

A UNIFIED METAMODEL FOR ASSESSING AND PREDICTING SOFTWARE

EVOLVABILITY QUALITY

ASEEL HMOOD

A THESIS

IN THE DEPARTMENT OF

COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY (COMPUTER SCIENCE)

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

OCTOBER 2013

© ASEEL HMOOD, 2013

CONCORDIA UNIVERSITY  
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By:                   Aseel Hmood

Entitled:            A Unified Metamodel for Assessing and Predicting Software Evolvability Quality

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

DOCTOR OF PHILOSOPHY (Computer Science)

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

Signed by the final examining committee:

\_\_\_\_\_ Chair  
Dr. R. Dssouli

\_\_\_\_\_ External Examiner  
Dr. D. Amyot

\_\_\_\_\_ External to Program  
Dr. A. Hamou-Lhadj

\_\_\_\_\_ Examiner  
Dr. P. Grogono

\_\_\_\_\_ Examiner  
Dr. R. Witte

\_\_\_\_\_ Thesis Supervisor

Dr. J. Rilling

Approved by \_\_\_\_\_

Dr. V. Haarslev, Graduate Program Director

December 12, 2013

\_\_\_\_\_  
Dr. Christopher W. Trueman, Dean  
Faculty of Engineering & Computer Science

## ABSTRACT

### A Unified Metamodel for Assessing and Predicting Software Evolvability Quality

Aseel Hmood, PhD

Concordia University, 2013

Software quality is a key assessment factor for organizations to determine the ability of software ecosystems to meet the constantly changing requirements. Many quality models exist that capture and assess the changing factors affecting the quality of a software product. Common to these models is that they, contrary to the software ecosystems they are assessing, are not evolvable or reusable. The thesis first defines what constitutes a unified, evolvable, and reusable quality metamodel. We then introduce SE-EQUAM, a novel, ontological, quality assessment metamodel that was designed from the ground up to support quality unification, reuse, and evolvability. We then validate the reusability of our metamodel through instantiating a domain specific quality assessment model called OntEQAM that assesses evolvability as a non-functional software quality based on product and community dimensions. A fuzzy logic based assessment process that addresses uncertainties around score boundaries supports the evolvability quality assessment. The presented assessment process also uses the unified representation of the input knowledge artifacts, the metamodel, and the model to provide a fuzzy assessment score. Finally, we further interpret and predict the evolvability assessment scores using a novel, cross-disciplinary approach that re-applies financial technical analysis, which are indicators, and patterns typically used for price analysis and the forecasting of stocks in financial markets. We performed several case studies to illustrate and evaluate the applicability of our proposed evolvability score prediction approach.

## ACKNOWLEDGEMENT

First and foremost I would like to deeply thank my supervisor Professor Juergen Rilling without his guidance, support and patience this work would not have happened. Throughout this journey, Professor Rilling was always available to openly discuss new research ideas, ask challenging questions and therefore elevate the work to the best it can be.

I would like to thank my committee members, Professor Daniel Amyot (University of Ottawa), Professor Peter Grogono, Professor René Witte, and Professor Abdelwahab Hamou-Lhadj, whose work and valuable feedback expanded my horizon and helped me discover new research opportunities that significantly improved my research.

In addition, a thank you to all my colleagues and co-authors at Concordia University: Dr. Philipp Schugerl, David Walsh, Ninus Khamis, Mostafa Erfani, Chris Forbes, Chris Neil, and George Peristerakis.

A special thanks to a unique friend and colleague Dr. Iman Keivnloo who was there for the past four years to discuss and share research ideas, challenges, and literature. Our discussions led to new published work and more research opportunities.

Last and not least, I want to express my gratitude and appreciation to my father Professor Khadair Hmood who believed in me and supported me through all the steps of my journey. His advice and words of wisdom were the light to guide me and help me see the glory to come and the payoff for what I have invested. Thank you for your unconditional love and for giving me the chance to improve myself. Thanks to my mother, Intesar Al-Naib, from whom I learnt the patience, tolerance, and openness and finally thanks to my siblings: Ala'a, Ali, Jaffar, Zainab, Hadeel, and Fatima for being there for me and for understanding my absence sometimes.

