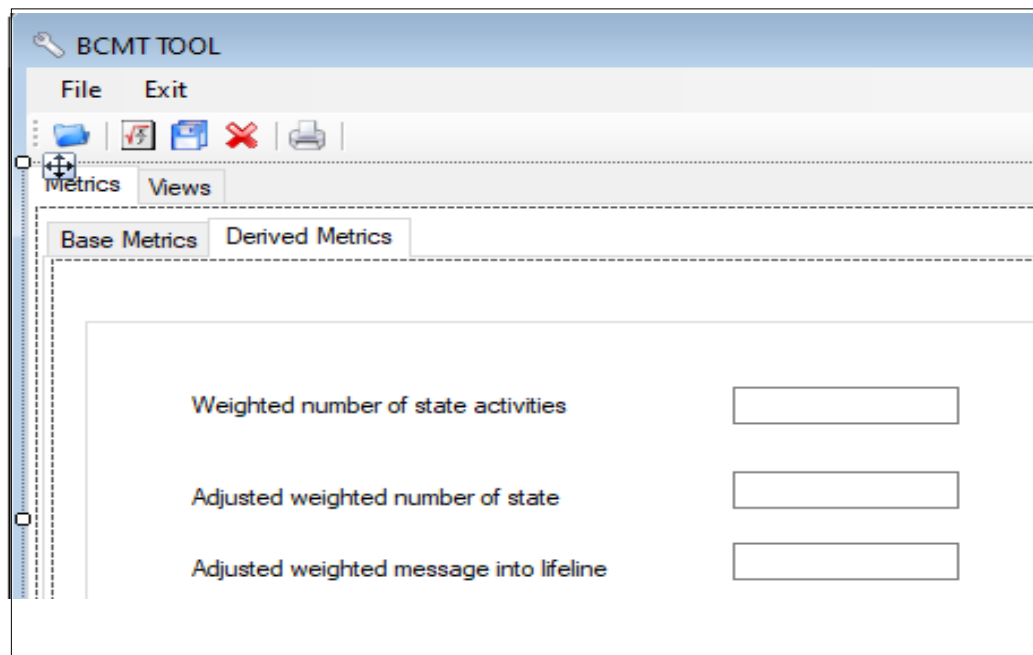


Figure 7.4: User Interface Design

Use case diagrams represent how the user interacts with the BCMT tool. The users invents include load and view XML file, process XML file, calculate metrics, save the metrics and print metric values. Figure 7.5 represents the use case for the BCMT tool.

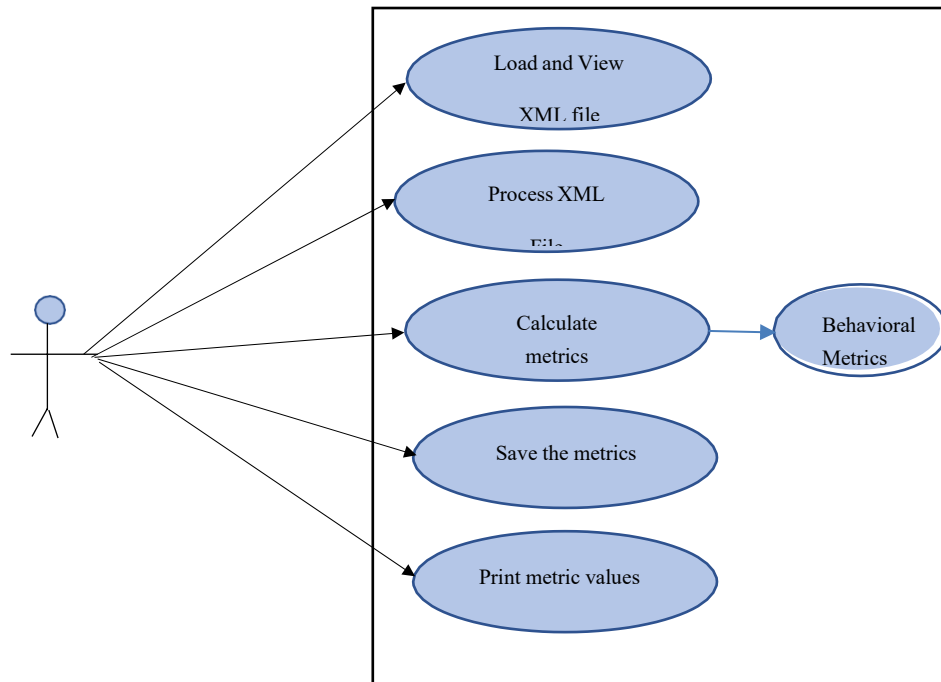


Figure 7.5: Use Case for the System

7.7 Algorithm Design

This section presents algorithms used in the prototype tool for the computation of metrics. It outlines the logic, structure, and functionality of these algorithms, describing how they process input data, perform calculations, and generate meaningful metric values.

7.7.1 Weighted Number of State and Activities (WNSA) Algorithm

7.7.1.1 Number of Simple States (NSS)

To count the NSS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any simple state detected right from the start of a line.
- ii. If a simple state existed, then the flag's status is set to true else its false status remains.
- iii. NSS tally is incremented, else it remains the same.

7.7.1.2 Number Initial States (NIS)

To count the NIS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any initial state detected right from the start of a line.
- ii. If an initial state existed, then the flag's status is set to true else its false status remains.
- iii. NIS tally is incremented, else it remains the same.

7.7.1.3 Number of Final states (NFS)

To count the NFS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any final state detected right from the start of a line.
- ii. If a final state existed, then the flag's status is set to true else its false status remains.
- iii. NFS tally is incremented, else it remains the same.

7.7.1.4 Number of Composite States (NCS)

To count the NCS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any composite state detected right from the start of a line.
- ii. If a composite state existed, then the flag's status is set to true else its false status remains.
- iii. NCS tally is incremented, else it remains the same.

7.7.1.5 Number of History States (NHS)

To count the NHS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any history state detected right from the start of a line.
- ii. If any submachine state exists, then the flag's status is set to true else its false status remains.
- iii. NHS tally is incremented, else it remains the same.

7.7.1.6 Number of Orthogonal States (NOS)

To count the NOS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any orthogonal state detected right from the start of a line.

- ii. If an orthogonal state existed, then the flag's status is set to true else its false status remains.
- iii. NOS tally is incremented, else it remains the same.

7.7.1.7 Number of Submachine States (NSMS)

To count the NSMS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any orthogonal state detected right from the start of a line.
- ii. If submachine state existed, then the flag's status is set to true else its false status remains.
- iii. NOS tally is incremented, else it remains the same.

7.7.1.8 Number of Activities inside a State (NAS)

To count the NAS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any do, entry and exit activity detected in a state right from the start of a line.
- ii. If a do, entry and exit activity existed, then the flag's status is set to true else its false status remains.
- iii. NAS tally is incremented, else it remains the same.

7.7.1.9 To measure Weighted Number of States Activities (WNSA)

To count the WNSA, the following steps are followed:

- i. Find the variables that have values of NSS, NIS, NFS, NCS, NOS, NHS, NSMS and NAS.
- ii. Calculate the product of NSS and its designated complexity factor, then further multiply by the complexity value attributed to states containing actions (NAS).
- iii. Take NIS, multiply it by its defined complexity coefficient, and again by the NAS weight to reflect activity-based complexity.
- iv. Calculate the value by multiplying NFS with its assigned complexity score, followed by a multiplication with the NAS weight.
- v. Derive the result by multiplying NCS with its relevant complexity value, then by the weight associated with active states (NAS).
- vi. Multiply the quantity of NOS with its respective complexity weight and then with the NAS-based activity complexity rating.

- vii. Multiply NHS by its linked complexity factor and further apply the NAS complexity weight to adjust for state activity.
- viii. Obtain the final result by taking NSMS, applying its assigned weight, and then multiplying with the NAS activity complexity rating.
- ix. Obtain the sum total of the new values of NSS, NIS, NFS, NCS, NOS, NHS and NSMS.

7.7.2 Total Sequence Complexity (TSC) Algorithm

7.7.2.1 Number of Delay Messages (NDM)

To count the NDM, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any delay message detected into lifeline or out of lifeline right from the start of a line.
- ii. If a delay message existed, then the flag's status is set to true else its false status remains.
- iii. NDM tally is incremented; else it remains the same.

7.7.2.2 Number of Synchronous Messages (NSM)

To count the NSM, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any synchronous message detected into lifeline or out of lifeline right from the start of a line.
- ii. If a synchronous message existed, then the flag's status is set to true else its false status remains.
- iii. NSM tally is incremented, else it remains the same.

7.7.2.3 Number of Asynchronous Messages (NAM)

To count the NAM, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any asynchronous message detected into lifeline or out of lifeline right from the start of a line.
- ii. If asynchronous message existed, then the flag's status is set to true else its false status remains.
- iii. NAM tally is incremented, else it remains the same.

7.7.2.4 Number of Return Messages (NRM)

To count the NRM, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any return message detected into lifeline or out of lifeline right from the start of a line.
- ii. If a return message existed, then the flag's status is set to true else its false status remains.
- iii. NRM tally is incremented, else it remains the same.

7.7.2.5 Number of Lifelines Transmitted Over (NLTO)

To count the NLTO, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any lifeline detected in a message from the start of a line.
- ii. If a lifeline exists, then the flag's status is set to true else its false status remains.
- iii. NLTO tally is incremented, else it remains the same.

7.7.2.6 Adjusted Weighted Message into Lifeline (AWMIL)

To calculate the AWMIL, the following steps are followed:

- i. Find the variables that have values of NDM, NSM, NAM, and NLTO.
- ii. Multiply the value of NDM into lifeline with its corresponding complexity weight and the complexity weight assigned to the number of lifelines transmitted over (NLTO).
- iii. Multiply the value of NSM into lifeline with its corresponding complexity weight and the complexity weight assigned to the number of lifelines transmitted over (NLTO).
- iv. Multiply the value of NAM into lifeline with its corresponding complexity weight and the complexity weight assigned to the number of lifelines transmitted over (NLTO).
- v. Determine the product of the NRM value per lifeline and its associated complexity factor, then multiply this result by the complexity weight linked to the number of lifelines involved (NLTO).
- vi. Obtain the total of the new values of NDM, NSM, NAM and NRM.

7.7.2.7 Adjusted Weighted Message out of Lifeline (AWMOL)

To calculate the AWMOL, the following steps are followed:

- i. Find the variables that have values of NDM, NSM, NAM, and NLTO.

- ii. Calculate the result by multiplying the number of delay messages (NDM) leaving a lifeline with its assigned complexity factor, then apply the complexity value related to the total lifelines passed through (NLTO).
- iii. Take the count of synchronous messages (NSM) exiting a lifeline, multiply it by its respective complexity rating, and then factor in the complexity weight assigned to NLTO.
- iv. Multiply the total of asynchronous messages (NAM) sent from a lifeline by its designated complexity value, followed by multiplication with the complexity factor linked to NLTO.
- v. Calculate the outcome by taking the return message count (NRM) originating from a lifeline, applying its complexity coefficient, and then multiplying the result by the weight linked to lifeline transitions (NLTO).
- vi. Obtain the total of the new values of NDM, NSM, NAM, and NRM.

7.7.2.8 Total Sequence Complexity (TSC)

To calculate the TSC, the following steps are followed:

- i. Find the variables that have values of AWMIL and AWMOL.
- ii. Multiply the value of AWMIL by AWMOL.
- iii. Square the results obtained in 2 to obtain TSC.

7.7.3 Adjusted Weighted Number of State (AWNS) Algorithm

To calculate the AWNS, at first determine the number of action states (NAS), total count of initial states (TCIS) and total count of final states (TCFS). Calculate the AWNS value by multiplying the type of state with its corresponding weight (W_u).

7.7.3.1 Number of Action States (NAS)

To count the NAS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any action state detected right from the start of a line.
- ii. If an action state existed, then the flag's status is set to true else its false status remains.
- iii. NAS tally is incremented, else it remains the same.

7.7.3.2 Total Count of Initial States (TCIS)

To count the TCIS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any initial state detected right from the start of a line.
- ii. If an initial state existed, then the flag's status is set to true else its false status remains.
- iii. TCIS tally is incremented; else it remains the same.

7.7.3.3 Total Count of Final States (TCFS)

To count the TCFS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any final state detected right from the start of a line.
- ii. If a final state existed, then the flag's status is set to true else its false status remains.
- iii. TCFS tally is incremented, else it remains the same.

7.7.3.4 Weighted Number of States Activities (WNSA)

To calculate the WNSA, the following steps are followed:

- i. Find the variables that have values of NAS, TCIS and TCFS.
- ii. Multiply the value of NAS with its corresponding complexity weight.
- iii. Multiply the value of TCIS with its complexity weight.
- iv. Multiply the value of TCFS with its corresponding complexity weight.
- v. Obtain the total of the new values of NAS, TCIS, and TCFS.

7.7.4 Aggregate Control Flow Complexity (ACFC) Algorithm

To compute the ACFC value, begin by identifying the total sequences (NS), decision points (ND), loop structures (NL), and instances of parallel execution (NP). Further, for each category of sequence, count the number of guards inside it and finally, calculate the ACFC value by multiplying the type of control flow with its corresponding weight (W_c) and the weight of guards inside the sequence (G_w).

7.7.4.1 Number of Sequence (NS)

To count NS, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any sequence detected right from the start of a line.
- ii. If a sequence existed, then the flag's status is set to true else its false status remains.

- iii. NS tally is incremented, else it remains the same.

7.7.4.2 Number of Decision (ND)

To count ND, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any decision detected right from the start of a line.
- ii. If a decision existed, then the flag's status is set to true else its false status remains.
- iii. ND tally is incremented, else it remains the same.

7.7.4.3 Number of Loop (NL)

To count NL, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any loop detected right from the start of a line.
- ii. If a loop existed, then the flag's status is set to true else its false status remains.
- iii. NL tally is incremented, else it remains the same.

7.7.4.4 Number of Parallel (NP)

To count NP, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any parallel detected right from the start of a line.
- ii. If a parallel existed, then the flag's status is set to true else its false status remains.
- iii. NP tally is incremented, else it remains the same.

7.7.4.5 Number of Guards (NG)

To count NG, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any guard detected in a sequence from the start of a line.
- ii. If a guard exists, then the flag's status is set to true else its false status remains.
- iii. NG tally is incremented, else it remains the same.

7.7.4.6 Aggregate Control Flow Complexity (ACFC)

To calculate ACFC, the following steps are followed:

- i. Find the variables that have values of NS, ND, NL and NP.

- ii. To calculate the ACFC value, begin by identifying the total sequences (NS), decision points (ND), loop structures (NL), and instances of parallel execution (NP). Multiply the value of ND with its corresponding complexity weight.
- iii. Multiply the value of NL with its corresponding complexity weight.
- iv. Multiply the value of NP with its corresponding complexity weight.
- v. Obtain the total of the new values of NS, NG, ND, NL and NP.

7.7.5 Total Interaction Complexity (TIC) Algorithm

7.7.5.1 Number of Incoming Messages (NIM)

To count NIM, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any incoming message detected right from the start of a line.
- ii. If an incoming message existed, then the flag's status is set to true else its false status remains.
- iii. NIM tally is incremented, else it remains the same.

7.7.5.2 Number of Outgoing Messages (NOM)

To count NOM, the following steps are followed:

- i. The lexical analyzer should have a flag that indicates any outgoing message detected right from the start of a line.
- ii. If an outgoing message existed, then the flag's status is set to true else its false status remains
- iii. NOM tally is incremented, else it remains the same.

7.7.5.3 Incoming Interaction Complexity (IIC)

To calculate the IIC, the following steps are followed:

- i. Find the variables that have values of NIM.
- ii. Multiply the value of NIM with its corresponding complexity weight.
- iii. Obtain the sum total of the new values of NIM to obtain IIC.

7.7.5.4 Outgoing Interaction Complexity (OIC)

To calculate the OIC, the following steps are followed:

- i. Find the variables that have values of NOM.
- ii. Multiply the value of NOM with its corresponding complexity weight.
- iii. Obtain the sum total of the new values of NOM to obtain OIC.

7.7.5.5 Total Interaction Complexity (TIC)

To calculate the TIC, the following steps are followed:

- i. Find the variables that have values of IIC and OIC.
- ii. Sum the value of IIC and OIC to obtain TIC.

7.8 Implementation and Validation

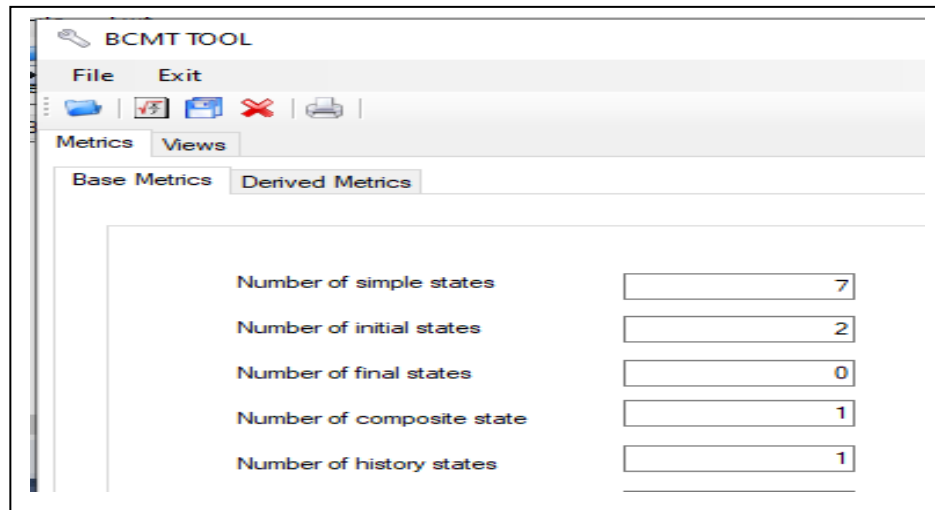
The high-level structure and detailed algorithmic logic previously described were realized using Visual Basic, a visual programming language within the Microsoft Visual Studio environment. Visual Basic was chosen for its ease of use, rapid application development capabilities, and strong support for building graphical user interfaces (GUIs). Its event-driven programming model made it ideal for implementing interactive features such as XML file handling, metric calculation triggers, and dynamic display of results. The language also provided robust integration with XML processing libraries, making it well-suited for parsing and analyzing structured input files. Full source code and implementation details are available in Appendix 4.

The BCMT tool supports the calculation of complexity metrics for UML behavioral diagrams and operates through the following user-guided steps:

- i. The user initiates the process by loading the XML input file, which can be done via the "Open" button, the "File → Open" menu option, or the corresponding icon on the toolbar.
- ii. To perform metric calculations, the user selects the "Calculate Metrics" option.
- iii. Once processing is complete, the output metrics are shown in specified text boxes within the interface.
- iv. The results may be saved by either choosing "File → Save" from the menu or clicking the Save icon; this action stores the output in a plain text format.
- v. For a physical copy, users can print the results using the "File → Print" option or the Print icon.
- vi. To exit the program, the user simply selects the "Exit" button to close the tool.

A validation exercise was carried out focusing on the computation of both base and derived metrics. The objective was to verify the tool's correctness, reliability, and alignment with theoretical definitions.

As shown in Figure 7.6, the base metrics are calculated directly from the XML file. These are foundational metrics and represent the tokens extracted at the analyzer phase. The base metrics represented include the Number of Simple States (NSS), Number of Initial States (NIS), Number of Final States (NFS), Number of Composite States (NCS), and Number of History States (NHS). These values provide a fundamental understanding of the complexity of behavioral diagrams.



The screenshot shows the BCMT TOOL application window. It has a menu bar with 'File' and 'Exit'. Below the menu bar is a toolbar with icons for file operations. The main area has two tabs: 'Metrics' and 'Views'. Under the 'Metrics' tab, there are two sub-tabs: 'Base Metrics' and 'Derived Metrics'. The 'Base Metrics' sub-tab is selected, displaying a list of five metrics with their corresponding values in input fields:

Metric	Value
Number of simple states	7
Number of initial states	2
Number of final states	0
Number of composite state	1
Number of history states	1

Figure 7.6: Base Metrics Values

As illustrated in Figure 7.7, the derived metrics are calculated from the base metrics using established equations. These values are intended to provide deeper insight into how analyzable UML behavioral diagrams are.

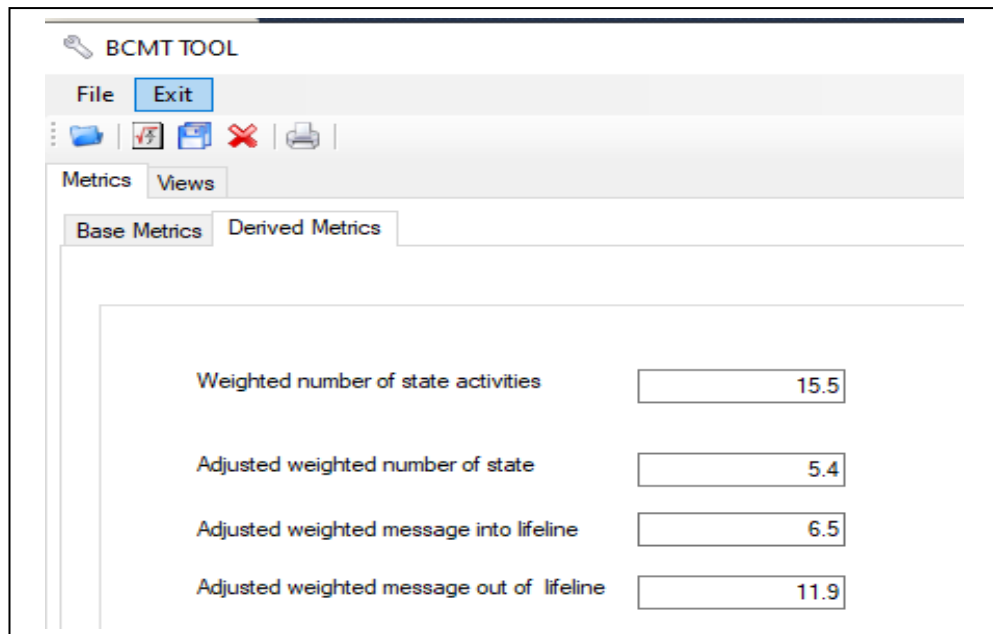
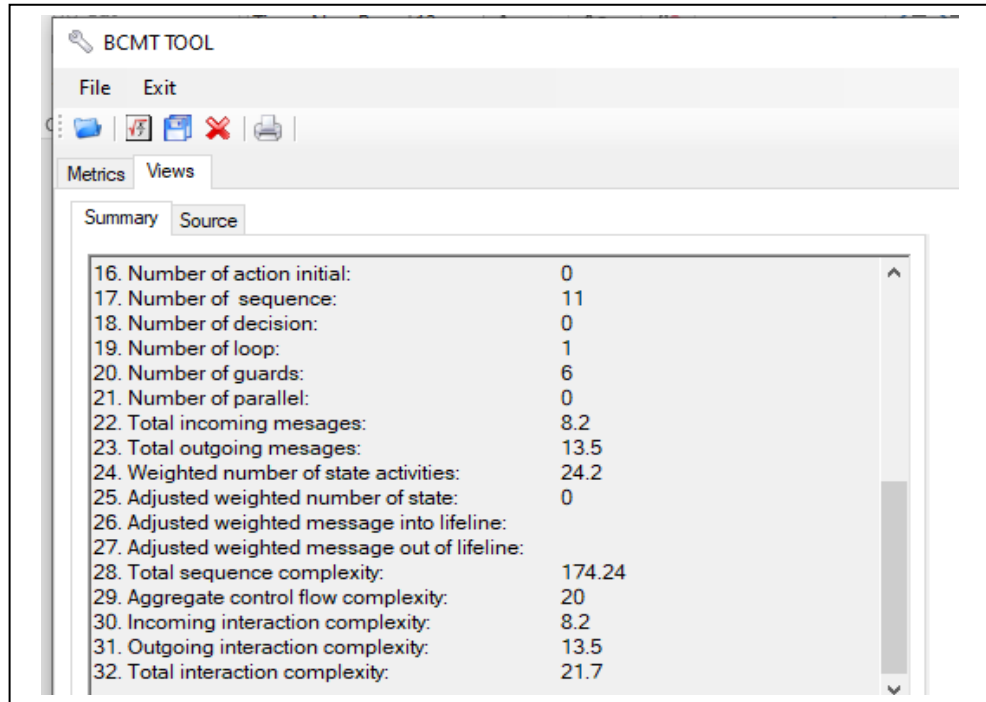


Figure 7.7: Derived Metrics Values

The computed metric outputs are stored in a plain text format. These values result from running the tool on an uploaded XML file and carrying out metric calculations. The saved data can be accessed later for continued analysis or documentation purposes. Figure 7.8 represents this structure.



The screenshot shows the BCMT TOOL window with a menu bar (File, Exit) and a toolbar. Below the menu bar are tabs for Metrics and Views. Under the Metrics tab, there are sub-tabs for Summary and Source. The Summary sub-tab is active, displaying a list of 16 metrics and their corresponding values in a text file format.

16. Number of action initial:	0
17. Number of sequence:	11
18. Number of decision:	0
19. Number of loop:	1
20. Number of guards:	6
21. Number of parallel:	0
22. Total incoming messages:	8.2
23. Total outgoing messages:	13.5
24. Weighted number of state activities:	24.2
25. Adjusted weighted number of state:	0
26. Adjusted weighted message into lifeline:	
27. Adjusted weighted message out of lifeline:	
28. Total sequence complexity:	174.24
29. Aggregate control flow complexity:	20
30. Incoming interaction complexity:	8.2
31. Outgoing interaction complexity:	13.5
32. Total interaction complexity:	21.7

Figure 7.8: Calculated Metrics Values in Text File

7.9 Discussion

The development and implementation of the Behavioral Complexity Metrics Tool (BCMT) automates the computation of complexity metrics for UML behavioral diagrams. The tool enhances efficiency, accuracy, and consistency in evaluating the complexity of statechart, sequence, and activity diagrams. The integration of base and derived metrics allows for a comprehensive assessment of behavioral complexity, providing valuable insights into model structure and interactions.

The BCMT is able to process Extensible Markup Language (XML) files, enabling users to seamlessly load UML behavioral diagrams and extract complexity data. The tool follows a well-defined architecture, composed of three key modules namely the input module, the analyzer module, and the output module. The input module facilitates the loading of XML files, ensuring that users can easily access and visualize diagram elements. The analyzer module, which comprises lexical analysis and parsing, executes the computation of complexity metrics, ensuring that both base and derived metrics are accurately calculated. Finally, the output module allows for easy review, storage, and printing of calculated metrics.

The BCMT reduces the likelihood of human error in UML diagram evaluation. Traditional manual assessment methods are often inconsistent and time-consuming, particularly for large and intricate behavioral diagrams. The predefined algorithms, ensure objective and repeatable complexity evaluations, making it a reliable asset for software designers, developers, and researchers. The ability to store and retrieve calculated metrics further enhances its practical utility, allowing users to track complexity trends over time and make informed design decisions.

Furthermore, the tool has the potential for industry adoption and integration with existing UML modeling environments such as Enterprise Architect, Visual Paradigm, and IBM Rational Software Architect. Incorporating BCMT into these platforms would allow for seamless complexity evaluation within professional workflows, improving design practices across software projects. Additionally, enhancements such as real-time complexity monitoring and interactive visualizations could further improve its usability in industrial settings.

7.10 Summary

This chapter introduced the Behavioral Complexity Metrics Tool (BCMT), developed to achieve the fourth research objective of automating complexity metric calculations for UML behavioral diagrams. It outlined the tool's core features, including XML file handling, metric calculation, and output functions such as saving and printing results. The structure of BCMT was explained through its input, analyzer, and output modules, each responsible for distinct phases of processing. Algorithms used to calculate both base and derived metrics were discussed, highlighting their role in ensuring precise complexity analysis. Finally, the execution process of the BCMT tool was described, outlining how users can load XML files, calculate metrics, review results, and save or print the calculated values. The tool's ability to generate stored complexity reports enables continuous analysis and model improvement.

CHAPTER EIGHT

EXPERIMENTAL VALIDATION OF UML BEHAVIORAL METRICS

8.1 Introduction

The chapter presents a detailed discussion on the validation of the defined behavioral complexity metrics. Specifically, the chapter solves research objective 5 which was to conduct a controlled experiment to evaluate whether the proposed metrics can be useful predictors of the analyzability of UML behavioral diagrams.

8.2 Context Definition

The experiment was conducted using thirty XML files. The use of thirty XML files was justified by the need to provide a diverse yet manageable sample that captures varying levels of complexity in UML behavioral diagrams. A sample size of 30 is widely accepted in empirical studies as sufficient for meaningful analysis while ensuring the results are statistically reliable and representative.

Students from Murang'a University of Technology were involved in the experiments, specifically those enrolled in Bachelors of Science in Software Engineering, Bachelors of Science in Information Technology, Bachelors of Science in Computer Science, Bachelors of Science in Computer Technology, and Bachelor of Business Information courses. The study focused on final-year students, as they were considered to have sufficient academic exposure to software engineering principles, programming, and complexity analysis. Given their advanced knowledge in the field, these students provided the necessary context for the study.

To ensure a structured and comprehensive approach, the experiment was conducted in three phases, with different groups of students participating in each. A total of nineteen students took part in both the first experiment and second experiment, while twenty students participated in the experiment.

All students underwent intensive training on UML modelling before conducting the experiment. This training aimed to bridge any gaps in their knowledge, ensuring they fully understood the behavioral structural and runtime behavior of UML behavioral diagrams being analyzed in the study. By involving students with a strong academic background and practical experience in software engineering, the study ensured that

the findings were reliable and that the validation process adhered to formal software complexity assessment standards

8.3 Experimental Execution Approach

Three separate experimental procedures were executed, each designed to assess behavioral complexity metrics across sequence diagrams, activity diagrams, and statechart diagrams. These experiments aimed to evaluate the impact of complexity on diagram analyzability of subjective assessments and objective measurements. A within subject design was used where each subject was required to work on all independently. The subjective phase required participants to rank analyzability based on their perception of the behavioral diagram's analyzability while the objective phase involved subjects recording time spent on each diagram. The time recording served as empirical indicators of how complexity levels affect system analyzability.

The first experiment focused on sequence diagrams, which model the interaction between objects over time. Subjects were provided with ten sequence diagrams of varying complexity, each containing different messages, lifelines, and control flows. In the subjective phase, they ranked the analyzability of sequence diagrams on a Likert scale of 1 to 5 where 1 is very complex, 2 is complex, 3 is neither complex nor easy, 4 is easy and 5 is very easy. In the objective phase, the subjects were required to record the time taken to analyze each sequence diagram.

The second experiment analyzed activity diagrams, which represent workflow and control flow within a system. Subjects were provided with ten activity diagrams of varying complexity, each containing different states, and control flows. In the subjective phase, they ranked the analyzability of activity diagrams on a Likert scale of 1 to 5 where 1 is very complex, 2 is complex, 3 is neither complex nor easy, 4 is easy and 5 is very easy. In the objective phase, they were asked to record the time taken to examine each activity diagram.

The third experiment examined statechart diagrams, which illustrate the different states of a system and the transitions between them. Subjects were provided with ten statechart diagrams of varying complexity, each containing different types of states, and control flows. In the subjective phase, they ranked the analyzability of statechart diagrams on a Likert scale of 1 to 5 where 1 is very complex, 2 is complex, 3 is neither

complex nor easy, 4 is easy and 5 is very easy . In the objective phase, they were asked to record the time taken to examine each statechart diagram.

A within-subject experimental design was used across all three experiments. To ensure consistency and reliability. Subjects performed their tasks individually within a two-hour session to ensure that their assessments were independent and unaffected by peer influence.

A total of 30 UML behavioral diagrams were used in the experiment, with 10 diagrams assigned to each experiment type. These diagrams were carefully selected to represent different complexity levels, as determined by predefined behavioral complexity metrics.

The BCMT tool was utilized to calculate and analyze complexity metrics for the diagrams. However, only the researcher interacted with the tool to gather metric values from UML diagrams, as the subjects did not have direct access to it. This approach ensured that the study remained focused on evaluating subjects' subjective perceptions while an independent complexity assessment was conducted using the BCMT tool.

8.4 Lessons Learned from the Pilot Study

The primary objective of the pilot study was to determine whether the proposed UML complexity metrics correlated with subjects' subjective ratings of analyzability. Additionally, the study aimed to establish whether these metrics could serve as predictors of UML diagram analyzability. To ensure accuracy and relevance, a pilot study was carried out to evaluate and validate the questionnaire developed for subjective and objective evaluations. The version employed in all three experiments is provided in Appendix 2.

The pilot study revealed that the experimental setup was well-structured, and subjects were able to complete their tasks within the expected timeframe without issues. No modifications were required for the number of questions in the questionnaire, as subjects were able to provide consistent and insightful responses. The findings from the pilot study confirmed that the proposed UML complexity metrics effectively predict the analyzability characteristics of sequence, activity, and statechart diagrams.

8.5 Subjects' Background

The subject's level of familiarity with software engineering was established to confirm their readiness to carry out the exercise. The study established that 13 subjects which account for 23.6% said they were conversant with 1 to 2 Software engineering units while those conversant with above 3 to 4 units were 22 subjects which accounts for 40% of the total and, 20 subjects had covered more than 5 subjects in software engineering which accounts for 36.4 % . The analysis indicate that majority of the subjects had done more than 3 subjects thus fit to participate in the experiment. This information of the subject's response is shown in Table 8.1

Table 8.1: Units in Software Engineering

Units Taken	Frequency	Percent (%)
1-2	13	23.6
3-4	22	40
5 and above	20	36.4
Total	55	100

Also, details about subjects' training on number of programming languages was established. Twenty three (23) subjects were conversant with 3 to 4 programming languages while the rest 32 were conversant with 5 and above as shown in Table 8.2

Table 8.2: Subjects by Number of Programming Languages

Range of Programming Languages	Frequency of Subjects	Percent (%)
3-4	23	41.8
5 and above	32	58.2
Total	55	100

Enquiries about whether subjects had heard about UML diagrams before the training class of the experiment was determined. Out of 55 subjects, all confirmed they had heard about UML diagrams. Table 8.3 presents this summary.

Table 8.3: Training on UML Diagrams

Training on UML Diagrams	Number of Subjects	Percent (%)
Yes	55	100
No	0	10
Total	55	100

Enquiries about the subject's depth of knowledge on UML was done. 1 of the subjects had low knowledge on UML behavioral diagrams while 38 subjects had moderate knowledge of UML diagrams, 14 subjects had high knowledge on behavioral diagrams and 2 had very high knowledge on UML as seen in Table 8.4. Only one subject (1.8%) exhibited low knowledge of UML behavioral diagrams. To ensure the reliability and validity of the results, this subject was given a brief orientation session covering the key concepts of UML behavioral diagrams. This indicates that the majority had moderate knowledge on UML and therefore could participate in the empirical exercise.

Table 8.4: Knowledge on Behavioral Diagrams

Knowledge on Behavioral Diagrams	Number of Subjects	Percent (%)
Low	1	1.8
Moderate	38	69.1
High	14	25.5
Very high	2	3.6
Total	55	100

The researcher established on the number of features of UML behavioral diagrams that the subjects could comfortably use. 23 of the subjects could comfortably use element feature while 16 subjects could use control flow, 5 subjects could use message, 1 subject could use both element and control flow, 2 subjects were comfortable with both control flow and message, 2 subjects were comfortable with both element and message and, 6 subjects were comfortable with all the features of a behavioral diagram as seen in Table 8.5.

Table 8.5: Use of Behavioral Features

Use of Behavioral features	Number of Subjects	Percent (%)
Element	23	41.8
Control flow	16	29.1
Message	5	9.1
Element, control flow	1	1.8
Element, message	2	3.6
Control flow, message	2	3.6
All	6	10.9
Total	55	100

8.6 Experimental Planning

This section provides a detailed description of the three experiments executed to validate the proposed metrics.

8.6.1 General Description of the Experiment

This study executed a set of experiments to find out whether the complexity metrics can aid in predicting how easy or hard it is to analyze UML behavioral diagrams. The goal was to check if these metrics could show how complex a diagram is, and whether they are linked to the time subjects take to analyze those diagrams.

Three separate experiments were conducted, with each one focusing on a different type of UML behavioral diagram, namely sequence diagrams, statechart diagrams, and activity diagrams. For each type, a group of complexity metrics was applied, which had been chosen based on the structures and behavior of those diagrams.

A pilot study was conducted before the main experiment. This aided the researcher in testing the questionnaire and making sure the tasks were clear and understandable to the subjects. The pilot study also confirmed that the selected metrics could be applied correctly and that the analysis methods were reliable.

Systematic random sampling was used to select the subjects. This method involves selecting every n th subject from the population, which ensures that the sample is random and representative. All selected subjects had already taken courses in software engineering, programming, and had been trained on UML diagram modeling, so they had the right background knowledge to carry out the task effectively.

A within-subject design was used whereby the subjects were asked to examine a set of diagrams on their own and to rate how easy or hard each one was to analyze. They also had to do this without any aid, so the results would not be influenced by others. This setup aided the researcher gather reliable and consistent data.

The study measured two variables namely, the independent variables (the complexity metric values) and the dependent variables (how the subjects rated the diagrams, and how long it took them to finish analyzing each one). Ratings were given using a Likert scale, and time was recorded for each session.

8.6.2 The First Experiment

Definition: The experiment was designed to investigate the relationship between the sequence metrics with sequence analyzability and time to analyze. The experiment also aimed to examine whether the selected sequence metrics could serve as indicators of analyzability.

Selection of Subjects: To ensure the validity of the experiment, systematic sampling was used to select a sample of nineteen subjects for the experiment. All had taken units in software engineering, programming languages, and training on UML diagrams, ensuring they had the required expertise to evaluate the analyzability of UML behavioral diagrams.

Variables: The experiment includes both the independent and dependent variables. Independent variables were the sequence metrics, while the dependent variable was the analyzability ranking and time to analyze a diagram.

Hypotheses: The following hypotheses were tested:

- i. Null Hypothesis, H_0 : The selected sequence metrics have no significant relationship with the analyzability rating of a sequence diagram.
- ii. Alternative Hypothesis, H_1 : The selected sequence metrics have a significant correlation with diagram analyzability.
- iii. Null Hypothesis, H_0 : The selected sequence metrics have no significant relationship with the time to analyze a diagram.
- iv. Alternative Hypothesis, H_1 : The selected sequence metrics have a significant correlation with time to analyze a diagram.

Experimental Design: A within-subject experimental design was employed, whereby each subject analyzed all the sequence diagrams independently. The design was selected because of its high predictive power. Subjects were asked to rate the analyzability of sequence diagrams using a Likert scale of –very complex, complex, neither complex nor easy, easy, very easy.

Metrics Data

The metrics values calculated from sequence diagrams were the derived metrics namely, Total Sequence Complexity (TSC), Aggregate Control Flow Complexity (ACFC), and Total Interaction Complexity (TIC) represented the independent variable. Table 8.6 presents a summary of the metric data collected.

Table 8.6: Metrics for Sequence Diagrams

Sequence diagram number	TSC	ACFC	TIC
1	67.24	11	9.6
2	213.16	12.5	11.5
3	116.64	17	11.2
4	190.44	17	12.3
5	207.36	12	14.2
6	243.36	20.5	11.7
7	912.04	17	15.7
8	556.96	18	14.9
9	519.84	30	18.2
10	49	10	7.1

To achieve a more comparative perspective, the derived metrics were subjected to descriptive statistics namely, mean, minimum, maximum, range and standard deviation. This analysis is represented in Table 8.7.