# TABLE OF CONTENTS

ABSTRACT .....	II
LIST OF FIGURES .....	VII
LIST OF TABLES .....	X
GLOSSARY .....	XI
CHAPTER 1: INTRODUCTION .....	1
1.1    MOTIVATION .....	1
1.2    RESEARCH APPROACH .....	3
1.3    ISSUES NOT ADDRESSED IN THIS THESIS .....	6
1.4    THESIS OUTLINE .....	7
CHAPTER 2: BACKGROUND .....	9
2.1    EVOLVABILITY DEFINED .....	9
2.2    QUALITY ASSESSMENT MODELS .....	10
2.3    TRADITIONAL ASSESSMENT MODELS .....	14
2.4    OPEN SOURCE ASSESSMENT MODELS .....	21
2.5    SOFTWARE QUALITY ASSESSMENT PROCESS .....	36
2.6    REPOSITORY OF REPOSITORIES .....	50
2.7    ONTOLOGIES AND MINING SOFTWARE REPOSITORIES .....	51
2.8    SUMMARY .....	56
CHAPTER 3: SE-EQUAM, AN EVOLVABLE QUALITY METAMODEL .....	62
3.1.    METAMODEL REUSABILITY REQUIREMENTS .....	63
3.2.    KNOWLEDGE MODELING REQUIREMENT .....	71
3.3.    KNOWLEDGE POPULATION REQUIREMENT .....	78
3.4.    KNOWLEDGE EXPLORATION REQUIREMENT .....	80
3.5.    SE-EQUAM RELATED WORK .....	82
3.6.    SUMMARY .....	89
CHAPTER 4: ONTEQAM: ONTOLOGICAL EVOLVABILITY QUALITY ASSESSMENT MODEL .....	91
4.1.    ONTEQAM ADDRESSES METAMODEL REUSE REQUIREMENTS .....	93
4.2.    ONTEQAM ADDRESSES KNOWLEDGE MODELING REQUIREMENT .....	100
4.3.    ONTEQAM ADDRESSES THE KNOWLEDGE POPULATION REQUIREMENT .....	105

4.4.	ONTEQAM ADDRESSES THE KNOWLEDGE EXPLORATION REQUIREMENT.....	111
CHAPTER 5: THE FUZZY LOGIC BASED ASSESSMENT PROCESS .....		115
5.1.	ASSESSMENT PROCESS .....	117
5.2.	CASE STUDY.....	130
5.3.	VALIDATION AGAINST QUALOSS .....	135
CHAPTER 6: EVOLVABILITY PREDICTION USING FINANCIAL TECHNICAL ANALYSIS.....		141
6.1	SOFTWARE QUALITY PREDICTION MODELS .....	141
6.2	FINANCIAL TECHNICAL ANALYSIS.....	145
6.3	SUMMARY .....	155
CHAPTER 7: EVOLVABILITY SCORE INTERPRETATION AND PREDICTION.....		156
7.1	FINANCIAL TECHNICAL ANALYSIS PROCESS.....	157
7.2	CASE STUDY- PREDICTING SOFTWARE EVOLVABILITY QUALITIES.....	159
7.3	SUMMARY .....	180
CHAPTER 8: CONCLUSION AND FUTURE WORK.....		182
8.1	REVISITING OUR RESEARCH QUESTIONS .....	182
8.2	THREAT TO VALIDITY .....	184
8.3	FUTURE WORK.....	186
REFERENCES.....		188