Table 8.7: Descriptive Statistics for Sequence Metrics

Metric	Mean	Minimum	Maximum	Range	Standard deviation
TSC	307.60	49.00	912.04	272.95	272.95
ACFC	16.5	10.00	30.00	5.87	5.87
TIC	12.64	7.10	18.20	3.20	3.20

As seen in Table 8.7, the lowest value is 49.00 and the highest is 912.04 for the TSC metric while the lowest value is 10.00 and the highest is 30.00 for the ACFC metric. The lowest value is 7.10 and the highest is 18.20 for the TIC metric. This shows that the metrics tool was measuring accurately as expected, meaning , a sequence diagram cannot have zero TSC unless if empty.

8.6.3 The Second Experiment

Definition: This experiment was designed to assess how the activity diagram metrics relate to both the analyzability and the time required to analyze them. Additionally, it sought to determine whether the chosen activity metrics could reliably act as predictors of analyzability.

Selection of Subjects: To ensure validity of the experiment, systematic random sampling was used to select a sample of nineteen subjects for the experiment. All had taken units in software engineering, programming languages, and training on UML diagrams ensuring they had the required expertise to evaluate the analyzability of UML behavioral diagrams.

Variables: The experiment includes both the independent and dependent variables. Independent variables were the activity metrics, while the dependent variable was the analyzability ranking and time to analyze of a diagram.

Hypotheses: The following hypotheses were tested:

- i. Null Hypothesis, H_0 : The selected activity metrics have no significant relationship with the analyzability rating of a sequence diagram.
- ii. Alternative Hypothesis, H_1 : The selected activity metrics have a significant correlation with diagram analyzability.
- iii. Null Hypothesis, H_0 : The selected activity metrics have no significant relationship with the time to analyze a diagram.
- iv. Alternative Hypothesis, H_1 : The selected activity metrics have a significant correlation with time to analyze a diagram.

Experimental Design: A within-subjects experimental design was employed, whereby each subject analyzed all the activity diagrams independently. The design was selected because of its high predictive power. Subjects were asked to rate the

analyzability of activity diagrams using a Likert scale of –very complex, complex, neither complex nor easy, easy, very easy

Metrics Data

Metrics data, representing the independent variable of the 10 activity diagrams, were collected from the BCMT tool. Table 8.8 represents metric values, which represent the independent variable of the 10 activity diagrams. The composite metrics calculated from activity diagrams were AWNS, ACFC, and TIC.

Table 8.8: Metrics for Activity Diagrams

Activity diagram number	AWNS	ACFC	TIC
1	24	28	27.5
2	22	21	25
3	20	15	20
4	10	8	8
5	24	21	28.4
6	26	20	30
7	24	23	25
8	24	22	27.9
9	14	11	12.5
10	16	11	15

To gain a more comparative perspective, the derived metrics were analyzed using descriptive statistics, including the mean, minimum, maximum, range, and standard deviation. The results of this analysis are presented in Table 8.9.

Table 8.9: Descriptive Statistics for Activity Metrics

Metric	Mean	Range	Minimum	Maximum	Standard deviation
AWNS	20.4	16.0	10.0	26.0	5.32
ACFC	18	20.0	8.0	28.0	6.41
TIC	21.93	22.0	8.0	30.0	7.65

As seen in Table 8.9, the lowest value is 16.00 and the highest is 26.0 for the AWNS metric while the lowest value is 8.00 and the highest is 28.00 for the ACFC metric. The lowest value is 8.00 and the highest is 30.0 for the TIC metric. This shows that

the metrics tool was measuring accurately as expected, meaning an activity diagram cannot have zero TSC unless if empty.

8.6.4 The Third Experiment

Definition: The experiment was designed to investigate the relationship between the metrics with statechart analyzability and time to analyze. The experiment also aimed to examine whether the selected activity metrics could serve as indicators of analyzability.

Selection of Subjects: To ensure validity of the experiment, systematic random sampling was used to select a sample of twenty subjects for the experiment. All had taken units in software engineering, programming languages, and training on UML diagrams ensuring they had the required expertise to evaluate the analyzability of UML behavioral diagrams.

Variables: The experiment incorporated both independent and dependent variables. The statechart metrics served as the independent variables, whereas the analyzability ratings and the time taken to analyze each diagram were the dependent variables.

Hypotheses: The following hypotheses were tested:

- i. Null Hypothesis, H_0 : The selected statechart metrics have no significant relationship with the analyzability rating of a sequence diagram.
- ii. Alternative Hypothesis, H_1 : The selected statechart metrics have a significant correlation with diagram analyzability.
- iii. Null Hypothesis, H_0 : The selected statechart metrics have no significant relationship with the time to analyze a diagram.
- iv. Alternative Hypothesis, H_1 : The selected statechart metrics have a significant correlation with time to analyze a diagram.

Experimental Design: A within-subjects experimental design was employed whereby each subject analyzed all the statechart diagrams independently. The design was selected because of its high predictive power. Subjects were asked to rate the analyzability of statechart diagrams using a Likert scale of –very complex, complex, neither complex nor easy, easy, very easy

Metrics Data

Metrics data, representing the independent variable of the 10 statechart diagrams, were collected from the BCMT tool. This data is presented in Table 8.10. The data were then analyzed using descriptive statistics and the results are shown in Table 8.11.

Table 8.10: Metrics for Statechart Diagrams

Statechart diagram number	WNSA	ACFC	TIC
1	24.2	20	21.7
2	15.5	16	25.6
3	34.5	15	22.2
4	23	16	21
5	22.5	22	16.3
6	19.8	8	17
7	14	11.7	9.2
8	11	12	13.2
9	21.2	15.5	11.9
10	16.5	11	18.1

To obtain a more comparative perspective, the derived metrics were analyzed using descriptive measures, including the mean, minimum, maximum, range, and standard deviation. The results of this analysis are presented in Table 8.11.

Table 8.11: Descriptive Statistics for Statechart Metrics

Metric	Mean	Minimum	Maximum	Range	Standard deviation
WNSA	20.22	11.00	34.50	23.50	6.61
ACFC	14.72	8.00	22.00	14.00	4.22
TIC	17.62	9.20	25.60	16.40	5.14

As seen in Table 8.11, the lowest value is 11.00 and the highest is 34.50 for the WNSA metric while the lowest value is 8.00 and the highest is 22.00 for the ACFC metric. The lowest value is 9.20 and the highest is 25.60 for the TIC metric. This shows that the metrics tool was measuring accurately as expected, meaning a statechart diagram cannot have zero WNSA unless if empty.

8.6.5 Validity Evaluation

8.6.5.1 Threat to Construct Validity

The three experiments conducted in this study were subjective, as the dependent variable was measured based on the subjects' personal experiences and perceptions. Subjective measurements introduce potential risks to construct validity, as they rely on individual interpretations, prior knowledge, and cognitive biases. If the participants lack sufficient expertise or relevant background knowledge, their responses may not accurately reflect the intended constructs, leading to skewed results. Therefore, measures were put in place to ensure that the dependent variables were as constructively valid as possible.

To mitigate the risk posed by subjective data, subjects who were in their final year of study were selected for the study. This is because the final-year students have covered more units related to software, programming languages, and Unified Modeling Language (UML) diagrams. Additionally, they have received detailed training in dynamic behavioral diagrams namely, statechart diagrams, sequence diagrams, and activity diagrams. This approach minimized variations in analyzing and interpretation.

8.6.5.2 Threat to Internal Validity

The selection of behavioral diagrams was carefully considered to ensure they were both simple and general. Therefore, the chosen diagrams did not require detailed prior knowledge beyond what the subjects had acquired in their studies. Additionally, it allowed the subjects to focus on key aspects of the experiment.

The selection of behavioral diagrams was guided by the need to ensure variation in their metric values, as determined by the proposed complexity metrics. This variation allowed the study to capture a broad spectrum of structural and behavioral characteristics, enabling comprehensive analysis and comparison. By including diagrams with diverse complexity levels, the experiment accommodated a range of subject opinions and supported meaningful evaluation of the metrics

8.6.5.3 Threat to External Validity

Subject characteristics that normally threaten the generalizability of research results were carefully considered, such as the use of student subjects. The reliance on students as subjects is a common limitation in experimental studies, as their knowledge,

experience, and cognitive abilities may not fully represent those of professionals in the field. To mitigate this concern and enhance the external validity of the study, specific measures were taken in the selection process.

Only students in their final year of study were selected as subjects. This decision was based on the assumption that final-year students have acquired more knowledge in software engineering, programming languages, and UML diagram techniques, including behavioral diagrams.

8.6.5.4 Threat to Conclusion Validity

Sample data sizes have been known to affect conclusion validity of a study if they are insufficient. The following sample data sizes were used: 10 sequence diagrams and 19 subjects resulting to 190 data values for the first experiment, 10 statechart diagrams and 18 subjects resulting to 180 data values for the second experiment, and 10 activity diagrams and 18 subjects resulting to 180 data values for the third experiment. The size of these data values provided the predictive power that aided mitigate this type of threat to conclusion validity.

Sample data sizes have been known to affect conclusion validity of a study. Insufficient sample sizes can lead to unreliable results, reduced statistical power, and an increased likelihood of errors, such as Type I or Type II errors. The size of the dataset directly influences the ability to detect significant relationships, patterns, or differences within the collected data. To mitigate this threat and ensure reliable and meaningful insights, appropriate sample sizes were selected for this study.

In the first experiment, 10 sequence diagrams were used with 19 subjects, generating a total of 190 data values. The second experiment utilized 10 statechart diagrams with 18 subjects, resulting in 180 data values. Similarly, the third experiment employed 10 activity diagrams with 18 subjects, also yielding 180 data values. In total, across all three experiments, 550 data values were collected. The size of these data values provided the predictive power that aided mitigate this type of threat to conclusion validity.

Predictive power is essential in confirming that observed effects are not due to random variation but rather reflect actual relationships within the data. The inclusion of 10

distinct diagrams in each experiment allowed for a diverse representation of behavioral patterns, while the number of subjects per experiment ensured adequate variation in participant responses. This combination aided reinforce the reliability of the results.

8.7 Results

This sections presents the data analysis results namely, correlation and regression calculated from the data collected.

8.7.1 Results from the First Experiment

In the initial experimental stage, a set of complexity metrics for sequence was extracted from a series of UML sequence diagrams. To assess the relationships among these metrics, Spearman's rank-order correlation was employed. The intent behind this correlation analysis was to examine the extent to which the metrics were correlated with each other.

Results revealed statistically significant and strong correlations among the three sequence diagram metrics namely, Total Sequence Components (TSC), Aggregate Control Flow Complexity (ACFC), and Total Interaction Complexity (TIC). Specifically, the TSC metric had a strong positive relationship with both ACFC and TIC. The correlation between TSC and ACFC was 0.718, significant at the 95% confidence level ($p = 0.019$), while the correlation between TSC and TIC was even higher at 0.867, significant at the 99% confidence level ($p = 0.019$). Additionally, ACFC and TIC also showed a strong correlation of 0.693, with statistical significance at the 95% confidence level ($p = 0.026$). Table 8.12 shows the summary of this analysis.

Table 8.12: Correlation between Sequence Metrics

Metric	TSC	ACFC	TIC
TSC	1.00	0.718*, $p=0.019$	0.867**, $p=0.019$
ACFC		1.00	0.693*, $p=0.026$
TIC			1.00

Key: **=99% Confidence; *=95% Confidence

The strength of these associations indicates that the metrics may be capturing overlapping structural or behavioral attributes of the sequence diagrams. Therefore, the study adopted an approach based on individual linear regression analysis for each metric to eliminate the influence of multicollinearity.

Following the regression analysis setup, data from the experimental subjects were reviewed. The returned questionnaires from the subjects were complete and, therefore, included in the analysis. To evaluate the subjective perception of analyzability, means of subject ratings for each diagram were calculated. The dataset was examined for normality using the Shapiro-Wilk test which revealed that the data was not normal. Therefore, Spearman's rho was employed to test the hypotheses.

The significance level was set at $\alpha = 0.05$ to control the Type 1 error. This means that any test result with a p-value less than 0.05 would lead to rejection of the null hypothesis (H_0) in favor of the alternative hypothesis (H_1). The two-tailed nature of the tests accounted for the possibility of both positive and negative correlations. A strong negative correlation was observed between the metrics and analyzability. TSC showed the strongest correlation, with a coefficient of -0.888 and a p-value of <0.001, indicating a highly significant relationship at the 99% confidence level. ACFC demonstrated a correlation of -0.775 with a p-value of 0.024, also statistically significant at the 99% confidence level, while TIC recorded a correlation coefficient of -0.742 and a p-value of 0.023, marking significance at the 95% confidence level.

These results, summarized in Table 8.13, supported the rejection of the null hypothesis for all three metrics.

Table 8.13: Correlation with Analyzability Ranking

Metric	Coefficients	Sig. (2-tailed)
TSC	-0.888**	<0.001
ACFC	-0.775**	0.024
TIC	-0.742*	0.023

Key: **=99% Confidence; *=95% Confidence

To further investigate the predictive strength of each metric, individual linear regression analyses were carried out. This approach was particularly appropriate given

the strong correlations among metrics, which raised concerns about multicollinearity. The coefficient of determination (R^2) and the p-value for each regression model provided insights into how well each metric explained the variation in analyzability ratings. The TSC obtained an R^2 of 0.737, indicating that 73.7% of the variability in analyzability, with a p-value of 0.001. ACFC accounted for 51.9% of the variation ($R^2 = 0.519$) and had a p-value of 0.019, confirming statistical significance. TIC explained 57.0% of the variance ($R^2 = 0.570$) and yielded a p-value of 0.012. These findings, shown in Table 8.14, affirm the predictive validity of the selected metrics, especially TSC, which emerged as the strongest predictor of diagram analyzability.

Table 8.14: Regression Results for Analyzability Rating

Metric	R^2	p-value
TSC	0.737	.001
ACFC	0.519	.019
TIC	0.570	.012

Key: **=99% Confidence; *=95% Confidence

In addition to subjective ratings, the study captured an objective measure of analyzability, which is the time taken to analyze a sequence diagram. Subjects were required to indicate the time they started working on a diagram and the time they completed working on it. Time to analyze is then calculated as the difference in these time recordings.

The Shapiro- Wilk check was conducted to verify the data for further analysis. It revealed that the data was not normal. Due to this, Spearman's rho was utilized to establish the relationship between the two variables. The TSC metric achieved a correlation coefficient of 0.988, the ACFC metric achieved a correlation coefficient of 0.718 and the TIC metric achieved 0.867 respectively. Table 8.15 shows the summary of the results. This indicates a positive relationship between the sequence metrics and time to analyze a sequence diagram.

Table 8.15: Relationship with Analyzability Timing

Metric	Coefficients	Sig. (2-tailed)
TSC	0.988**	<0.001
ACFC	0.706*	0.023
TIC	0.806**	0.005

Key: **=99% Confidence; *=95% Confidence

Regression models were applied once more to confirm these correlations and assess predictive capability. TSC and TIC have high R^2 values and p-values below 0.05 implying that they can be used as predictors of time to analyze a sequence diagram. The ACFC metric has an R^2 of 0.243 and a p-value higher than 0.05 meaning that it cannot be relied upon as a predictor of time to analyze a sequence diagram. Table 8.16 represents the analysis results.

Table 8.16: Regression Results for Analyzability Timing

Metric	R^2	p-value
TSC	0.769	<.001
ACFC	0.243	0.147
TIC	0.554	0.014

8.7.2 Results from the Second Experiment

In this second experiment, the focus was placed on evaluating the relationships among complexity metrics derived from UML activity diagrams. These metrics included the Adjusted Weighted Number of State (AWNS), Aggregate Control Flow Complexity (ACFC), and Total Interaction Complexity (TIC). To assess the strength of correlation among these variables, the Spearman's rho correlation coefficient was utilized. This choice was informed by the non-parametric nature of the data.

Results revealed strong and statistically significant positive correlations between the metrics. AWNS and ACFC recorded a correlation coefficient of 0.764, with a p-value of 0.010, indicating significance at the 95% confidence level. The association between AWNS and TIC was even stronger, with a coefficient of 0.956 and a p-value less than 0.001, signifying significance at the 99% confidence level. Additionally, ACFC and TIC showed a strong correlation of 0.670, also significant at the 95% level with a p-value of 0.034. These correlations suggested the presence of overlapping measurement

dimensions among the metrics, leading to the choice of individual regression analysis. Table 8.17 represents a summary of these results.

Table 8.17: Correlation between Activity Metrics

Metric	AWNS	ACFC	TIC
AWNS	1.00	0.764*, p=0.010	0.956**, p< 0.001
ACFC		1.00	0.670*, p=0.034
TIC			1.00

Key: **=99% Confidence; *=95% Confidence

All subjects' questionnaires were reviewed, and 1 was found to be incomplete, hence not considered for data analysis. The mean analyzability rating for each diagram was calculated. To assess whether the data followed a normal distribution, the Shapiro-Wilk test was applied. The results indicated that the data was not normal hence the use of Spearman's rho to test the study hypotheses. The results demonstrated a strong negative correlation between each complexity metric and the analyzability ratings, indicating that as the complexity of a diagram increased, its perceived ease of analysis decreased. Specifically, AWNS showed a correlation of -0.834, with a p-value of 0.003, ACFC produced a correlation of -0.844, and a p-value of 0.002, while TIC yielded the strongest inverse correlation of -0.854, also with a p-value of 0.002. All three relationships were statistically significant at the 99% confidence level, as presented in Table 8.18. These findings supported rejection of the null hypothesis and acceptance of the alternative hypothesis, confirming that all three metrics are significantly and negatively associated with analyzability.

Table 8.18: Correlation with Analyzability Ranking

Metric	Coefficients	Sig. (2-tailed)
AWNS	-0.834**	0.003
ACFC	-0.844**	0.002
TIC	-0.854**	0.002

Key: **=99% Confidence; *=95% Confidence

Individual linear regression analyses were performed for each metric to further validate their predictive utility. The coefficient of determination (R^2) and p-values

were used to evaluate the models. The results showed that AWNS had an R^2 of 0.749 and a p-value of 0.001, explaining nearly 75% of the variance in analyzability ratings. ACFC accounted for 73.2% of the variation, with $R^2 = 0.732$ and a p-value of 0.002. TIC had the highest explanatory power, with $R^2 = 0.763$ and a p-value of less than 0.001. These outcomes, summarized in Table 8.19, reinforce the metrics' roles as reliable predictors of analyzability.

Table 8.19: Regression Results for Analyzability Rating

Metric	R²	p-value
AWNS	0.749	.001
ACFC	0.732	.002
TIC	0.763	<0.001

Subjects' were asked to record both the start and end times during the process of analyzing each activity diagram to objectively measure analyzability. The time to analyze was calculated as the difference between these two values. As with the subjective data, the Shapiro-Wilk test indicated that the time data did not follow a normal distribution, prompting the use of Spearman's rho for correlation analysis.

The results revealed positive and significant relationships between complexity metrics and the time taken to analyze the diagrams. AWNS was strongly correlated with analysis time ($\rho = 0.900$, $p < 0.001$), TIC followed closely, with a correlation of 0.930, also with $p < 0.001$, while ACFC showed a significant correlation of 0.659, with a p-value of 0.038. These relationships are presented in Table 8.20.

Table 8.20: Relationship with Analyzability Timing

Metric	Coefficients	Sig. (2-tailed)
AWNS	0.900**	<.001
ACFC	0.659*	0.038
TIC	0.930**	<.001

Key: **=99% Confidence; *=95% Confidence

Further, regression model analysis was conducted to quantify the predictive power of each metric. TIC demonstrated the strongest predictive capacity with an R^2 of 0.760

and a p-value of 0.001. AWNS also showed high predictability, with an R^2 of 0.703 and a p-value of 0.002. ACFC showed predictability with an R^2 of 0.560 and a p-value of 0.013. Table 8.21 displays these results.

Table 8.21: Regression Results with Analyzability Timing

Metric	R²	p-value
AWNS	0.703	0.002
ACFC	0.560	0.013
TIC	0.760	0.001

8.7.3 Results from the Third Experiment

In this third experiment, the focus was placed on evaluating the relationships among complexity metrics derived from UML statechart diagrams. These metrics included Weighted Number of State Activities (WNSA), Aggregate Control Flow Complexity (ACFC), and Total Interaction Complexity (TIC). To assess the strength of correlation among these variables, the Spearman's rho correlation coefficient was utilized.

The results indicate that the three metrics are not significantly correlated with one another. Specifically, the correlation between WNSA and ACFC was 0.486, with a p-value of 0.154, which is above the 0.05 significance threshold. Similarly, the relationship between WNSA and TIC yielded a correlation coefficient of 0.479 and a p-value of 0.162, also statistically insignificant. ACFC and TIC had a correlation of 0.316 and a p-value of 0.374. These results suggest the absence of multicollinearity among the statechart metrics. These relationships are presented in Table 8.22.

Table 8.22: Correlation between Statechart Metrics

Metric	WNSA	ACFC	TIC
WNSA	1.00	0.486*, p=0.154	0.479**, p=0.162
ACFC		1.00	0.316*, p=0.374
TIC			1.00

Key: **=99% Confidence; *=95% Confidence

All subjects' questionnaires were reviewed, and 2 were found to be incomplete, hence not considered for data analysis. The mean analyzability rating for each diagram was calculated. Shapiro-Wilk test was conducted to assess the normality of the distribution. The results indicated it was non normal, justifying the continued use of Spearman's rho for hypothesis testing.

The findings revealed strong negative correlations between all three complexity metrics and the analyzability of statechart diagrams. WNSA exhibited a correlation of -0.845 with a p-value of 0.002, indicating a significant inverse relationship. ACFC showed a correlation coefficient of -0.674, with a p-value of 0.033. TIC displayed a strong inverse correlation at -0.657, also with a p-value of 0.039. As shown in Table 8.23, all these relationships were statistically significant at the 99% and 95% confidence level, confirming that as the complexity measured by these metrics increases, the diagrams are perceived to be more difficult to analyze.

Table 8.23: Relationship with Analyzability Ranking

Metric	Coefficients	Sig. (2-tailed)
WNSA	-0.845**	0.002
ACFC	-0.674*	0.033
TIC	-0.657**	0.039

Key: **=99% Confidence; *=95% Confidence

Therefore, the null hypothesis was rejected for all three cases. This supports the acceptance of the alternative hypothesis, confirming that the metrics are valid indicators of analyzability in the context of statechart diagrams.

Simple linear regression analyses were conducted to quantify the degree of influence of metrics on analyzability ratings. The metrics explained 91.8% of the variance in analyzability ($R^2 = 0.918$), with a p-value of 0.001. The results are outlined in Table 8.24.

Table 8.24: Regression Results for Analyzability Rating

Metric	R²	p-value
WNSA	0.918	0.001
ACFC		
TIC		

In addition to subjective ratings, an objective measure of analyzability was calculated by recording the time each subject took to complete the analysis of a statechart diagram. Subjects' indicated their start and finish times, and the duration was calculated as the time difference. The Shapiro-Wilk test showed that the time data were non-normally distributed, hence the correlation analysis again relied on Spearman's rho.

The correlation results showed positive and statistically significant relationships between the complexity metrics and the time taken to analyze each diagram. WNSA achieved a correlation coefficient of 0.745, with a p-value less than 0.013, indicating a very strong and significant relationship. TIC obtained a coefficient of 0.648 and a p-value of 0.043. ACFC produced a moderately strong correlation of 0.657, with a p-value of 0.039. The results are summarized in Table 8.25.

Table 8.25: Relationship with Analyzability timing

Metric	Coefficients	Sig. (2-tailed)
WNSA	0.745*	0.013
ACFC	0.657*	0.039
TIC	0.648*	0.043

Key: **=99% Confidence; *=95% Confidence

Further, regression analysis was conducted to evaluate the proportion of variance in analysis time that could be explained by each metric. The metrics obtained 89.8% of the variability in time ($R^2 = 0.898$), with a p-value of 0.002. Table 8.26 shows that all three metrics have high R^2 values and the p-value is less than 0.05. Therefore, the null hypothesis is rejected and accepted the alternative hypothesis.

Table 8.26: Regression Results with Analyzability Timing

Metric	R²	p-value
WNSA ACFC TIC	0.898	0.002

8.8 Discussion

This section discusses the findings from the three experimental phases namely, sequence diagrams, activity diagrams, and statechart diagram.

8.8.1 Implications of Results from the First Experiment

Analysis of sequence diagrams revealed that Total Sequence Complexity (TSC), Average Control Flow Complexity (ACFC), and Total Interaction Complexity (TIC) strongly correlated with the subjects rating of analyzability of sequence diagrams. The regression results demonstrated that these metrics significantly predicted the difficulty and time required to analyze a sequence diagram. The results suggest that as sequence complexity increases, the analyzability of the diagram decreases, requiring more time examine.

These findings align with existing study that complexity affects system analyzability. The strong negative correlation between TSC and analyzability ranking implies that highly complex sequence diagrams hinder quick analysis. Similarly, the significant correlation between TIC and analysis time indicates that interaction complexity significantly impacts analyzability.

8.8.2 Implications of Results from the Second Experiment

Results from activity diagram experiments further strengthened the hypothesis that complexity metrics significantly impact analyzability. The study showed that Adjusted Weighted Number of States (AWNS), ACFC, and TIC were strongly correlated with the time required to analyze activity diagrams and their perceived difficulty. The regression results confirmed that these metrics could serve as analyzability indicators.

The strong negative correlation between AWNS and analyzability ranking supports the argument that increased complexity reduces analyzability. Additionally, the

positive correlation between AWNS and time to analyze imply that diagram structure significantly influences analyzability.

8.8.3 Implications of Results from the Third Experiment

The statechart diagram analysis revealed similar trends, with Weighted Number of State Activities (WNSA), ACFC, and TIC indicating significant correlations with analyzability and time to analyze. However, unlike the sequence and activity diagram results, statechart diagram metrics exhibited a lower correlation, suggesting that statechart diagrams may inherently have a more complex interpretation structure.

The results demonstrated that WNSA had a negative correlation with analyzability ranking, while TIC had a significant positive correlation with analysis time. These findings suggest that statechart complexity contributes to analysis difficulty but not as strongly as the other diagram types.

8.8.4 Global Comparison of Results

The results obtained from the three experimental phases focused on sequence diagrams, activity diagrams, and statechart diagrams support the hypothesis that complexity metrics can serve as valid indicators and predictors of analyzability in UML behavioral diagrams. Across all diagram types, a clear trend emerged that as the complexity of a diagram increased, both the subjective ratings of analyzability and the time taken to analyze the diagrams were negatively impacted.

The sequence and activity diagrams metrics demonstrated inter-correlation within themselves, while the statechart diagram metrics were independent. This absence of multicollinearity in the statechart experiment allowed for more reliable multimetric modeling, unlike the sequence and activity cases where individual regression was preferred to isolate each metric's influence.

Across all three experiments, both subjective and objective analyses confirmed that diagrams deemed more complex based on the defined metrics were also rated as more difficult to analyze and took longer to analyze. These consistent patterns reinforce the robustness of the proposed metrics and highlight their potential utility in practical settings, such as complexity assessment during software design reviews.

The findings both support and extend existing research on UML analyzability. Previous work has consistently shown that increased diagram complexity can hinder understandability, a trend also observed in this study across sequence, activity, and statechart diagrams. However, unlike earlier approaches that often apply uniform metrics, this research identified differences in metric behavior depending on the type of diagram. For instance, statechart metrics were independent, while those for sequence and activity diagrams were interrelated. Such distinctions suggest that prior models may have overlooked the role of diagram-specific characteristics. Additionally, the use of a perspective-based framework in this study, combining structural and runtime factors, provides a broader view than traditional single-dimensional models. These differences may be attributed to the inclusion of multiple behavioral diagrams. Overall, the findings strengthen the theoretical foundation for using complexity metrics as reliable predictors of analyzability in UML behavioral diagrams.

8.9 Summary

This chapter presented the experimental validation of the proposed complexity metrics through controlled experiments. The study assessed the relationship between metric values and analyzability using both subjective user ratings and objective time measurements. The findings revealed consistent and statistically significant correlations, confirming that the metrics indicate the analyzability of each diagram. Regression analysis further demonstrated the predictive power of the proposed metrics.

CHAPTER NINE

CONCLUSION, RECOMMENDATIONS AND FUTURE WORK

9.1 Conclusion

This study was carried out with the main goal of developing a perspective-based metrics framework for evaluating the complexity of UML behavioral diagrams and predicting their analyzability. UML behavioral diagrams are essential tools in modern software engineering because they aid communicate the dynamic behavior of systems to both technical and non-technical stakeholders. However, as software systems grow in complexity, these diagrams also become increasingly difficult to analyze. This complexity can hinder communication, delay development, increase the likelihood of design errors, and reduce overall software quality. Given this challenge, the study focused on designing a framework and a corresponding suite of metrics that would not only reflect the true complexity of behavioral diagrams but also support the prediction of how analyzable these diagrams are. To accomplish this goal, five specific objectives were outlined and addressed systematically throughout the study.

The first objective was to identify the key structural and dynamic features that contribute to the complexity of UML behavioral diagrams. This was achieved through a comprehensive literature review, which revealed that complexity in these diagrams arises from several interrelated factors. As a result of this analysis, the study proposed three perspectives for evaluating diagram complexity: the element perspective, the control-flow perspective, and the interaction perspective. Each of these perspectives offered a different lens through which to understand and measure the structural and behavioral characteristics that influence diagram analyzability. The element perspective focused on the number and types of modeling elements, such as states, messages, and actions. The control-flow perspective examined how behavior is executed through decision paths, sequences, loops, and parallel processes. Lastly, the interaction perspective addressed how components within the system communicate with one another, capturing the flow of information and signals in the diagram. These perspectives provided a structured foundation for further metric development and offered a more holistic understanding of what makes a behavioral diagram complex.

The second objective of the study was to develop a robust validation framework that could be used to assess the quality and usefulness of any complexity metrics derived

from the perspectives. This led to the development of the Perspective-Based Metrics Validation Framework (PBMVF), a structured model that was designed to test the soundness, relevance, and practical applicability of behavioral complexity metrics. The PBMVF was informed by existing validation methodologies, particularly Kaner's evaluation questions and Briand's theoretical properties, but was modified to specifically align with the perspective-based framework defined in this study. The framework incorporated measurement principles for each complexity perspective, allowing for a systematic review of metrics based on defined attributes. For example, in the element perspective, the framework evaluated metrics in terms of the types and frequencies of elements, and their complexity weight. Similar validation properties were defined for the control-flow and interaction perspectives. The control flow evaluated the types of and frequencies of the control flow, the distance travelled by an edge, and their complexity weights. Interaction evaluated the metrics in terms of their types and frequencies of the outgoing and incoming interactions, and their complexity weights. By grounding the validation framework in the perspectives identified in the first objective, the PBMVF enabled a consistent, objective, and repeatable process for evaluating complexity metrics in UML behavioral diagrams.

The third objective was centered on the development of specific complexity metrics based on the perspective-based framework. A total of nine metrics were defined and categorized according to the three perspectives previously described. These metrics were carefully defined to ensure they were not only measurable but also aligned with the attributes and characteristics that affect diagram complexity and analyzability. Under the element perspective, five metrics were introduced: Weighted Number of State Actions (WNSA), Adjusted Weighted Number of States (AWNS), Adjusted Weighted Message into Lifeline (AWMIL), Adjusted Weighted Message out of Lifeline (AWMOL), and Total Sequence Complexity (TSC). These metrics captured different dimensions of how the elements are represented and weighted within a diagram. For the control-flow perspective, a single metric named Aggregate Control Flow Complexity (ACFC) was defined to assess the flow paths within the diagrams. The interaction perspective included three metrics: Incoming Interaction Count (IIC), Outgoing Interaction Count (OIC), and Total Interaction Count (TIC), all of which were designed to measure how messages and interactions are distributed across the diagram. The theoretical and practical validity of these metrics was tested using

multiple approaches. Each metric was first subjected to an evaluation against Weyuker's nine properties to determine whether it exhibited desirable theoretical qualities. The metrics were also reviewed using the PBMVF to confirm their logical alignment with the identified complexity perspectives. Furthermore, practical case studies involving real behavioral diagrams were used to validate the usability and intuitiveness of the metrics. These evaluations collectively confirmed that the proposed metrics were theoretically sound, practically applicable, and capable of offering meaningful insights into diagram complexity.

The fourth objective of the research focused on developing a software tool to automate the process of metric computation. The tool, named the Behavioral Complexity Measurement Tool (BCMT), was designed to process UML behavioral diagrams saved in XML format and automatically calculate the values of the proposed metrics. The tool consisted of three main components: an input module that loads and reads the diagram file, a processing module that analyzes and tokenizes the content, and an output module that displays the calculated metrics in an organized format. The development of this tool was a significant achievement because it allowed for consistent, error-free, and repeatable measurement of diagram complexity. The BCMT tool serves as a bridge between theory and practice, demonstrating how academic research on metrics can be effectively translated into usable software.

The final objective of the study was to validate the effectiveness of the defined metrics in predicting the analyzability of behavioral diagrams through empirical experimentation. A controlled laboratory experiment was designed and conducted to assess the relationship between metric values and the time required to analyze UML behavioral diagrams. The experiment involved human subjects who were asked to analyze different diagrams, with their response times and subjective evaluations of difficulty being recorded. The results showed a strong and statistically significant correlation between higher metric values and longer analysis times, indicating that diagrams with higher complexity scores were indeed more difficult to understand. Regression analysis further supported these findings by confirming that the defined metrics could be used to accurately predict analyzability outcomes. This empirical validation provided strong evidence that the proposed metrics are not only theoretically robust but also practically effective in analyzing behavioral diagrams.

The achievement of these five objectives marks a significant contribution to both the theoretical and practical aspects of software modeling. The perspective-based framework developed in this study provides a new way to understand and measure the complexity of UML behavioral diagrams by combining structural and behavioral attributes into a unified model. The validation framework ensures that metrics derived from this model are reliable, meaningful, and consistent. The defined metrics themselves offer a fine-grained and practical way to evaluate complexity, while the BCMT tool allows for automated, repeatable, and unbiased metric computation. Finally, the empirical experiment demonstrated that these metrics are reliable indicators of analyzability, establishing their value for both research and practice.

This study also has practical implications for the software engineering industry. By enabling developers and designers to assess the complexity of their models during the early stages of system development. The framework and metrics presented in this study can support better decision-making, improve model clarity, and reduce the risk of design errors. The ability to identify overly complex diagrams early on can lead to more robust systems, ultimately reducing the cost and time associated with future changes, testing, and maintenance. Moreover, the availability of an automated measurement tool makes it easier for teams to integrate complexity assessment into their development workflow without additional overhead.

In conclusion, this study has developed a comprehensive, validated, and automated approach for measuring and predicting the analyzability of UML behavioral diagrams. It has addressed a critical gap in software modeling by providing a structured framework, a set of reliable metrics, and an operational tool that collectively enhance the ability to understand, analyze, and improve system behavior models. The findings of this study contribute to improving the quality of software systems and supporting informed design practices that are grounded in empirical evidence and theoretical rigor.

9.2 Recommendations

This section gives suggestions on how the findings can be used. The recommendations are grouped into two parts: those meant for creating or improving policy, and those meant for guiding practice in real-world software development.

9.2.1 Recommendations for Policy

9.2.1.1 Integration of Perspective-Based Metrics into National Software Development Standards

One of the major steps that can be taken at the policy level is to include the perspective-based complexity metrics in national software development standards. Regulatory agencies such as the Ministry of ICT, and national standards organizations should include these metrics into technical specifications that guide software development projects. These metrics provide a clear structure for evaluating diagram complexity, allowing government and industry to align on consistent quality benchmarks. Such a standard can serve as a reference for auditors and developers, ensuring that UML behavioral diagrams meet analyzability requirements before moving into production phases. Moreover, adopting this approach will promote the reuse of design knowledge and reduce time in development of systems.

9.2.1.2 Promote Development and Use of Automated Tools through Innovation Policies

Government innovation programs, ICT authorities, and donor-funded digital transformation initiatives should actively support the development, improvement, and deployment of automated tools like the Behavioral Complexity Metrics Tool (BCMT). Policies should promote partnerships between academia and industry for co-development of such tools and provide funding or incubation spaces to allow refinement and scaling. Agencies such as innovation hubs, research funds, and software development boards can prioritize projects that incorporate quality assurance automation in software engineering.

9.2.1.3 Formal Certification and Quality Compliance Standards

Policy reforms should ensure that software development companies seeking quality certification such as ISO 9001 or ISO/IEC 25010 demonstrate the use of structured complexity evaluation in their development processes. Certification agencies, such as KEBS or global ISO auditors, should provide criteria that verify whether organizations use standard complexity metrics during design and validation phases. Such metrics can serve as part of the software quality checklist for certification. This ensures that developers are not only meeting functional requirements but also delivering software that is easy to analyze and improve in the future.

9.2.2 Recommendations for Practice

9.2.2.1 Use of the Framework in Regular Software Design Processes

Software developers, system analysts, and architects are encouraged to adopt the perspective-based metrics framework during the early stages of design. The framework supports a structured breakdown of complexity based on perspectives like control flow, element structure, and interaction, making it easier to interpret complex models. By analyzing UML behavioral diagrams using these perspectives, designers can identify sections that may be difficult to analyze. This aids improve the overall effectiveness of system documentation and design specifications, which are often used by multiple teams throughout the software development lifecycle.

9.2.2.2 Application of the BCMT Tool in Development Environments

Software development teams should integrate the BCMT tool into their modeling environments to support fast and accurate evaluations of behavioral diagram complexity. The BCMT automates metric collection, reducing manual effort and increasing precision. It can be integrated with widely used modeling software or used as a standalone tool to generate metrics reports for UML diagrams. This tool aids in identifying problematic areas early in the design phase, aiding teams avoid costly redesigns later. Additionally, by using the BCMT tool regularly, teams can monitor complexity trends across iterations and make informed decisions about simplifying their models.

9.2.2.3 Training and Professional Development for Practitioners

Organizations should invest in upskilling their staff to ensure they understand and can apply the perspective-based complexity metrics and BCMT tool effectively. This can

be done through short courses, workshops, or internal seminars. Human resource departments in software firms should consider adding UML complexity analysis to staff development plans. In addition, technical leads and quality assurance officers should be trained to interpret metric reports and use them to guide code reviews or design revisions. Training should also include guidance on integrating the framework into agile, waterfall, or hybrid development methodologies.

9.2.2.4 Feedback Mechanisms for Continuous Improvement

Development teams should establish feedback systems to evaluate the usefulness of the framework and the BCMT tool during project execution. By collecting insights from developers, testers, and users, organizations can refine how they apply complexity metrics and adjust their use of the tool to better fit their needs. These reviews can also highlight opportunities to enhance the tool's interface, add new metric types, or improve how metric results are presented. This process will ensure that the framework and tool remain practical and user-friendly.

9.2.2.5 Adoption in Agile and Model-Driven Development Environments

This framework is useful for teams using agile methods or model-driven development (MDD) by aiding them in checking if their system models are ready and easy to analyze. In these types of projects, UML diagrams are often made as part of the design stage. The complexity metrics will aid in identifying how simple or difficult a model is before development starts. In model-driven development, where code is created from diagrams, these metrics aid in assessing whether the code is simple or complex.

9.3 Future Work

Although the objectives of this study were successfully achieved, certain relevant areas were not examined, as they were outside the intended scope. These areas remain important and should be addressed in future studies to build upon the foundation laid by this work. The following suggestions outline directions that future researchers can take to improve and expand the contributions of this study.

9.3.1 Extending the Perspective-Based Complexity Framework

This study recommends extending the perspective-based complexity framework to cover additional attributes found in UML behavioral diagrams that were not addressed in the study. While the current framework focuses on selected behavioral diagrams,

other attributes also contribute to the complexity of behavioral diagrams that were not considered in the study. Future work can aim to include these additional diagrams attributes for a wider range of modeling scenarios.