## LIST OF FIGURES

FIGURE 1 RESEACH APPROACH.....	4
FIGURE 2 QUALITY ASSESSMENT COMPONENTS .....	7
FIGURE 3 EXTRACTED (SYNTACTIC) QUALITY ASSESSMENT METAMODEL .....	12
FIGURE 4 MCCALL QUALITY MODEL .....	16
FIGURE 5 BOEHM QUALITY MODEL .....	17
FIGURE 6 ISO/IEC 9126-1 QUALITY MODEL .....	18
FIGURE 7 PRODUCT QUALITY MODEL AS DEFINED BY THE ISO/IEC 25010:2011 .....	19
FIGURE 8 QUALITY IN USE MODEL AS DEFINED BY THE ISO/IEC 25010:2011 .....	19
FIGURE 9 DROMEY QUALITY MODEL [64] .....	20
FIGURE 10 NASA SATC QUALITY MODEL.....	21
FIGURE 11 QSOS QUALITY MODEL.....	23
FIGURE 12 CAPGEMINI OSMM QUALITY MODEL .....	24
FIGURE 13 OPENBRR QUALITY MODEL.....	25
FIGURE 14 SQO-OSS QUALITY MODEL.....	27
FIGURE 15 QUALOSS QUALITY MODEL.....	28
FIGURE 16 SIG MAINTAINABILITY MODEL.....	29
FIGURE 17 SQUALE QUALITY MODEL [68].....	31
FIGURE 18 SQUALE QUALITY MODEL PYRAMID [70].....	33
FIGURE 19 SAMPLE OF NONREMEDATION FACTORS ISSUED FROM A SPECIFIC CONTEXT [70] .....	33
FIGURE 20 FURPS+ QUALITY MODEL.....	34
FIGURE 21 QUALOSS ASSESSMENT PROCESS STEPS.....	39
FIGURE 22 QSOS ASSESSMENT PROCESS STEPS .....	41
FIGURE 23 THE CAPGEMINI OSMM AXES [57].....	42
FIGURE 24 CAPGEMINI OSMM ASSESSMENT PROCESS STEPS [57].....	43
FIGURE 25 SMM ASSESSMENT PROCESS STEPS [66] .....	45
FIGURE 26 SMM MEASURE BENCHMARKING [64].....	45
FIGURE 27 SQUALE ASSESSMENT PROCESS STEPS [69, 82] .....	46

FIGURE 28 SQALE SCALE [82] .....	47
FIGURE 29 LEFT, MODAL, AND RIGHT BOUNDARIES.....	48
FIGURE 30 FUZZY LOGIC BASES ASSESSMENT PROCESS STEPS .....	48
FIGURE 31 FUZZIFICATION OF AVERAGE CYCLOMATIC COMPLEXITY [84] .....	49
FIGURE 32 FUZZIFICATION OF OUTPUT VARIABLE – MAINTAINABILITY [84].....	49
FIGURE 33 URI GENERATION SCHEMA [94] .....	53
FIGURE 34 SE-EQUAM REUSABILITY STEPS.....	65
FIGURE 35 RELATIONSHIPS BETWEEN MODELS, METAMODELS, AND ONTOLOGY [123].....	66
FIGURE 36 SYNTACTIC METAMODEL AND DOMAIN MODLE INSTANCE (ISO/IEC 9126) EXAMPLE .....	67
FIGURE 37 SEMANTIC MAPPING TO ONTOLOGY METAMODEL .....	68
FIGURE 38 SAMPLE SQALE REUSE OF QUAMON .....	69
FIGURE 39 QUAMON REUSABILITY USING A DOMAIN MODEL ONTOLOGYEXAMPLE (SUBSET).....	70
FIGURE 40 QUAMON- METAMODEL ONTOLOGY (SUBSET) .....	72
FIGURE 41 SAMPLE SET OF URI GENERATION SCHEMA APPLICATION .....	79
FIGURE 42 ASSERTED VERSUS INFERRED QUERY KNOWLEDGE .....	81
FIGURE 43 SOFTWARE MEASUREMENT ONTOLOGY (SMO) [159].....	85
FIGURE 44 SMO DECISION CRITERIA [159] .....	87
FIGURE 45 SMO-II (SUBSET) [126] .....	88
FIGURE 46 ONTOLOGY MODELING FOR SOFTWARE EVALUATIONS [165] .....	89
FIGURE 47 REUSE OF SE-EQUAM METAMODEL TO INSTANTIATE A DOMAIN MODEL ONTOLOGY (ONTEQAM)[12] .....	92
FIGURE 48 THE SYNTACTIC METAMODEL AND ITS INSTANCE, THE DOMAIN MODEL (ONTEQAM) .....	94
FIGURE 49 SE-EQUAM SEMANTIC METAMODEL AND ITS INSTANCE, THE DOMAIN MODEL (ONTEQAM) .....	95
FIGURE 50 ONTEQAM DOMAIN MODEL RESUES QUAMON METAMODEL ONTOLOGY (SUBSET).....	97
FIGURE 51 HASMEASURE-RELATED PROPERTY CHAIN CONSTRUCTS (PROTEGE VIEW) .....	99
FIGURE 52 SE-EQUAM KNOWLEDGE ARTIFACTS.....	101
FIGURE 53 ONTOLOGY POPULATION (ONTOLOGICAL KNOWLEDGE MODEL).....	107
FIGURE 54 SEMANTICALLY RICHER KNOWLEDGE EXPLORATION .....	114
FIGURE 55 SIG MAINTAINABILITY MODEL BENCHMARKING APPROACH.....	115
FIGURE 56 SQALE SCALE [70] .....	116
FIGURE 57 FUZZY ASSESSMENT PROCESS STEPS.....	118



FIGURE 58 MEASURE FUZZY SCALE .....	123
FIGURE 59 WEIGHT FUZZY SCALE .....	126
FIGURE 60 QUALITY SCORE FUZZY SCALE.....	126
FIGURE 61 INFERENCE SCORE RESULTS FOR A GIVEN MEASURE AND WEIGHT .....	128
FIGURE 62 PMD NO. OF CONTRIBUTOR'S EVOLVABILITY SCORES.....	133
FIGURE 63 GOOGLE INC. 20/50 DAYS MA (6 MONTHS RANGE) .....	149
FIGURE 64 GOOGLE INC. 20 AND 50 DAYS MA CROSSOVER <sup>79</sup> .....	150
FIGURE 65 GOOGLE INC. 20 DAYS MA, INCLUDING SUPPORT AND RESISTANCE LINES <sup>79</sup> .....	151
FIGURE 66 GOOGLE INC. 6 MONTHS MACD.....	152
FIGURE 67 FINANCIAL TECHNICAL ANALYSIS PROCESS .....	158
FIGURE 68 M-TOP & W-BOTTOM PATTERNS .....	163
FIGURE 69 SAMPLE PATTERN DETECTION (BLACK IS A PATTERN AND RED IS NOT A PATTERN).....	163
FIGURE 70 W-BOTTOM AND M-TOP ROOT AND EVOLUTION PATTERNS.....	165
FIGURE 71 ARGOUML EVOLUTION PATTERNS 2008-2011.....	168
FIGURE 72 PULSE- PATTERNS ANNOTATION.....	168
FIGURE 73 PULSE CROSSOVER AND BREAKOUT PATTERNS .....	171
FIGURE 74 ARGOUML- RESISTANCE BREAKOUT PATTERN /BULLISH CROSSOVER CAUSES A PATTERN REVERSAL .....	172
FIGURE 75 PULSE- MA CROSSOVER PATTERN AND M-TOPS PATTERN CORRELATION .....	175
FIGURE 76 20-MA, 50-MA AND MA CROSSOVER PATTERNS VERSUS LINEAR REGRESSION .....	176