9.3.2 Extending the Behavioral Complexity Metrics Tool

It is also recommended that the Behavioral Complexity Metrics Tool (BCMT) be further developed. At present, the tool is capable of computing complexity metrics for three types of behavioral diagrams: activity, statechart, and sequence diagrams. However, the tool's usefulness can be improved by extending its support to cover other UML diagram types. Future researchers could work on upgrading the tool to handle metrics for additional behavioral diagrams. This would increase the tool's practical application.

9.3.3 Defining Metrics for other Behavioral Diagrams

This study defined nine metrics specifically for the statechart, activity and sequence diagrams. However, it did not cover the other behavioral diagrams such as use case and communication diagrams.

As a result, this study suggests the development of additional metrics to evaluate these diagrams. Moreover, before this study, no single metric had been proposed based on the perspective complexity of these diagrams, highlighting a significant gap that future studies should address.

9.3.4 Conduct Further Experimentations on the Proposed Metrics

Results of this study showed that the proposed metrics are useful for analyzing the complexity of UML behavioral diagrams. However, more extensive testing and validation are required to confirm their effectiveness and reliability in practical settings. Future research should involve conducting experimental studies with experienced professionals in the software industry. Involving practitioners such as software engineers, system architects, and model designers in real-world testing of the metrics would provide more meaningful validation. . Engaging industry professionals in these experiments will enhance the credibility and acceptance of the metrics among software modelers.

REFERENCES

- [1] M. Lahon and U. Sharma, "Complexity assessment based on UML-activity diagram," *Int. J. Recent Technol. Eng.*, vol. 8, no. 2, pp. 2391–2397, 2019, doi: 10.35940/ijrte.B1596.078219.
- [2] D. I. S. Sri *et al.*, "Analyzing the impact of software complexity on software design reviews and inspections," 2023, <https://doi.org/10.21203/rs.3.rs-3059408/v1>
- [3] A. Abuhassan and M. Alshayeb, "A metrics suite for UML model stability," pp. 557–583, 2019.
- [4] T. Shailesh, A. Nayak, and D. Prasad, "Transformation of sequence diagram to timed Petri net using Atlas Transformation Language metamodel approach," no. November 2021, pp. 1–22, 2022, doi: 10.1002/smr.2412.
- [5] M. Alshayeb, H. Mumtaz, S. Mahmood, and M. Niazi, "Improving the security of UML sequence diagram using genetic algorithm," **IEEE Access**, vol. 8, pp. 62738–62761, 2020, doi: 10.1109/ACCESS.2020.2983985.
- [6] R. Gh. Alsarraj, A. M. Altaie, and A. A. Fadhil, "Designing and implementing a tool to transform source code to UML diagrams," **[Journal Name]**, vol. 9, no. 2, pp. 430–440, 2021.
- [7] M. Alshayeb, H. Mumtaz, S. Mahmood and M. Niazi, "Improving the Security of UML Sequence Diagram Using Genetic Algorithm," *IEEE Access*, vol. 8, pp. 62738–62761, 2020, doi: 10.1109/ACCESS.2020.2981742.
- [8] M. Ozkaya and F. Erata, "A survey on the practical use of UML for different software architecture viewpoints," **Information and Software Technology**, vol. 121, p. 106275, 2020, doi: 10.1016/j.infsof.2020.106275.
- [9] A. Morozov, "Automated Transformation of UML/SysML Behavioral Diagrams for Stochastic Error Propagation Analysis of Autonomous Systems," *ASME J. Comput. Inf. Sci. Eng.*, vol. 21, no. 5, pp. 1–10, Oct. 2021.
- [10] T. Shailesh, A. Nayak, and D. Prasad, "Transformation of sequence diagram to timed Petri net using Atlas Transformation Language metamodel approach,"

- [11] R. G. Alsarraj, A. M. Altaie, and A. A. Fadhil, “Designing and implementing a tool to transform source code to UML diagrams,” *Period. Eng. Nat. Sci.*, vol. 9, no. 2, pp. 430–440, 2021, doi: 10.21533/pen.v9i2.1829.
- [12] É. André, S. Liu, Y. Liu, C. Choppy, J. Sun, and J. S. Dong, “Formalizing UML State Machines for Automated Verification - A Survey,” *ACM Comput. Surv.*, vol. 55, no. 13s, pp. 1–71, 2023, doi: 10.1145/3579821.
- [13] M. Asadi, “A Model for Object-Oriented Software Maintainability Measurement,” no. January, pp. 60–66, 2016, doi: 10.5815/ijisa.2016.01.07.
- [14] G. Hatzivasilis, I. Papaefstathiou, and C. Manifavas, “Software Security , Privacy , and Dependability Metrics and Measurement,” 2016.
- [15] A. Aloysius, “Coupling Complexity Metric : A Cognitive Approach,” no. August, pp. 29–35, 2012, doi: 10.5815/ijitcs.2012.09.04.
- [16] C. Gencel, K. Petersen, A. A. Mughal, and M. I. Iqbal, “Electronic Research Archive of Blekinge Institute of Technology Title : Author : Year : Issue : Pagination : URL / DOI to the paper : A Decision Support Framework for Metrics Selection in Goal-Based Measurement Programs : GQM-DSFMS,” 2013.
- [17] S. Gehlot, “Complexity Metrics for Component Based Software — A Comparative Study,” *J. Comput.*, vol. 14, no. 6, pp. 389–396, 2019, doi: 10.17706/jcp.14.6.389-396.
- [18] B. Shannaq and R. Adebiaye, “Software Product Quality Management Methodology & the Quantitative Assessment of Analyzability Indicators,” no. May, 2020, doi: 10.35940/ijrte.F7780.038620.
- [19] S. Sharma and P. C. Pendharkar, “On the analysis of power law distribution in software component sizes,” *J. Softw. Evol. Process*, vol. 34, no. 2, pp. 1–14, 2022, doi: 10.1002/smr.2417.
- [20] U. Dayanandan and K. Vivekanandan, “An empirical evaluation model for software architecture maintainability for object oriented design,” *ACM Int.*

- Conf. Proceeding Ser.*, vol. 25-26-Aug, pp. 1–4, 2016, doi: 10.1145/2980258.2980459.
- [21] E. M. Macharia, “A Metrics-based Framework for Estimating the Maintainability of Object-Oriented Software,” *J. Inf. Eng. Appl.*, vol. 9, no. 4, pp. 12–24, 2019, doi: 10.7176/jiea/9-4-02.
 - [22] E. Tempero and P. Ralph, “A Framework for Defining Coupling Metrics,” *Sci. Comput. Program.*, no. February, 2018, doi: 10.1016/j.scico.2018.02.004.
 - [23] A. Wambui, G. M. Muketha, and E. M. Micheni, “Complexity Metrics for Statechart Diagrams,” *International Journal of Software Engineering & its Applications*, vol.13,no. 3.
 - [24] A. Gadhi, R. M. Gondu, C. M. Bandaru, K. C. Reddy, and O. Abiona, “Applying UML and Machine Learning to Enhance System Analysis and Design,” *Int. J. Commun. Netw. Syst. Sci.*, vol. 16, no. 05, pp. 67–76, 2023, doi: 10.4236/ijcns.2023.165005.
 - [25] A. Metzner, “Systematic Teaching of UML and Behavioral Diagrams,” pp. 1–5, 2024, doi: 10.1109/cseet62301.2024.10663036.
 - [26] T. Jayalath, “A modified cognitive complexity metric to improve the readability of object-oriented software,” 2020.
 - [27] B. S. Alqadi and J. I. Maletic, “Slice-Based Cognitive Complexity Metrics for Defect Prediction,” pp. 411–422, 2020.
 - [28] D. Singh and H. J. S. Sidhu, “Optimizing the Software Metrics for UML Structural and Behavioral Diagrams Using Metrics Tool,” *Asian J. Comput. Sci. Technol.*, vol. 7, no. 2, pp. 11–17, 2018, doi: 10.51983/ajcst-2018.7.2.1877.
 - [29] D. S. Suriya and N. S., “Design of UML Diagrams for WEBMED - Healthcare Service System Services,” *EAI Endorsed Trans. e-Learning*, vol. 8, no. 1, p. e5, 2023, doi: 10.4108/eetel.v8i1.3015.
 - [30] R. G. Tiwari, A. Pratap Srivastava, G. Bhardwaj, and V. Kumar, “Exploiting UML Diagrams for Test Case Generation: A Review,” *Proc. 2021 2nd Int. Conf. Intell. Eng. Manag. ICIEM 2021*, no. June, pp. 457–460, 2021, doi:

10.1109/ICIEM51511.2021.9445383.

- [31] E. Semenova, V. Tynchenko, S. Chashchina, V. Suetin, and A. Stashkevich, "Using UML to Describe the Development of Software Products Using an Object Approach," *2022 IEEE Int. IOT, Electron. Mechatronics Conf. IEMTRONICS 2022*, pp. 1–4, 2022, doi: 10.1109/IEMTRONICS55184.2022.9795777.
- [32] D. Singh and H. J. S. Sidhu, "Optimizing the Software Metrics for UML Structural and Behavioral Diagrams Using Metrics Tool," *Asian J. Comput. Sci. Technol.*, vol. 7, no. 2, pp. 11–17, 2018, doi: 10.51983/ajcst-2018.7.2.1877.
- [33] K. Sharma, S. K. Dubey, P. Gaurav, and Prachi, "Functionality Assessment of Software System using Fuzzy Approach," *ICRITO 2020 - IEEE 8th Int. Conf. Reliab. Infocom Technol. Optim. (Trends Futur. Dir.)*, pp. 1206–1209, 2020, doi: 10.1109/ICRITO48877.2020.9197795.
- [34] Z. A. H. Alnaish and S. O. Hasoon, "Software Quality Measurement Analysis based on Techniques, Criteria, Metrics, Models, and Datasets," *2022 8th Int. Conf. Contemp. Inf. Technol. Math. ICCITM 2022*, pp. 176–181, 2022, doi: 10.1109/ICCITM56309.2022.10031990.
- [35] C. Kaner and W. P. Bond, "Software Engineering Metrics : What Do They Measure and How Do We Know ?," *10Th Int. Softw. Metrics Symp. Metrics 2004*, vol. 8, pp. 1–12, 2004.
- [36] M. Martinsons, R. Davison, and D. Tse, "The balanced scorecard : a foundation for the strategic management of information systems," 1999.
- [37] T. Yue and S. Ali, "A MOF-based framework for defining metrics to measure the quality of models," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8569 LNCS, pp. 213–229, 2014, doi: 10.1007/978-3-319-09195-2_14.
- [38] V. Basili, J. Heidrich, M. Lindvall, J. Münch, M. Regardie, D. Rombach, and A. Trendowicz, "GQM+ Strategies: A comprehensive methodology for aligning business strategies with software measurement," **arXiv preprint arXiv:1402.0292**, 2014 <https://arxiv.org/abs/1402.0292>

- [39] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software engineering measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68–86, Jan. 1996, doi: [10.1109/32.481535](https://doi.org/10.1109/32.481535).
- [40] T. Yue and S. Ali, "A MOF-based framework for defining metrics to measure the quality of models," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8569 LNCS, pp. 213–229, 2014, doi: 10.1007/978-3-319-09195-2_14.
- [41] S. Rajesh and A. Chandrasekar, "Metrics measurement model: To measure the object-oriented design metrics," in 2015 Seventh International Conference on Advanced Computing (ICoAC), Chennai, India, Dec. 2015, pp. 1–6.
- [42] C. Gencel, K. Petersen, A. A. Mughal, and M. I. Iqbal, "A decision support framework for metrics selection in goal-based measurement programs: GQM-DSFMS," *J. Syst. Softw.*, vol. 86, no. 12, pp. 3091–3108, 2013, doi: 10.1016/j.jss.2013.07.022.
- [43] K. Petersen, C. Gencel, N. Asghari, and S. Betz, "Anicitation instrument for operationalising GQM + Strategies (GQM + S-EI)," 2014, doi: 10.1007/s10664-014-9306-z.
- [44] D. Amara, L. Ben, and A. Rabai, "ScienceDirect Towards a New Framework of Software Reliability Measurement Based on Software Metrics," *Procedia Comput. Sci.*, vol. 109, pp. 725–730, 2017, doi: 10.1016/j.procs.2017.05.428.
- [45] S. R. Chidamber and C. F. Kemerer, "Towards a metrics suite for object oriented design," *ACM SIGPLAN Not.*, vol. 26, no. 11, pp. 197–211, 1991, doi: 10.1145/118014.117970.
- [46] O. B. A. Agbaje, S. M. Wakil, and A. A. Osowole, "Synthesis, Spectroscopic Characterization and Antimicrobial Activities of Some Mixed Drug Metal (II) Complexes of Sulfamethoxazole and Paracetamol," *Oluwatoosin B. A. Agbaje , Sherifat M. Wakil Aderoju A. Osowole*, vol. 2014, 2014, doi: 10.5171/2014.
- [47] V. Basili *et al.*, "GQM+Strategies: A Comprehensive Methodology for Aligning Business Strategies with Software Measurement," no. MetriKon, pp. 253–266, 2014, [Online]. Available: <http://arxiv.org/abs/1402.0292>.

- [48] V. Mandić and N. Gvozdenović, "An extension of the GQM+Strategies approach with formal causal reasoning," *Inf. Softw. Technol.*, vol. 88, pp. 127–147, 2017, doi: 10.1016/j.infsof.2017.04.004.
- [49] T. Tahir and C. Gencel, "A structured goal-based measurement framework enabling traceability and prioritization," in 2010 6th International Conference on Emerging Technologies (ICET), Islamabad, Pakistan, Oct. 2010, pp. 282–286.
- [50] C. Engineering, "2ategor CognitivenComplexitygMeasure : based Wherek = isteinfomatio andre," *Control*, pp. 120–125, 2007.
- [51] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, 3rd ed. Chapman & Hall/CRC Innovations in Software Engineering and Software Development Series, 2014
- [52] J. A. McCall, P. K. Richards, and G. F. Walters, *Factors in Software Quality: Concepts and Definitions of Software Quality*. Rome Air Development Center, Air Force Systems Command, 1977.
- [53] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. McLeod, and M. Merritt, "Characteristics of Software Quality," *North Holland, Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1978.
- [54] R. B. Grady and D. L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice Hall, 1987
- [55] R. G. Dromey, *Software Product Quality: Theory, Model, and Practice*. Software Quality Institute, Brisbane, Australia, 1999.
- [56] ISO/IEC, "Software engineering — Product quality — Part 1: Quality model," ISO/IEC 9126-1:2001, 2001.
- [57] ISO/IEC, "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," ISO/IEC 25010:2011, 2011.
- [58] F. Alaswad, and E. Poovammal, Software quality prediction using machine learning. *Materials Today: Proceedings*, 2022.

- [59] D. Alawad, M. Panta, M. Zibrán, and M. R. Islam, "An empirical study of the relationships between code readability and software complexity," *27th Int. Conf. Softw. Eng. Data Eng. SEDE 2018*, pp. 122–127, 2018.
- [60] M. Sharma and G. Singh, "Analysis of Static and Dynamic Metrics for Productivity and Time Complexity," vol. 30, no. 1, pp. 7–13, 2011.
- [61] S. Pressman and B. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed., New York, NY, USA: McGraw-Hill, 2014.
- [62] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, 1989.
- [63] A. J. Albrecht, "Measuring application development productivity," in *Proceedings of the Joint IBM/SHARE/GUIDE Application Development Symposium*, 1979, pp. 83-92.
- [64] M. H. Halstead, *Elements of Software Science*. New York, NY, USA: Elsevier North-Holland, 1977.
- [65] S. Dhawan and Kiran, "Software metrics – A tool for measuring complexity," *International Journal of Software and Web Sciences (IJSWS)*, vol. 2, no. 1, pp. 4-7, 2012.
- [66] A. Caniglia, V. Dentamaro, S. Galantucci, and D. Impedovo, "FOBICS: Assessing project security level through a metrics framework that evaluates DevSecOps performance," *Information and Software Technology*, vol. 178, p. 107605, 2025..
- [67] A. K. Jakhar and K. Rajnish, "Measurement of complexity and comprehension of a program through a cognitive approach," *International Journal of Engineering, Transactions B: Applications*, vol. 28, no. 11, pp. 1579–1588, 2015. doi: 10.5829/idosi.ije.2015.28.11b.05.
- [68] M. A. A. Mamun, C. Berger, and J. Hansson, "Effects of measurements on correlations of software code metrics," *Empirical Software Engineering*, vol. 24, no. 4, pp. 2764–2818, Aug. 2019.
- [69] B. Kitchenham, S. L. Pfleeger, Z. C. Society, N. Fenton, and I. C. Society,

“Measurement Validation,” vol. 21, no. 12, 1995.

- [70] L. Arockiam and A. Aloysius, “Attribute Weighted Class Complexity : A New Metric for Measuring Cognitive Complexity of OO Systems,” *World Acad. Sci. Eng. Technol. Int. J. Comput. Electr. Autom. Control Inf. Eng.*, vol. 42 5, no. 10, pp. 1151–1156, 2011.
- [71] K. Peffers, T. Tuunanen, C. E. Gengler, M. Rossi, W. Hui, V. Virtanen, and J. Bragge, "Design Science Research Process: A Model for Producing and Presenting Information Systems Research," *arXiv preprint arXiv:2006.02763*, 2020
- [72] J. W. Njuki, "A Security Measurement Framework and Metrics for Open-Source Enterprise Resource Planning Software," Ph.D. dissertation, Murang'a University of Technology, 2022.
- [73] E. J. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1357-1365, Sep. 1988, doi: 10.1109/32.6178.
- [74] J. G. Ndia, "Structural complexity framework and metrics for analyzing the maintainability of Sassy Cascading Style Sheets," Ph.D. dissertation, Masinde Muliro Univ. of Sci. and Technol., 2019.
- [75] S. S. L. B. Abdullah Kamal, “Research Paradigm and the Philosophical Foundations of a Qualitative Study,” *PEOPLE Int. J. Soc. Sci.*, vol. 4, no. 3, pp. 1386–1394, 2019, doi: 10.20319/pijss.2019.43.13861394.
- [76] U. K. B. Dubey and D. P. Kothari, *Research Methodology: Techniques and Trends*, 1st ed. Boca Raton, FL: Chapman and Hall/CRC, 2022.
- [77] D. Makwana, P. Engineer, A. Dabhi, and H. Chudasama, "Sampling Methods in Research: A Review," *International Journal of Trend in Scientific Research and Development (IJTSRD)*, vol. 7, no. 3, pp. 762–768, Jun. 2023..
- [78] L. A. Palinkas, S. M. Horwitz, C. A. Green, J. P. Wisdom, N. Duan, and K. Hoagwood, “Purposeful sampling for qualitative data collection and analysis in mixed method implementation research,” *Implement. Sci.*, vol. 10, no. 1, p. 117,

2015.

- [79] A. Sankaran, M. Kumar, and R. Prakash, "Expert survey on software quality metrics: Challenges and recommendations," in *Proc. IEEE Int. Conf. Software Quality Assurance*, 2019, pp. 123–130.
- [80] Y. Wang, J. Li, and X. Zhang, "Validation of software metrics through expert opinion: A case study," *IEEE Trans. Software Eng.*, vol. 47, no. 5, pp. 850–862, May 2021.
- [81] H. K. Mohajan, "Two criteria for good measurements in research: Validity and reliability," *Annals of Spiru Haret University. Economic Series*, vol. 17, no. 4, pp. 59–82, 2017.
- [82] J. C. Nunnally, *Psychometric Theory*, 2nd ed. New York, NY, USA: McGraw-Hill, 1978.
- [83] A. Doğaner, "The approaches and expectations of the health sciences students towards artificial intelligence," **Karya Journal of Health Science**, vol. 2, no. 1, pp. 5–11, 2021.
- [84] M. Naderifar, H. Goli, and F. Ghaljaie, "Snowball sampling: A purposeful method of sampling in qualitative research," *Sdmej*, vol. 14, no. 3, 2017.
- [85] Y. Wang and J. Shao, "A new measure of software complexity based on cognitive weights," *Journal of Electrical and Computer Engineering*, vol. 28, no. 2, pp. 69-74, 2003.
- [86] N. K. Maina, G. M. Muketha, and G. M. Wambugu, "A new complexity metric for UML sequence diagrams," *International Journal of Software Engineering & Applications*, vol. 14, no. 1, pp. 9–22, 2023. doi: 10.5121/ijsea.2023.14102.
- [87] O. Masmali, "Towards the development of a cohesive design-driven code quality metrics," Ph.D. dissertation, The Univ. of Texas at El Paso, 2020.
- [88] C. Wohlin *et al.*, *Experimentation in Software Engineering* (2024 edition), Berlin, Germany: Springer, Mar. 2024, doi:10.1007/978-3-662-69306-3.
- [89] J. Melegati, K. Conboy, and D. Graziotin, "Qualitative surveys in software

engineering research: Definition, critical review, and guidelines,” *IEEE Trans. Softw. Eng.*, to appear 2024.

- [90] A. Alarainy, N. Madi, A. Abolkhair, A. AlMuaythir, O. Almeshari, and M. Al-Razgan, “Support mechanisms for conducting empirical research in software engineering: A systematic mapping study,” *Int. J. Comp. Apps.*, vol. x, no. y, pp. 1–20, Feb. 2025.
- [91] S. Imenda, “Is there a conceptual difference between theoretical and conceptual frameworks?,” *Journal of Social Sciences*, vol. 38, no. 2, pp. 185–195, 2014.
- [92] E. V. Sunitha and P. Samuel, "Automatic code generation from UML state chart diagrams," *IEEE Access*, vol. 7, pp. 8591-8608, 2019, doi: 10.1109/ACCESS.2019.2890970.
- [93] M. Genero, D. Miranda, and M. Piattini, "Defining metrics for UML statechart diagrams in a methodological way," in *International Conference on Conceptual Modelling*, 2003, pp. 118-128, doi: 10.1007/978-3-540-39597-3_12.
- [94] K. Sudheesh, K. N. Krishna, P. A. Krishnan, K. Susmitha, S. S. Mol, and O. Anjali, "Indoor localization," *International Journal of Engineering and Management Research (IJEMR)*, vol. 7, no. 2, pp. 120-126, 2017.
- [95] S. Suriya, "Design of UML diagrams for WEBMED-healthcare service system services," *EAI Endorsed Transactions on E-Learning*, vol. 8, no. 1, 2022.
- [96] M. Lahon and U. Sharma, "Complexity assessment based on UML-activity diagram," *International Journal of Recent Technology and Engineering*, vol. 8, no. 2, pp. 2391-2397, 2019
- [97] J. Sweller, “Cognitive load during problem solving: Effects on learning,” *Cognitive Science*, vol. 12, no. 2, pp. 257–285, 1988.
- [98] A. Paivio, *Mental Representations: A Dual Coding Approach*, Oxford University Press, 1986.
- [99] P. Johnson-Laird, *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*, Harvard University Press, 1983

- [100] N.E. Fenton and S.L. Pfleeger: Software Metrics: A Rigorous and Practical Approach, 2nd Edition Revised ed. Boston: PWS Publishing, 1997.
- [101] A. T. Nilsson and J. L. Rise, *Performance Measurements, Procedure to Design Measures – Goal Attribute Measure (GAM)*, Ericsson Quality Institute, LME/Q-93:332, Rev.E, Internal Publication, 1996.
- [102] H. Kim and C. Boldyreff, “Developing software metrics applicable to UML models,” ... *Approaches Object-Oriented Softw.* ..., no. March, 2002.
- [103] S. Misra, A. Adewumi, L. Fernandez-Sanz, and R. Damasevicius, "A suite of object oriented cognitive complexity metrics," *IEEE Access*, vol. 6, pp. 8782–8796, 2018
- [104] S. Misra and A. K. Misra, "Evaluating cognitive complexity measure with Weyuker properties," in *Proc. 3rd IEEE Int. Conf. Cognitive Informatics (ICCI)*, 2004, pp. 103–108.
- [105] D. S. Kushwaha and A. K. Misra, "Robustness analysis of cognitive information complexity measure using Weyuker properties," *ACM SIGSOFT Softw. Eng. Notes*, vol. 31, no. 1, pp. 1–6, 2006.
- [106] S. Misra, "Modified cognitive complexity measure," in *Lecture Notes in Computer Science*, vol. 4263, pp. 1050–1059, 2006, https://doi.org/10.1007/11902140_109.
- [107] S. Misra, "Cognitive program complexity measure," in *Proc. 6th IEEE Int. Conf. Cognitive Informatics (ICCI)*, Sept. 2007, pp. 120–125 .
- [108] G. M. Muketha, *Size and complexity metrics as indicators of maintainability of Business Process Execution Language process models*, Ph.D. dissertation, Universiti Putra Malaysia, 2011.
- [109] S. Misra and I. Akman, A new complexity metric based on cognitive informatics, In International Conference on Rough Sets and Knowledge Technology, Springer, Berlin, Heidelberg, pp. 620-627, 2008.
- [110] O. Esther, O. Stephen, O. Elijah, A. Rafiu, T. Dimple, and Y. Olajide, “Development of an Improved Cognitive Complexity Metrics for Object-

- Oriented Codes,” *Br. J. Math. Comput. Sci.*, vol. 18, no. 2, pp. 1–11, 2016, doi: 10.9734/bjmcs/2016/28515.
- [111] S. K. Singh and P. Singh, *Research Methodology and Statistical Techniques*. New Delhi, India: S. Chand Publishing, 2018.
- [112] M. N. Arifin and D. Siahaan, "Structural and Semantic Similarity Measurement of UML Use Case Diagram," *Lontar Komputer: Jurnal Ilmiah Teknologi Informatika*, vol. 11, no. 2, p. 88, Aug. 2020.
- [113] Z. H. Muhamad, D. A. Abdulmonim, and B. Alathari, "An Integration of UML Use Case Diagram and Activity Diagram with Z Language for Formalization of Library Management System," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 4, pp. 3069–3076, Aug. 2019.
- [114] Visual Paradigm, *"Visual Modeling and Diagramming Tools,"* Visual Paradigm, <https://www.visual-paradigm.com>.
- [115] K. Ye, F. Yan, and S. Gerasimou, "Quantitative Assurance and Synthesis of Controllers from Activity Diagrams," *arXiv preprint arXiv:2403.00169*, Mar. 2024.
- [116] M. N. Gedam and B. B. Meshram, "Proposed Secure Activity Diagram for Software Development," *International Journal of Advanced Computer Science and Applications*, vol. 14, no. 6, pp. 671–678, Jun. 2023.
- [117] S. M. A. Hossain, M. M. Rahman, and M. A. Rahman, "Automated Model-Based Test Case Generation for UML Activity Diagrams," in *Proceedings of the 2023 Conference on Software Testing*, Yerevan, Armenia, pp. 25–30.
- [118] A. Patil et al., "Development of Functional Test Cases Using FSM and UML Activity Diagrams for MDT," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 14s, pp. 12–21, Dec. 2024.
- [119] K. Ye, F. Yan, and S. Gerasimou, "Quantitative Assurance and Synthesis of Controllers from Activity Diagrams," *arXiv preprint arXiv:2403.00169*, Mar. 2024.
- [120] E. B. Belachew et al., "Towards a New Framework of Software Reliability

- Measurement Based on Software Metrics," *ResearchGate*, Jun. 2023.
- [121] M. Unterkalmsteiner et al., "A Conceptual Framework for SPI Evaluation," *arXiv preprint arXiv:2307.13089*, Jul. 2023.
- [122] R. V. Binder, "Software Dependability Measurement at the Age of 36," *arXiv preprint arXiv:2311.10039*, Nov. 2023.
- [123] F. Basciani, D. Di Pompeo, J. Di Rocco, and A. Pierantonio, "A Customizable Approach to Assess Software Quality Through Multi-Criteria Decision Making," *arXiv preprint arXiv:2301.12202*, Jan. 2023.
- [124] J. Frattini, D. Méndez, D. Méndez Fernández, and B. Penzenstadler, "Requirements Quality Research: A Harmonized Theory, Evaluation, and Roadmap," *arXiv preprint arXiv:2309.10355*, Sep. 2023.
- [125] International Organization for Standardization, "ISO/IEC 25059:2023 Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — Quality Model for AI Systems," ISO, Geneva, Switzerland, 2023.
- [126] M. Glinz, D. Méndez Fernández, and B. Penzenstadler, "Towards a Modern Quality Framework," in *Requirements Engineering: Foundation for Software Quality*, Springer, 2023, pp. 1–17.
- [127] A. Gill, "Software Metrics: Scope and Importance," *International Journal of Software Engineering and Applications*, vol. 13, no. 4, pp. 25–35, Jul. 2022.
- [128] S. Jin et al., "Software Code Quality Measurement: Implications from Metric Distributions," *arXiv preprint arXiv:2307.12082*, Jul. 2023.
- [129] N. Synovic et al., "Snapshot Metrics Are Not Enough: Analyzing Software Repositories with Longitudinal Metrics," *arXiv preprint arXiv:2207.11767*, Jul. 2022.
- [130] D. Zayour and K. El-Fakih, "Theoretical Validation of Software Complexity Metrics," *International Journal of Computer Applications*, vol. 184, no. 37, pp. 1–7, Nov. 2022.

- [131] A. Abran, J. W. Moore, P. Bourque, R. Dupuis, and L. L. Tripp, "Guide to the Software Engineering Body of Knowledge (SWEBOK)," *IEEE Computer Society*, 2021 Edition.
- [132] E. G. L. Frias and S. Soares, "Evaluation of Complexity Metrics through Theoretical Validation and Empirical Studies," *Journal of Systems and Software*, vol. 191, p. 111387, Feb. 2022.
- [133] P. C. Pendharkar, "Research and Conceptual Frameworks for Metrics-Based Software Maintenance and Evolution," *Journal of Systems and Software*, vol. 193, p. 111398, Mar. 2023.
- [134] E. B. Belachew et al., "Towards a New Framework of Software Reliability Measurement Based on Software Metrics," *ResearchGate*, Jun. 2023.
- [135] C. Sampaio et al., "Quality Assessment of Complexity Metrics through Controlled Experiments," *Proceedings of the International Conference on Software and System Processes (ICSSP)*, May 2023.
- [136] S. Cruz, F. Q. B. Silva, and L. M. P. Verissimo, "Towards a Standard Empirical Validation Framework for Software Metrics," *Empirical Software Engineering Journal*, vol. 27, no. 4, pp. 1–40, Apr. 2023.
- [137] SLOC Metrics (n.d.). Retrieved from <http://microguru.com/products/sloc>
- [138] NDepend (n.d.), (Version 6). Retrieved from 44
<http://www.ndepend.com/docs/getting-started-with-ndepend> and
<https://en.wikipedia.org/wiki/NDepend>.
- [139] Code Counter Pro (n.d.). Retrieved from <http://www.geronesoft.com>

APPENDICES

Appendix1: Expert Opinion Survey for Validating the Perspective-Based Complexity Framework

ANN WAMBUI KING'ORI

MURANG'A UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

Email: annkings2011@gmail.com

Dear Madam/Sir

RE: LETTER OF INTRODUCTION

I am a PhD (Information Technology) student at the School of Computing and Information Technology, Department of Information Technology at Murang'a University of Technology, undertaking the research topic "***A Perspective-Based Metrics Framework for Predicting the Analyzability of UML Behavioral Diagrams***". Analyzability has been defined as checking the extent to which a software system is insufficient or cause software failure. Therefore, it is critical to analyze UML behavioral diagrams based on the perspectives to identify their deficiencies. The objective of this exercise is to determine the relevance and the inclusiveness of the proposed framework in the analyzability of UML behavioral diagrams complexity.

You are selected to provide expert opinion that will aid validate the proposed framework. Your participation as an expert in the exercise will assist in the achievement of the research objectives.

Kindly note that all the data collected from the exercise will be held with utmost confidentiality and will be destroyed after the achievement of this study.

Your time and effort are highly valued and appreciated.

Sincerely,



Ann Wambui King'ori

Reg. no. SC500/5700/2021

PART A: PERSONAL INFORMATION

- 1) Please tick your highest attained level of academic qualification
☐ Bachelor's Degree ☐ Master's Degree ☐ PhD ☐ Other
- 2) Kindly tick the number of years you have served in the industry
☐ 0-1 ☐ 2-3 ☐ 4-5 ☐ 6-7 ☐ 8 and above
- 3) How do you rank your knowledge in modeling with UML diagrams? Kindly tick
☐ Extremely low ☐ low ☐ Moderate ☐ High ☐ Extremely high

PART B: RELEVANCE OF THE FRAMEWORK

NB: Relevance is the appropriateness of the framework to analyze the complexity of UML behavioral diagrams.

Although software engineers have increasingly adopted UML modeling in recent times, it is not an easy exercise as it requires years of experience to master. More specifically, UML behavioral diagrams such as statechart diagrams, sequence diagrams, and activity diagrams, etc., have inherent complexity which can be viewed from different perspectives other than the simplistic single view that currently exists. This has necessitated the development of the attached perspective-based framework that is meant to be used as a taxonomy of UML behavioral diagram complexity.

- 1) Please rate the extent to which you agree that the framework is relevant in identifying the complexity perspectives in UML behavioral diagrams by ticking (✓) in the appropriate box.

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree

PART C: INCLUSIVENESS OF THE PERSPECTIVE BASED FRAMEWORK

NB: Inclusiveness is the representation of elements that contribute to complexity of UML behavioral diagrams.

It has been asserted that the framework's element perspective, control flow perspective and interaction perspective, as well as their respective attributes as shown in the tables below and in the attached framework, form the main components in the taxonomy of UML behavioral complexity.

- 1) Please rate the extent to which you agree that the 3 attributes of element perspective namely state, action state and message contribute to the complexity of UML behavioral diagrams by ticking (✓) in the appropriate boxes.

Element perspective attributes	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree
State					
Action state					
Message					

- 2) Please rate the extent to which you agree that the 4 attributes of control flow perspective namely sequence, repetitive, decision and parallel contribute to the complexity of UML behavioral diagrams by ticking (✓) in the appropriate boxes.

Control flow perspective attributes	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree
Sequence					
Decision					
Repetitive					
Parallel					

- 3) Please rate the extent to which you agree that the 2 attributes of interaction perspective namely sequence, repetitive, decision and parallel contribute to the complexity of UML behavioral diagrams by ticking (✓) in the appropriate boxes.

Interaction perspective attributes	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree
Incoming					
Outgoing					

PART D: ADOPTION OF THE PERSPECTIVE BASED FRAMEWORK

- 1) In your opinion, to what extent do you agree that the perspective based framework can be adopted by software engineers for practice as a

complexity taxonomy for UML behavioral diagrams by ticking (✓) in the appropriate boxes.

Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly Agree

- 2) Please list any other Perspective that in your opinion has not been captured or represented well in this proposed framework.

Appendix 2: Experimental Questionnaire for Validating a Suite of Metrics for UML Behavioral Diagrams

ANN WAMBUI KING'ORI
MURANG'A UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY
P.O. BOX 75-20100
MURANG'A-KENYA.
Email: annkings2011@gmail.com

Dear Madam/Sir

RE: LETTER OF INTRODUCTION

I am a PhD (Information Technology) student at the School of Computing and Information Technology, Department of Information Technology at the Murang'a University of Technology. I am undertaking the research topic “*A Perspective-Based Metrics Framework for Predicting the Analyzability of UML Behavioral Diagrams*”. Analyzability has been defined as checking the extent to which a software system can be examined or analyzed to pinpoint issues. Therefore, analyzing UML behavioral diagrams based on the proposed metrics is critical. You are selected to provide an opinion that will aid validate the proposed metrics. Your participation in the exercise will aid achieve the research objectives.

The objective of this exercise is to determine the relationship between your analyzability of UML behavioral diagrams complexity and the proposed UML behavioral diagrams metrics.

Please read and answer ALL questions carefully. There is no right or wrong answer. If you are unsure of some questions, simply indicate your best from the provided options. You are required to tick (✓) the appropriate box where applicable. You will also be required to record certain measurements in the spaces provided.

Kindly note that all the data collected from the exercise will be held with utmost confidentiality and will be destroyed after the achievement of this study.

Your time and effort are highly valued and appreciated.

Sincerely,

A handwritten signature in black ink, appearing to read 'A King's', with a stylized flourish.

Ann Wambui King'ori

Reg. no. SC500/5700/2021

PART A: PERSONAL INFORMATION

4) Age range

☐ 20 and below ☐ 21-25 ☐ 26-30 ☐ Above 30

5) Programme of study

☐ Information Technology ☐ Software Engineering ☐ Computer Science
☐ Computer Technology

6) Year of study

☐ 3rd ☐ 4th

PART B: FAMILIARITY WITH UML BEHAVIORAL DIAGRAMS

1) How many programming languages have you done?

☐ None ☐ 1-3 ☐ 4 and above

2) How proficient are you in these languages?

☐ Low ☐ Moderate ☐ High ☐ Very high

3) Do you have any training on UML diagrams?

☐ low ☐ Moderate ☐ High ☐ Very high

4) If yes, how deep is your knowledge on UML behavioral diagrams?

☐ Low ☐ Moderate ☐ High ☐ Very high

5) Were you familiar with UML behavioral diagrams before it was introduced in this class?

☐ Yes ☐ No

PART C: OPINION ON THE COMPLEXITY OF EACH OF THE UML BEHAVIORAL DIAGRAMS

NB: Complexity refers to how easily a diagram can be examined or analyzed

You are required to enter the name of the behavioral diagram and rate its complexity by ticking (✓) in the appropriate box.