## LIST OF TABLES

TABLE 1 OVERVIEW OF SOFTWARE MAINTENANCE COST TO TOTALLIFECYCLE COST [6].....	2
TABLE 2 SQO-OSS MAINTANABILITY QUALITY ASSESSMENT SCORE (SUBSET) [42] .....	44
TABLE 3 COMPARATIVE STUDY FOR EXISTING QUALITY ASSESSMENT MODELS .....	58
TABLE 4 DEFINITIONS OF QUAMON CONCEPTS .....	73
TABLE 5 DEFINITIONS OF QUAMON ATTRIBUTES.....	74
TABLE 6 DEFINITIONS OF QUAMON RELATIONSHIPS.....	74
TABLE 7 OWL2 PROPERTY CHAIN USAGE.....	77
TABLE 8 DEFINITIONS OF TOOLON CONCEPTS .....	102
TABLE 9 DEFINITIONS OF TOOLON RELATIONSHIPS .....	102
TABLE 10 ESTABLISHING TRACEABILITY LINKS USING SPARQL AND INFERENCE RULES.....	105
TABLE 11 ONTEQAM MEASURES ASSIGNMENTS (SUBSET) .....	119
TABLE 12 SAMPLE SPARQL QUERIES FOR MEASURES CALCULATION.....	122
TABLE 13 SAMPLE FUZZY CONTROL LANGUAGE (FCL) FILE.....	127
TABLE 14 PMD MEASURE VALUES, SCORES, AND FIRED RULES .....	133
TABLE 15 BASE MEASURES.....	136
TABLE 16 ONTEQAM LOCAL & GLOBAL SCORES VERSUS QUALOSS (THREE OSS PROJECTS).....	138
TABLE 17 A SUMMARY OF TECHNICAL INDICATORS AND THEIR ASSOCIATED PATTERNS .....	146
TABLE 18 W-BOTTOM ROOT PATTERNS EVOLUTION .....	154
TABLE 19 M-TOP ROOT PATTERNS EVOLUTION .....	154
TABLE 20 SUMMARY OF TECHNICAL INDICATORS AND ASSOCIATED PATTERNS.....	157
TABLE 21 M-TOP PATTERN APPLICABILITY (ACROSS ALL PROJECTS).....	160
TABLE 22 THE MAPPING BETWEEN STOCK PRICE AND EVOLVABILITY SCORE PATTERN INTERPRETATION.....	166
TABLE 23 W-BOTTOM PATTERN APPLICABILITY (ACROSS ALL PROJECTS).....	169
TABLE 24 M-TOP PATTERN APPLICABILITY (ACROSS ALL PROJECTS).....	169
TABLE 25 QUANTITATIVE ANALYSIS OF MA-CROSSOVER AND SUPPORT/RESISTANCE LINES.....	173
TABLE 26 QUANTITATIVE ANALYSIS OF MA-CROSSOVERS AND SUPPORT/RESISTANCE PER INDIVIDUAL PROJECTS .....	174
TABLE 27 ACTUAL VERSUS PREDICTED SCORE FOR THE LAST THREE DATA POINTS USING 20-MA AND LR.....	177
TABLE 28 PERCENT PREDICTION ERROR RELATIVE TO THE TOTAL DATASET PER PROJECT (USING 20-MA) .....	178
TABLE 29 PERCENT PREDICTION ERROR RELATIVE TO THE TOTAL DATASET PER PROJECT (USING LR).....	178

## GLOSSARY

Below is the list of definitions and acronyms used throughout the thesis:

**Model:** In our context, a model is used to refer to a quality model. According to ISO/IEC 14598 [33] standard, a quality model is defined as: “The set of characteristics and relationships between them, which provides the basis for specifying quality requirements and evaluating quality”

**Metamodel:** an abstraction, template, or frame that consists of a collection of concepts and relationships. A model is an instance of a metamodel.

**Score:** it is also called a rank or a rate. Given a quality to assess and a weight of its importance, it refers to the quality assessment result such as poor, average, or excellent.

**Assessment process:** the set of steps followed to calculate the quality score.

**Measurement process:** in our context, it is the same as an assessment process.

**Uncertainty:** refers to the level of confidence about the relationship of a value in a range. In our work, it is the confidence of the quality score boundaries whether the score can be precisely poor or it could be somewhere in between poor and very poor for example.

**Domain:** refers to the area of interest such as saying a domain specific ontology where the domain is the evolvability quality.

**Ontology:** a formal (machine-readable) representation of shared knowledge.

**Knowledge:** in our context, it is meaningful information or facts about the software to be assessed such as contributing community, historical releases, activities on code, and bugs/issues.

**Artifact:** represents the repository where the software knowledge is stored such as versioning system, issue tracker, mailing lists. Artifacts differ in their knowledge abstraction levels and representations.

**OWL:** Web Ontology Language, a family of knowledge representation languages used to model ontologies.

**OWL-DL:** Web Ontology Language designed enriched with reasoning expressiveness derived from the Description Logic field of study.

**OWA:** Open World Assumption, the assumption associated with the use of ontologies and it means that the truth-value of the knowledge is not derived from its explicit representation i.e., if it does not exist that does not mean it's false. The opposite is the Closed World Assumption.

**URI:** Uniform Resource Identifier, a string of characters used to identify a resource on the Internet.

**W3C:** the World Wide Web Consortium, the standards body for web technologies.

**RDF:** Resource Description Framework, a conceptual description of information that is implemented in web resources.

**RDFS:** Resource Description Framework Schema, a set of classes with certain properties for RDF.

**SPARQL:** an RDF query language for triple storage.

**Triple:** An expression in a form of subject-predicate-object used to format knowledge in RDF data model. For example, to represent the statement "the chair is red", the subject is *the chair*, the predicate is *has color* and the subject is *red*. Throughout the thesis, triples are represented in the following format: *<subject><predicate><object>*.

**N-triple:** is a plain text, line-based data representation format used for RDF data storage and transmission. In our research, we used an n-triple file format to store and transmit the triples that represent the software knowledge.

**XML:** Extensible Markup Language, a set of rules for encoding documents in machine-readable form.

**T-Box:** Terminology Box, a term used to describe a statement to establish a common, shared vocabulary for concepts and their relationships in ontologies. For example, X is an instance of Y.

**A-Box:** Assertion Box, it is associated with a T-Box and represents the individual or instance of a terminological statement represented in the T-Box. For example, Circle is an instance of Shape.

**Concept:** is a vocabulary or term defined as part of the T-Box to uniquely identify all its instances/individuals, for example, Quality Factor, Attribute, Measure, and Score.

**Individual:** is an instance, a real life example of a concept defined as part of the A-Box. In our context, for example, Evolvability is an individual of the concept Quality Factor.

**FLOSS:** stands for free, libre open source software. It refers to software systems that openly shared with online community to use, copy and change.

**Software Ecosystem:** refers to the collection of software artifacts that developed to constitute a software system.

**Semantic Web:** an initiative led by W3C (World Wide Web Consortium) that aims to convert the unstructured information available online into a semi-formal, and structured format.

**VERON<sup>1</sup>:** a VERsion control artifact ONtology: it defines concepts such as commit and commits date/time.

**ISSUEON<sup>2</sup>:** an ISSUE tracker ONtology: it defines concepts such as issue/bug, issue history, comments, and description.

**METON<sup>3</sup>:** a METadata ONtology: it defined shared concepts among various software artifacts such as contributors and files.

**QUAMON:** a QUAlity Metamodel ONtology: it defines the common concepts and relationships used by a quality model.

**TOOLON:** a TOOLs ONtology: it defines measurements related concepts used in assessing various evolvability qualities that are extracted from external code analysis tools.

---

<sup>1</sup> <http://aseg.cs.concordia.ca/ontologies/2010/11/veron.rdf>

<sup>2</sup> <https://sites.google.com/site/asegsecold/ontology/Issueon.owl>

<sup>3</sup> <http://aseg.cs.concordia.ca/ontologies/2010/11/meton.rdf>

# CHAPTER 1: INTRODUCTION

*A sign that the software engineering profession has matured will be that we lose our preoccupation with the first release and focus on the long-term health of our products. – David Parnas (1994)*

## 1.1 MOTIVATION

---

Software evolvability is a non-functional software quality that refers to the ability of software to adapt to continuously changing requirements over time. Requirements for changes in software come in different forms, such as newly added features, bug fixes, or technology migration. The challenge of achieving evolvability quality while facing changing requirements and environments remains a paramount issue for software systems. One of the challenges is the dispersed nature of software knowledge resources over different representation formats and abstraction levels. Another challenge is the lack of awareness of the software contributing community about the initial design decisions that makes the software changes, while maintaining its quality, a complex and time consuming process.