Starting time (hh:mm:ss) _____

Diagram name	Very complex (1)	Complex (2)	Neither complex nor easy (3)	Easy (4)	Very easy (5)

Ending time (hh:mm:ss) _____

**PART D: TIME TAKEN TO EXAMINE / ANALYZE EACH OF THE UML
BEHAVIORAL DIAGRAMS**

- 1) You are required to enter the time taken to analyze/ examine the behavioral
diagram in seconds _____

Appendix 3: BCMT Code

```
Imports System.IO
Imports System.Drawing.Graphics
Imports System.Drawing.Drawing2D
Imports System.Collections
Imports System.Text.RegularExpressions
Imports System.Xml

Public Class MainForm
    Dim FileReader As StreamReader
    Dim FileWriter As StreamWriter
    Dim NSS, NIS, NFS, NCS, NHS, NOS, NSMS, NAS As Integer
    Dim NDM, NSM, NAM, NRM, NLTO As Integer
    Dim TCAS, TCIS, TCFS As Integer
    Dim NS, ND, NL, NP, NG As Integer
    Dim NIM, NOM As Integer
    Private Sub openFile()
        Dim results As DialogResult
        results = OpenFileDialog1.ShowDialog
        If results = DialogResult.OK Then
            FileReader = New StreamReader(OpenFileDialog1.FileName)
            txtSource.Text = FileReader.ReadToEnd()

            ProcessNameLabel.Text =
Path.GetFileNameWithoutExtension(OpenFileDialog1.FileName)
            DirectoryLabel.Text = Path.GetDirectoryName(OpenFileDialog1.FileName)
            FileReader.Close()
        End If
    End Sub

    Public Function jParser(ByVal myFile As String) As Decimal

        Dim str As String = txtSource.Text
        Dim arr As String() = str.Split(Nothing)
        'search terms for statechart diagram
        Dim searchNSTerm As String = "<state"
        Dim searchNISTerm As String = "initial" ' state"
        Dim searchNFSTerm As String = "final" ' state"
        Dim searchNCSTerm As String = "composite" ' state"
        Dim searchNHSTerm As String = "history" ' state"
        Dim searchNOSTerm As String = "orthogonal" ' state"
        Dim searchNSMSTerm As String = "submachine" ' state"
        'Dim searchNASTerm As String = "activity"
        Dim searchNASTerm() As String = {"Activity0", "Activity1", "Activity2",
"Activity3"}
        'search terms for sequence diagram
        Dim searchNDMTerm As String = "delay"
        Dim searchNSMTerm As String = "synchronous"
        Dim searchNAMTerm As String = "asynchronous"
```

```

Dim searchNRMTerm As String = "return"
Dim searchNLTOTerm As String() = {"NLTO1", "NLTO2", "NLTO3"}
'search terms for activity diagram
Dim searchTCASTerm As String = "startEvent"
Dim searchTCISTerm As String = "<activity"
Dim searchTCFSTerm As String = "endEvent"
'search terms for control flows
Dim searchNSTerm As String = "<flow"
Dim searchNDTerm As String = "<decision"
Dim searchNLTerm As String = "<loop"
Dim searchNPSTerm As String = "parallel"
Dim searchNGTerm() As String = {"Guard1", "Guard2", "Guard3"}
'search terms for interaction
'Dim searchNIMTerm As String = "into"
Dim searchNIMTerm() As String = {"into1", "into2", "into3", "into4", "into5",
"into6"}

Dim searchNOMTerm() As String = {"outof1", "outof2", "outof3", "outof4",
"outof5", "outof6", "outof7"}
'Dim searchNOMTerm As String = "outof"
'create queries to search for statechart terms

Dim matchNSSQuery = From word1 In arr Where word1.ToLowerInvariant() =
searchNSSTerm.ToLowerInvariant() Select word1
Dim matchNISQuery = From word2 In arr Where word2.ToLowerInvariant() =
searchNISTerm.ToLowerInvariant() Select word2
Dim matchNFSQuery = From word3 In arr Where word3.ToLowerInvariant() =
searchNFSTerm.ToLowerInvariant() Select word3
Dim matchNCSQuery = From word4 In arr Where word4.ToLowerInvariant()
= searchNCSTerm.ToLowerInvariant() Select word4
Dim matchNHSQuery = From word5 In arr Where word5.ToLowerInvariant()
= searchNHSTerm.ToLowerInvariant() Select word5
Dim matchNOSQuery = From word6 In arr Where word6.ToLowerInvariant()
= searchNOSTerm.ToLowerInvariant() Select word6
Dim matchNSMSQuery = From word7 In arr Where
word7.ToLowerInvariant() = searchNSMSTerm.ToLowerInvariant() Select word7
'Dim matchNASQuery = From word8 In arr Where word8.ToLowerInvariant()
= searchNASTerm.ToLowerInvariant() Select word8
'create queries to search for sequence terms
Dim matchNDMQuery = From word9 In arr Where word9.ToLowerInvariant()
= searchNDMSTerm.ToLowerInvariant() Select word9
Dim matchNSMQuery = From word10 In arr Where
word10.ToLowerInvariant() = searchNSMSTerm.ToLowerInvariant() Select word10
Dim matchNAMQuery = From word11 In arr Where
word11.ToLowerInvariant() = searchNAMSTerm.ToLowerInvariant() Select word11
Dim matchNRMQuery = From word12 In arr Where
word12.ToLowerInvariant() = searchNRMSTerm.ToLowerInvariant() Select word12
'Dim matchNLTOQuery = From word13 In arr Where
word13.ToLowerInvariant() = searchNLTOSTerm.ToLowerInvariant() Select word13

```

```

'create queries to search for activity terms
Dim matchTCASQuery = From word14 In arr Where
word14.ToLowerInvariant() = searchTCASTerm.ToLowerInvariant() Select word14
Dim matchTCISQuery = From word15 In arr Where
word15.ToLowerInvariant() = searchTCISTerm.ToLowerInvariant() Select word15
Dim matchTCFSQuery = From word16 In arr Where
word16.ToLowerInvariant() = searchTCFSTerm.ToLowerInvariant() Select word16
'create queries to search for control flow terms
Dim matchNSQuery = From word17 In arr Where word17.ToLowerInvariant()
= searchNSTerm.ToLowerInvariant() Select word17
Dim matchNDQuery = From word18 In arr Where word18.ToLowerInvariant()
= searchNDTerm.ToLowerInvariant() Select word18
Dim matchNLQuery = From word19 In arr Where word19.ToLowerInvariant()
= searchNLTerm.ToLowerInvariant() Select word19
Dim matchNPQuery = From word20 In arr Where word20.ToLowerInvariant()
= searchNPTerm.ToLowerInvariant() Select word20
'Dim matchNGQuery = From word21 In arr Where word21.ToLowerInvariant()
= searchNGTerm.ToLowerInvariant() Select word21
'create queries to search for interaction terms
'Dim matchNIMQuery = From word22 In arr Where
word22.ToLowerInvariant() = searchNIMTerm.ToLowerInvariant() Select word22
'Dim matchNOMQuery = From word23 In arr Where
word23.ToLowerInvariant() = searchNOMTerm.ToLowerInvariant() Select word23
' Display the lowercase terms in a multiline TextBox
incomingTextBox.Clear()
incomingTextBox.Text = String.Join(Environment.NewLine, searchNIMTerm)
' Count statechart states
Dim countNSS As Integer = matchNSSQuery.Count()
NSS = countNSS
Dim countNIS As Integer = matchNISQuery.Count()
NIS = countNIS
Dim countNFS As Integer = matchNFSQuery.Count()
NFS = countNFS
Dim countNCS As Integer = matchNCSQuery.Count()
NCS = countNCS
Dim countNHS As Integer = matchNHSQuery.Count()
NHS = countNHS
Dim countNOS As Integer = matchNOSQuery.Count()
NOS = countNOS
Dim countNSMS As Integer = matchNSMSQuery.Count()
NSMS = countNSMS
'Dim countNAS As Integer = matchNASQuery.Count()
'NOS = countNAS
' Count sequence messages
Dim countNDM As Integer = matchNDMQuery.Count()
NDM = countNDM
Dim countNSM As Integer = matchNSMQuery.Count()
NSM = countNSM
Dim countNAM As Integer = matchNAMQuery.Count()

```

```

NAM = countNAM
Dim countNRM As Integer = matchNRMQuery.Count()
NRM = countNRM
'Dim countNLTO As Integer = matchNLTOQuery.Count()
' NLTO = countNLTO
' Count activity states
Dim countTCAS As Integer = matchTCASQuery.Count()
TCAS = countTCAS
Dim countTCIS As Integer = matchTCISQuery.Count()
TCIS = countTCIS
Dim countTCFS As Integer = matchTCFSQuery.Count()
TCFS = countTCFS
' Count control flows
Dim countNS As Integer = matchNSQuery.Count()
NS = countNS
Dim countND As Integer = matchNDQuery.Count()
ND = countND
Dim countNL As Integer = matchNLQuery.Count()
NL = countNL
Dim countNP As Integer = matchNPQuery.Count()
NP = countNP
' Dim countNG As Integer = matchNGQuery.Count()
'NG = countNG
'display direct metrics
simpleTextBox.Text = NSS.ToString()
InitialTextBox.Text = NIS.ToString()
finalTextBox.Text = NFS.ToString()
compositeTextBox.Text = NCS.ToString()
historyTextBox.Text = NHS.ToString()
submachineTextBox.Text = NSMS.ToString()
orthogonalTextBox.Text = NOS.ToString()
stateactivityTextBox.Text = NAS.ToString()
delayTextBox.Text = NDM.ToString()
synchronousTextBox.Text = NSM.ToString()
asynchronousTextBox.Text = NAM.ToString()
returnTextBox.Text = NRM.ToString()
lifelineTextBox.Text = NLTO.ToString()
actionstateTextBox.Text = TCAS.ToString()
actioninitialTextBox.Text = TCIS.ToString()
actionfinalTextBox.Text = TCFS.ToString()
sequenceTextBox.Text = NS.ToString()
decisionTextBox.Text = ND.ToString()
loopTextBox.Text = NL.ToString()
parallelTextBox.Text = NP.ToString()
guardTextBox.Text = NG.ToString()
incomingTextBox.Text = NIM.ToString()
outgoingTextBox.Text = NOM.ToString()
'hide textboxes
txtFactor1.Hide()
txtFactor2.Hide()

```

```

txtFactor3.Hide()
txtFactor4.Hide()
txtFactor5.Hide()
txtFactor6.Hide()
txtFactor7.Hide()
txtFactor8.Hide()
txtFactor9.Hide()
txtFactor10.Hide()
txtFactor11.Hide()
txtFactor12.Hide()
txtFactor13.Hide()
searchguard.Hide()
searchlifeline.Hide()
searchincoming.Hide()
searchoutgoin.Hide()
searchactivity.Hide()
'calculate derived metrics for statechart diagram
txtFactor1.Text = 1.0
txtFactor2.Text = 1.2
txtFactor3.Text = 1.4
txtFactor4.Text = 1.6
txtFactor5.Text = 1.5
txtFactor9.Text = 1.8
txtFactor10.Text = 2.0
Dim WNSA, AC, ASC As Decimal

```

$$ASC = (NIS * 1) + (NFS * 1) + (NCS * 2.5) + (NHS * 2) + (NSMS * 2.5) + (NOS * 1) + (NSS * 1.5)$$

```

' calculate activity complexity
' Clear previous results
searchactivity.Items.Clear()
stateactivityTextBox.Clear()
' Variable to store the total sum of multiplied values
Dim totalactivitySum As Double = 0

' Convert the target string to lowercase for case-insensitive search
Dim lowerStr2 As String = str.ToLower()
' Create an array of TextBoxes for multiplying factors
Dim multipleFactorTextBoxes() As TextBox = {txtFactor5, txtFactor2,
txtFactor3, txtFactor4}
' Loop through each search term
For i As Integer = 0 To searchNASTerm.Length - 1
    Dim term As String = searchNASTerm(i).ToLower()
    Dim count2 As Integer = 0
    Dim index2 As Integer = 0
    ' Count occurrences of the search term
    While index2 < lowerStr2.Length
        index2 = lowerStr2.IndexOf(term, index2)
        If index2 = -1 Then Exit While
        count2 += 1
    End While

```

```

        index2 += term.Length
    End While

    ' Add the search result to lstResults
    searchactivity.Items.Add(term & ": " & count2.ToString())

    ' Multiply the count by the corresponding multiplying factor from the
    TextBox
    Dim multiplier2 As Double
    If Double.TryParse(multipleFactorTextBoxes(i).Text.Trim(), multiplier2)
Then
        Dim multipliedCount As Double = count2 * multiplier2

        ' Add the multiplied result to the total sum
        totalactivitySum += multipliedCount
    Else
        MessageBox.Show("Invalid multiplying factor in TextBox " & (i +
1).ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Function
    End If
Next

    ' Display the total sum of multiplied values in the TextBox
    stateactivityTextBox.Text = totalactivitySum.ToString()
    AC = stateactivityTextBox.Text

    WNSA = ASC + AC
    'calculate derived metrics for activity diagram
    Dim AWNS As Integer
    AWNS = (TCAS * 2) + (TCIS * 1) + (TCIS * 1)
    'calculate derived metrics for sequence diagram
    'AWMIL
    Dim TSC As Decimal
    'calculate derived metrics for control flow

    Dim ACFC, CC, GC As Decimal
    CC = (ND * 2) + (NL * 3) + (NP * 4) + (NS * 1)
    ' Clear previous results
    searchguard.Items.Clear()
    guardTextBox.Clear()
    ' Variable to store the total sum of multiplied values
    Dim totalguardSum As Double = 0
    ' Convert the target string to lowercase for case-insensitive search
    Dim lowerStr3 As String = str.ToLower()

    ' Create an array of TextBoxes for multiplying factors
    Dim multipleFactor3TextBoxes() As TextBox = {txtFactor2, txtFactor3,
txtFactor4}
    ' Loop through each search term
    For i As Integer = 0 To searchNGTerm.Length - 1

```

```

Dim term As String = searchNGTerm(i).ToLower()
Dim count3 As Integer = 0
Dim index3 As Integer = 0
' Count occurrences of the search term
While index3 < lowerStr3.Length
    index3 = lowerStr3.IndexOf(term, index3)
    If index3 = -1 Then Exit While
    count3 += 1
    index3 += term.Length
End While

' Add the search result to lstResults
searchguard.Items.Add(term & ": " & count3.ToString())

' Multiply the count by the corresponding multiplying factor from the
TextBox
Dim multiplier3 As Double
If Double.TryParse(multipleFactorTextBoxes(i).Text.Trim(), multiplier3)
Then
    Dim multipliedCount As Double = count3 * multiplier3

    ' Add the multiplied result to the total sum
    totalguardSum += multipliedCount
Else
    MessageBox.Show("Invalid multiplying factor in TextBox " & (i +
1).ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    Exit Function
End If
Next

' Display the total sum of multiplied values in the TextBox
guardTextBox.Text = totalguardSum.ToString()
GC = guardTextBox.Text

ACFC = CC + GC
' calculate derived metrics for sequence complexity
' Clear previous results
searchlifeline.Items.Clear()
lifelineTextBox.Clear()
' Variable to store the total sum of multiplied values
Dim totalNLTOsum As Double = 0
' Convert the target string to lowercase for case-insensitive search
Dim lowerStr4 As String = str.ToLower()
' Create an array of TextBoxes for multiplying factors
Dim multipleFactor4TextBoxes() As TextBox = {txtFactor1, txtFactor2,
txtFactor3}
' Loop through each search term
For i As Integer = 0 To searchNLTOTerm.Length - 1
    Dim term As String = searchNLTOTerm(i).ToLower()
    Dim count4 As Integer = 0

```

```

Dim index4 As Integer = 0
' Count occurrences of the search term
While index4 < lowerStr4.Length
    index4 = lowerStr4.IndexOf(term, index4)
    If index4 = -1 Then Exit While
    count4 += 1
    index4 += term.Length
End While
' Add the search result to lstResults
searchlifeline.Items.Add(term & ": " & count4.ToString())
' Multiply the count by the corresponding multiplying factor from the
TextBox
Dim multiplier4 As Double
If Double.TryParse(multipleFactor4TextBoxes(i).Text.Trim(), multiplier4)
Then
    Dim multipliedCount As Double = count4 * multiplier4
    ' Add the multiplied result to the total sum
    totalNLTOsum += multipliedCount
Else
    MessageBox.Show("Invalid multiplying factor in TextBox " & (i +
1).ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
Exit Function
End If
Next

' Display the total sum of multiplied values in the TextBox
lifelineTextBox.Text = totalNLTOsum.ToString()
Dim Sc, TNLTO As Decimal
lifelineTextBox.Text = TNLTO
Sc = ((ND * 1.0) + (NS * 1.2) + (NRM * 1.4) + (NAS * 1.4))
TSC = (Sc + TNLTO) ^ 2
TSCTextBox.Text = TSC
'calculate derived metrics for interaction
Dim TIC, IIC, OIC As Decimal
'incoming interaction IIC
' Clear previous results
searchincoming.Items.Clear()
incomingTextBox.Clear()
' Variable to store the total sum of multiplied values
Dim totalSum As Double = 0
' Convert the target string to lowercase for case-insensitive search
Dim lowerStr As String = str.ToLower()
' Create an array of TextBoxes for multiplying factors
Dim multiplyingFactorTextBoxes() As TextBox = {txtFactor1, txtFactor2,
txtFactor3, txtFactor4, txtFactor9, txtFactor10}
' Loop through each search term
For i As Integer = 0 To searchNIMTerm.Length - 1
    Dim term As String = searchNIMTerm(i).ToLower()
    Dim count As Integer = 0
    Dim index As Integer = 0

```

```

' Count occurrences of the search term
While index < lowerStr.Length
    index = lowerStr.IndexOf(term, index)
    If index = -1 Then Exit While
    count += 1
    index += term.Length
End While
' Add the search result to lstResults
searchincoming.Items.Add(term & ": " & count.ToString())

' Multiply the count by the corresponding multiplying factor from the
TextBox
Dim multiplier As Double
If Double.TryParse(multiplyingFactorTextBoxes(i).Text.Trim(), multiplier)
Then
    Dim multipliedCount As Double = count * multiplier
    ' Add the multiplied result to the total sum
    totalSum += multipliedCount
Else
    MessageBox.Show("Invalid multiplying factor in TextBox " & (i +
1).ToString(), "Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
Exit Function
End If
Next
' Display the total sum of multiplied values in the TextBox
incomingTextBox.Text = totalSum.ToString()
IIC = incomingTextBox.Text

'calculate outgoing interaction
txtFactor5.Text = 1.5
txtFactor6.Text = 2.0
txtFactor7.Text = 2.5
txtFactor8.Text = 3.0
txtFactor11.Text = 3.5
txtFactor12.Text = 4.0
txtFactor13.Text = 4.5

' Clear previous results
searchoutgoin.Items.Clear()
outgoingTextBox.Clear()

' Variable to store the total sum of multiplied values
Dim totaloutgoingSum As Double = 0
' Convert the target string to lowercase for case-insensitive search
' Dim lowerStr As String = str.ToLower()
' Create an array of TextBoxes for multiplying factors
Dim multiplyFactorTextBoxes() As TextBox = {txtFactor5, txtFactor6,
txtFactor7, txtFactor8, txtFactor11, txtFactor12, txtFactor13}
' Loop through each search term

```

```

For i As Integer = 0 To searchNOMTerm.Length - 1
    Dim term As String = searchNOMTerm(i).ToLower()
    Dim count1 As Integer = 0
    Dim index1 As Integer = 0
    ' Count occurrences of the search term
    While index1 < lowerStr.Length
        index1 = lowerStr.IndexOf(term, index1)
        If index1 = -1 Then Exit While
        count1 += 1
        index1 += term.Length
    End While

    ' Add the search result to lstResults
    searchoutgoin.Items.Add(term & ": " & count1.ToString())

    ' Multiply the count by the corresponding multiplying factor from the
    TextBox
    Dim multiplier As Double
    If Double.TryParse(multiplyFactorTextBoxes(i).Text.Trim(), multiplier)
Then
        Dim multipliedCount1 As Double = count1 * multiplier

        ' Add the multiplied result to the total sum
        totaloutgoingSum += multipliedCount1
    Else
        MessageBox.Show("Invalid multiplyfactor in TextBox " & (i + 1).ToString(),
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        Exit Function
    End If
Next
' Display the total sum of multiplied values in the TextBox
outgoingTextBox.Text = totaloutgoingSum.ToString()
OIC = outgoingTextBox.Text
'calculate NOM

TIC = IIC + OIC
'display derived metrics
WNSATextBox.Text = WNSA.ToString()
AWNSTextBox.Text = AWNS.ToString()
'AWMILTextBox.Text = AWMIL.ToString()
' AWMOLTextBox.Text = AWMOL.ToString()
ACFCTextBox.Text = ACFC.ToString()
TSCTextBox.Text = TSC.ToString()
IICTextBox.Text = IIC.ToString()
OICTextBox.Text = OIC.ToString()
TICTextBox.Text = TIC.ToString()
End Function
Private Sub savetoFile()
    Dim sw As StreamWriter
    Dim fs As FileStream

```

```

fs = New FileStream("D:\MyNewTextFile.doc", FileMode.Create)
sw = New StreamWriter(fs)

'sw.WriteLine(Me.ProcessNameTextBox.Text & " Summary Report")
sw.WriteLine("SUMMARY REPORT")

sw.WriteLine()
sw.WriteLine("BASE METRICS" & vbTab & "VALUE")
sw.WriteLine("1. Number of simple states: " & vbTab & simpleTextBox.Text)
sw.WriteLine("2. Number of final states: " & vbTab & finalTextBox.Text)
sw.WriteLine("3. Number of initial states: " & vbTab & InitialTextBox.Text)
sw.WriteLine("4. Number of composite states: " & vbTab &
compositeTextBox.Text)
sw.WriteLine("5. Number of orthogonal states: " & vbTab &
orthogonalTextBox.Text)
sw.WriteLine("6. Number of history states: " & vbTab & historyTextBox.Text)
sw.WriteLine("7. Number of submachine states: " & vbTab &
submachineTextBox.Text)
sw.WriteLine("8. Number of activities in a state: " & vbTab &
stateactivityTextBox.Text)
sw.WriteLine("9. Number of delay messages: " & vbTab &
delayTextBox.Text)
sw.WriteLine("10. Number of synchronous messages : " & vbTab &
synchronousTextBox.Text)
sw.WriteLine("11. Number of asynchronous messages: " & vbTab &
asynchronousTextBox.Text)
sw.WriteLine("12. Number of return messages: " & vbTab &
returnTextBox.Text)
sw.WriteLine("13. Number of lifelines transmitted over: " & vbTab &
lifelineTextBox.Text)
sw.WriteLine("14. Number of action states: " & vbTab &
actionstateTextBox.Text)
sw.WriteLine("15. Number of action initial: " & vbTab &
actioninitialTextBox.Text)
sw.WriteLine("16. Number of action initial: " & vbTab &
actionfinalTextBox.Text)
sw.WriteLine("17. Number of sequence: " & vbTab & sequenceTextBox.Text)
sw.WriteLine("18. Number of decision: " & vbTab & decisionTextBox.Text)
sw.WriteLine("19. Number of loop: " & vbTab & loopTextBox.Text)
sw.WriteLine("20. Number of guards: " & vbTab & guardTextBox.Text)
sw.WriteLine("21. Number of parallel: " & vbTab & parallelTextBox.Text)
sw.WriteLine("22. Total incoming mesages: " & vbTab &
incomingTextBox.Text)
sw.WriteLine("23. Total outgoing mesages: " & vbTab &
outgoingTextBox.Text)
sw.WriteLine()
sw.WriteLine("DERIVED METRICS" & vbTab & "VALUE")
sw.WriteLine("1. Weighted number of state activities: " & vbTab &
WNSATextBox.Text)

```

```

        sw.WriteLine("2. Adjusted weighted number of state: " & vbTab &
AWNSTextBox.Text)
        sw.WriteLine("3. Adjusted weighted message into lifeline: " & vbTab &
AWMILTextBox.Text)
        sw.WriteLine("4. Adjusted weighted message out of lifeline: " & vbTab &
AWMOLTextBox.Text)
        sw.WriteLine("5. Total sequence complexity: " & vbTab & TSCTextBox.Text)
        sw.WriteLine("6. Aggregate control flow complexity: " & vbTab &
ACFCTextBox.Text)
        sw.WriteLine("7. Incoming interaction complexity: " & vbTab &
IICTextBox.Text)
        sw.WriteLine("8. Outgoing interaction complexity: " & vbTab &
OICTextBox.Text)
        sw.WriteLine("9. Total interaction complexity: " & vbTab &
TICTextBox.Text)
        sw.WriteLine()
        sw.Close()
        fs.Close()
End Sub
Private Sub saveToResults()
    rtbResults.ReadOnly = True
    rtbResults.SelectionTabs = New Integer() {100, 270}
    rtbResults.Clear()
    rtbResults.AppendText(vbCrLf & "  METRIC" & vbTab & vbTab &
"VALUE" & vbCrLf)
    rtbResults.AppendText(vbCrLf)

    ' rtbResults.AppendText("  1. Number of simple states:" & vbTab &
simpleTextBox.Text & vbCrLf)
    'rtbResults.AppendText("  2. Number of initial states:" & vbTab &
InitialTextBox.Text & vbCrLf)
    rtbResults.AppendText("1. Number of simple states: " & vbTab &
simpleTextBox.Text & vbCrLf)
    rtbResults.AppendText("2. Number of final states: " & vbTab &
finalTextBox.Text & vbCrLf)
    rtbResults.AppendText("3. Number of initial states: " & vbTab &
InitialTextBox.Text & vbCrLf)
    rtbResults.AppendText("4. Number of composite states: " & vbTab &
compositeTextBox.Text & vbCrLf)
    rtbResults.AppendText("5. Number of orthogonal states: " & vbTab &
orthogonalTextBox.Text & vbCrLf)
    rtbResults.AppendText("6. Number of history states: " & vbTab &
historyTextBox.Text & vbCrLf)
    rtbResults.AppendText("7. Number of submachine states: " & vbTab &
submachineTextBox.Text & vbCrLf)
    rtbResults.AppendText("8. Number of activities in a state: " & vbTab &
stateactivityTextBox.Text & vbCrLf)
    rtbResults.AppendText("9. Number of delay messages: " & vbTab &
delayTextBox.Text & vbCrLf)

```

```

        rtbResults.AppendText("10. Number of synchronous messages : " & vbTab &
synchronousTextBox.Text & vbCrLf)
        rtbResults.AppendText("11. Number of asynchronous messages: " & vbTab &
asynchronousTextBox.Text & vbCrLf)
        rtbResults.AppendText("12. Number of return messages: " & vbTab &
returnTextBox.Text & vbCrLf)
        rtbResults.AppendText("13. Number of lifelines transmitted over: " & vbTab &
lifelineTextBox.Text & vbCrLf)
        rtbResults.AppendText("14. Number of action states: " & vbTab &
actionstateTextBox.Text & vbCrLf)
        rtbResults.AppendText("15. Number of action initial: " & vbTab &
actioninitialTextBox.Text & vbCrLf)
        rtbResults.AppendText("16. Number of action initial: " & vbTab &
actionfinalTextBox.Text & vbCrLf)
        rtbResults.AppendText("17. Number of sequence: " & vbTab &
sequenceTextBox.Text & vbCrLf)
        rtbResults.AppendText("18. Number of decision: " & vbTab &
decisionTextBox.Text & vbCrLf)
        rtbResults.AppendText("19. Number of loop: " & vbTab & loopTextBox.Text
& vbCrLf)
        rtbResults.AppendText("20. Number of guards: " & vbTab &
guardTextBox.Text & vbCrLf)
        rtbResults.AppendText("21. Number of parallel: " & vbTab &
parallelTextBox.Text & vbCrLf)
        rtbResults.AppendText("22. Total incoming messages: " & vbTab &
incomingTextBox.Text & vbCrLf)
        rtbResults.AppendText("23. Total outgoing messages: " & vbTab &
outgoingTextBox.Text & vbCrLf)

        rtbResults.AppendText("24. Weighted number of state activities: " & vbTab &
WNSATextBox.Text & vbCrLf)
        rtbResults.AppendText("25. Adjusted weighted number of state: " & vbTab &
AWNSTextBox.Text & vbCrLf)
        rtbResults.AppendText("26. Adjusted weighted message into lifeline: " &
vbTab & AWMILTextBox.Text & vbCrLf)
        rtbResults.AppendText("27. Adjusted weighted message out of lifeline: " &
vbTab & AWMOLTextBox.Text & vbCrLf)
        rtbResults.AppendText("28. Total sequence complexity: " & vbTab &
TSCTextBox.Text & vbCrLf)
        rtbResults.AppendText("29. Aggregate control flow complexity: " & vbTab &
ACFCTextBox.Text & vbCrLf)
        rtbResults.AppendText("30. Incoming interaction complexity: " & vbTab &
IICTextBox.Text & vbCrLf)
        rtbResults.AppendText("31. Outgoing interaction complexity: " & vbTab &
OICTextBox.Text & vbCrLf)
        rtbResults.AppendText("32. Total interaction complexity: " & vbTab &
TICTextBox.Text & vbCrLf)

```

End Sub

```

Private Sub btnPrint_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs)
    PrintDocument1.Print()
End Sub

Private Sub ToolStripButton6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ToolStripButton6.Click
    ' PrintPreviewDialog1.Document = PrintDocument1
    ' PrintPreviewDialog1.ShowDialog()
    PrintDocument1.Print()
End Sub

```

```

Private Sub SaveToXMLToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
SaveToXMLToolStripMenuItem.Click

    ' if the user did not click on the OK button return. we can do nothing.
    If SaveFileDialog1.ShowDialog(Me) <> Windows.Forms.DialogResult.OK
Then
        Return
    End If

```

```

    ' get the file name.
    Dim fileName = SaveFileDialog1.FileName

```

```

    ' compile the xml.
    Dim xml = <?xml version="1.0" encoding="utf-8"?>
        <userrecord>
            <NSS><%= simpleTextBox.Text %></NSS>
            <NIS><%= InitialTextBox.Text %></NIS>
            <NFS><%= finalTextBox.Text %></NFS>
            <NCS><%= compositeTextBox.Text %></NCS>
            <NHS><%= historyTextBox.Text %></NHS>
            <NOS><%= orthogonalTextBox.Text %></NOS>
            <NSMS><%= submachineTextBox.Text %></NSMS>
            <NAS><%= stateactivityTextBox.Text %></NAS>
            <WNSA><%= WNSATextBox.Text %></WNSA>

            <NDM><%= delayTextBox.Text %></NDM>
            <NSM><%= synchronousTextBox.Text %></NSM>
            <NAM><%= asynchronousTextBox.Text %></NAM>
            <NRM><%= returnTextBox.Text %></NRM>
            <NLTO><%= lifelineTextBox.Text %></NLTO>

```

```

<AWMIL><%= AWMILTextBox.Text %></AWMIL>
<AWMOL><%= AWMOLTextBox.Text %></AWMOL>
<TSC><%= TSCTextBox.Text %></TSC>
<TCAS><%= actionstateTextBox.Text %></TCAS>
<TCIS><%= actioninitialTextBox.Text %></TCIS>
<TCFS><%= actionfinalTextBox.Text %></TCFS>
<AWNS><%= AWNSTextBox.Text %></AWNS>
<NS><%= sequenceTextBox.Text %></NS>
<ND><%= decisionTextBox.Text %></ND>
<NL><%= loopTextBox.Text %></NL>
<NP><%= parallelTextBox.Text %></NP>
<NG><%= guardTextBox.Text %></NG>
<ACFC><%= ACFCTextBox.Text %></ACFC>
<NIM><%= incomingTextBox.Text %></NIM>
<NOM><%= outgoingTextBox.Text %></NOM>
<IIC><%= IICTextBox.Text %></IIC>
<OIC><%= OICTextBox.Text %></OIC>
<TIC><%= TICTextBox.Text %></TIC>

```

```

</userrecord>

```

```

' write the xml to the file.

```

```

xml.Save(fileName)

```

```

' show a message box with the results.

```

```

MessageBox.Show("The file has been successfully written to: " + fileName,
"Success!")

```

```

End Sub

```

```

Private Sub SaveToXMLToolStripMenuItem_Click_1(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles

```

```

SaveToXMLToolStripMenuItem.Click

```

```

    PrintPreviewDialog1.Document = PrintDocument1

```

```

    PrintPreviewDialog1.ShowDialog()

```

```

End Sub

```

```

Private Sub ToolStripButton1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ToolStripButton1.Click

```

```

    openFile()

```

```

End Sub

```

```

Private Sub ToolStripButton2_Click_1(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ToolStripButton2.Click

```

```

    jParser(txtSource.Text)

```

```

    saveToResults()

```

```

End Sub

```

```

Private Sub ToolStripButton3_Click_1(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ToolStripButton3.Click

```

```

    savetoFile()

```

```

End Sub

```

```
Private Sub ToolStripButton4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ToolStripButton4.Click
```

```
    lifelineTextBox.Clear()
    simpleTextBox.Clear()
    InitialTextBox.Clear()
    finalTextBox.Clear()
    compositeTextBox.Clear()
    historyTextBox.Clear()
    orthogonalTextBox.Clear()
    submachineTextBox.Clear()
    stateactivityTextBox.Clear()
    WNSATextBox.Clear()

    delayTextBox.Clear()
    synchronousTextBox.Clear()
    asynchronousTextBox.Clear()
    returnTextBox.Clear()
    lifelineTextBox.Clear()
    AWMILTextBox.Clear()
    AWMOLTextBox.Clear()
    TSCTextBox.Clear()
```

```
    actionstateTextBox.Clear()
    actioninitialTextBox.Clear()
    actionfinalTextBox.Clear()
    AWNSTextBox.Clear()
```

```
    sequenceTextBox.Clear()
    decisionTextBox.Clear()
    loopTextBox.Clear()
    parallelTextBox.Clear()
    guardTextBox.Clear()
    ACFCTextBox.Clear()
```

```
    incomingTextBox.Clear()
    outgoingTextBox.Clear()
    IICTextBox.Clear()
    OICTextBox.Clear()
    TICTextBox.Clear()
    rtbResults.Text = ""
    txtSource.Clear()
```

```
    'If Not (PBarChart.Image Is Nothing) Then
    'PBarChart.Image.Dispose()
    'PBarChart.Image = Nothing
    ' End If
```

```
End Sub
```

```
Private Sub OpenToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles OpenToolStripMenuItem.Click
    openFile()
```

```

End Sub
Private Sub CalculateToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CalculateToolStripMenuItem.Click
    jParser(txtSource.Text)
    saveToResults()
End Sub

Private Sub SaveAsTextToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
SaveAsTextToolStripMenuItem.Click
    savetoFile()
End Sub
Private Sub ClearToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ClearToolStripMenuItem.Click
    lifelineTextBox.Clear()
    simpleTextBox.Clear()
    InitialTextBox.Clear()
    finalTextBox.Clear()
    compositeTextBox.Clear()
    historyTextBox.Clear()
    orthogonalTextBox.Clear()
    submachineTextBox.Clear()
    stateactivityTextBox.Clear()
    WNSATextBox.Clear()
    delayTextBox.Clear()
    synchronousTextBox.Clear()
    asynchronousTextBox.Clear()
    returnTextBox.Clear()
    lifelineTextBox.Clear()
    AWMILTextBox.Clear()
    AWMOLTextBox.Clear()
    TSCTextBox.Clear()
    actionstateTextBox.Clear()
    actioninitialTextBox.Clear()
    actionfinalTextBox.Clear()
    AWNSTextBox.Clear()
    sequenceTextBox.Clear()
    decisionTextBox.Clear()
    loopTextBox.Clear()
    parallelTextBox.Clear()
    guardTextBox.Clear()
    ACFCTextBox.Clear()
    incomingTextBox.Clear()
    outgoingTextBox.Clear()
    IICTextBox.Clear()
    OICTextBox.Clear()
    TICTextBox.Clear()
    rtbResults.Text = ""
    txtSource.Clear()

```

```

        'If Not (PBBarChart.Image Is Nothing) Then
        'PBBarChart.Image.Dispose()
        'PBBarChart.Image = Nothing
        'End If
    End Sub
    Private Sub PrintToolStripMenuItem_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles PrintToolStripMenuItem.Click
        ' PrintPreviewDialog1.Document = PrintDocument1
        'PrintPreviewDialog1.ShowDialog()
        'PrintDocument1.Print()
    End Sub
    Private Sub QuitApplicationToolStripMenuItem_Click(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
QuitApplicationToolStripMenuItem.Click
        Me.Close()
    End Sub
End Class

```

Appendix 4: MUT Research Authorization Letter



MURANG'A UNIVERSITY OF TECHNOLOGY

Office of the Director, Postgraduate Studies

E-mail: bps@mut.ac.ke

Website: www.mut.ac.ke

P.O. Box 75-10200

Murang'a, Kenya

Ref: MUT/ARP/PGS/19/2020 /VOL.1

Date: 19th May, 2023

Ms. Ann Wambui King'ori (SC500/5700/2021),
School of Computing and Information Technology,
Murang'a University of Technology,
P.O. Box 75-10200
Murang'a.

Dear Ms. King'ori,

RE: APPROVAL OF RESEARCH PROPOSAL AND SUPERVISORS

I am pleased to inform you that the Directorate of Postgraduate Studies on 25th March, 2023 considered and approved your Doctoral research proposal entitled "*A Perspective Based Metrics Framework for Predicting the Analyzability of UML Behavioral Diagrams*" and appointed the following as supervisors:

1. Prof. Geoffrey Muchiri - Murang'a University of Technology
2. Dr. John Ndia - Murang'a University of Technology

You may now proceed with your data collection subject to obtaining research permit from NACOSTI, if required. You should also begin consulting your supervisors and submit through them quarterly progress reports to the Director Postgraduate Studies through your CoD and School Dean. Progress Reports forms can be accessed in the University Website.

It is the policy and regulations of the University that you observe deadlines. The guidelines on Postgraduate supervision can be accessed in the post graduate Handbook.

Your responsibilities as a student will include, among others;

- i. Maintain regular consultation with your supervisor(s), at least once a month
- ii. Submit quarterly reports on time, through your supervisors, CoD, Dean and to the Director of Postgraduate Studies;
- iii. Ensure quality work all through;



MUT IS ISO 9001: 2015 & ISO/IEC 27001:2013 CERTIFIED



- iv. Present your research findings at 2-3 seminars/conferences prior to thesis examination.
- v. Publish two articles from your research findings in a refereed journal prior to thesis examination.

For any further clarification please contact the undersigned.

Yours Sincerely,



Prof. Geoffrey Muchiri, PhD

Director, Postgraduate Studies

GM/MMK

Copy to:

- 1. Registrar, Academics, Research and Students Affairs
- 2. Dean, School of Computing and Information Technology



MUT IS ISO 9001: 2015 & ISO/IEC 27001:2013 CERTIFIED



Appendix 5: NACOSTI Research Permit (Period: 30TH June 2023 TO 30TH June 2024)

 <p>REPUBLIC OF KENYA</p>		 <p>NATIONAL COMMISSION FOR SCIENCE, TECHNOLOGY & INNOVATION</p>	
<p>Ref No: 638039</p>		<p>Date of Issue: 30 June 2023</p>	
<h3>RESEARCH LICENSE</h3>			
			
<p>This is to Certify that Mrs. ANN WAMBUI KING'OREI of Murang'a University of Technology, has been licensed to conduct research as per the provision of the Science, Technology and Innovation Act, 2012 (Rev.2014) in Murang'a on the topic: PERSPECTIVE-BASED METRICS FRAMEWORK FOR PREDICTING THE ANALYZABILITY OF UML BEHAVIORAL DIAGRAMS for the period ending : 30 June 2024.</p>			
<p>License No: NACOSTI/P/23/27183</p>		<p>Applicant Identification Number 638039</p>	
<p>Director General</p>		<p>Verification QR Code</p>	
<p>NATIONAL COMMISSION FOR SCIENCE, TECHNOLOGY & INNOVATION</p>			
<p>NOTE: This is a computer generated License. To verify the authenticity of this document, Scan the QR Code using QR scanner application.</p>			
<p>See overleaf for conditions</p>			

Appendix 6: NACOSTI Research Permit (Period: 20TH October 2023 TO 20TH October 2025)

 <p>REPUBLIC OF KENYA</p>	 <p>NATIONAL COMMISSION FOR SCIENCE, TECHNOLOGY & INNOVATION</p>
<p>Ref No: 693953</p>	<p>Date of Issue: 29 October 2024</p>
<p>RESEARCH LICENSE</p>	
	
<p>This is to Certify that Mwalimu ANN WAMBUI KING'ORI of Maraga's University of Technology, has been licensed to conduct research as per the provision of the Science, Technology and Innovation Act, 2013 (Rev.2014) in Maraga on the topic: A PERSPECTIVE-BASED METRICS FRAMEWORK FOR PREDICTING THE ANALYZABILITY OF UML BEHAVIORAL DIAGRAMs for the period ending : 29 October 2025.</p>	
<p>License No: NACOSTI/24/61465</p>	
<p>Applicant Identification Number: 693953</p>	
<p>Director General NATIONAL COMMISSION FOR SCIENCE, TECHNOLOGY & INNOVATION</p>	
<p>Verification QR Code</p> 	
<p>NOTE: This is a computer generated License. To verify the authenticity of this document, Scan the QR Code using QR scanner application.</p>	
<p>See overleaf for conditions</p>	

Appendix 7: Publications

The following papers have been published from this thesis:

1. W. King'ori, G. M. Muketha, and J. G. Ndia, "A Suite of Metrics for UML Behavioral Diagrams Based on Complexity Perspectives," *Int. J. Softw. Eng. Appl. (IJSEA)*, vol. 15, no. 2, 2024, doi.org/10.5121/ijsea.2024.15201
2. W. King'ori, G. M. Muketha, and J. G. Ndia, "A Framework for Analyzing UML Behavioral Metrics Based on Complexity Perspectives," *Int. J. Softw. Eng. (IJSE)*, vol. 11, no. 1, 2024.
3. W. King'ori, G. M. Muketha, and J. G. Ndia, "A Perspective-based Complexity Analysis Framework for UML Behavioral Diagrams," *Int. J. Comput. Appl.*, vol. 186, no. 71, pp. 52–59, Mar. 2025, doi.org/10.5120/ijca2025924560