Despite the importance of non-functional requirements (NFRs), such as software evolvability, and the fact that software maintenance contributes up to 90% of the total software life cycle cost [1, 2], most existing processes still lack adequate support for NFRs in software maintenance.

The *Seattle Times* [3] estimated that the payroll cost alone to develop Microsoft's Vista operating system was \$10 billion while the evolution cost starting in the late 1980s was roughly \$30 billion.

It is estimated that 120 billion lines of source code were being maintained in 1990 [4]. In 2000, this number was about 250 billion and increasing [5]. Companies add an average of 10% more lines of source code each year only in enhancements [2], resulting in a doubling of the code size of the system being maintained every 7 years [2]. It has been estimated that in the US alone, more than \$70 billion is dedicated to maintenance annually [1]. For example, existing data associated with correcting the Y2K bug showed that the US government alone spent about \$8.38 billion during a 5-year period

to perform Y2K bug corrections, and, at a company-level, Nokia Inc. spent about \$90 million on preventative Y2K bug corrections. Despite the fact that both researchers and practitioners have gained a better understanding of the challenges and problems associated with software maintenance, the cost associated with maintaining and managing the evolution of software systems has increased from 67% of the total cost in 1979 [6] to more than 90% in 2000 [7].

In [8], source code artifact measures were applied to predict maintenance efforts depending on the evolvability of a software system; the study showed that in less evolvable systems, the time put towards requirement changes and fixing errors was between 25% and 36% greater. Table 1 summarizes the cost of maintenance as a percentage of the total software cost, which has clearly been increasing over time.

**TABLE 1 OVERVIEW OF SOFTWARE MAINTENANCE COST TO TOTAL LIFECYCLE COST [6]**

<b>Year</b>	<b>Cost of software maintenance</b>	<b>Definition</b>
2000	>90%	Software cost devoted to system maintenance & evolution / total software costs
1993	75%	Software maintenance / information system budget (Fortune 1000 companies)
1988	60-70%	Software maintenance / total management information systems operating budgets
1984	65-75%	Effort spent on software maintenance / total available software engineering effort
1981	>50%	Staff time spent on maintenance / total time (in 487 organizations)
1979	67%	Maintenance costs / total software costs

The term evolvability was widely popularized by Lehman [9, 10] as part of his eight laws explaining what constitutes software evolvability. Initially formulated in the mid-seventies [11] as three laws, Lehman's laws then evolved over the years [9-11] to the eight laws outlined below, which are defined in terms of E-type systems. An E-Type system or program refers to a feedback system that is embedded in real-world contexts where both the program and its environment evolve over time [9-11], we will be referring to some of the laws throughout the thesis.

- **Law I: Continuing Change** — An E-type program must be continually adapted or it becomes progressively less satisfactory.

- **Law II: Increasing Complexity** — As a program is evolved, its complexity increases unless work is done to maintain or reduce it.
- **Law III: Self-Regulation** — The program evolution process is self-regulating when there is a close to normal distribution of measures of product and process attributes.
- **Law IV: Conservation of Organizational Stability** (invariant work rate) — The average effective global activity rate on an evolving system is invariant over a product's lifetime.
- **Law V: Conservation of Familiarity** — During the active life of an evolving program, the content of successive releases is statistically invariant.
- **Law VI: Continuing Growth** — The functional content of a program must continually increase over its lifetime to maintain user satisfaction.
- **Law VII: Declining Quality** — E-type programs will be perceived as of declining quality unless they are rigorously maintained and adapted to a changing operational environment.
- **Law VIII: Feedback System** — E-type programming processes constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.

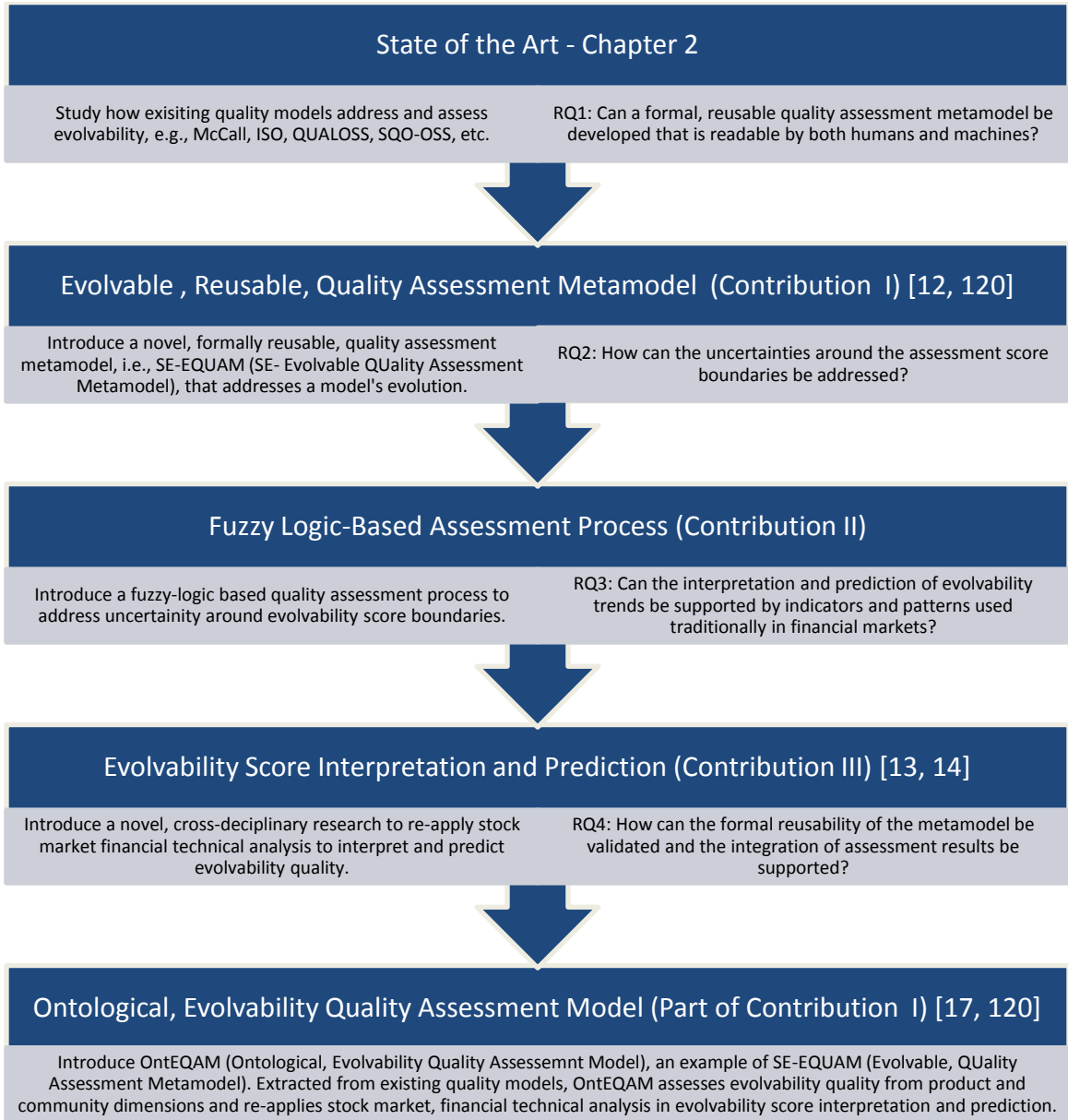
Software evolvability is used in this research as a non-functional quality attribute that reflects software adaptability to continuous change and its ease of reuse.

## 1.2 RESEARCH APPROACH

---

In this section, we describe our research approach (Figure 1). The approach is arranged in five steps, each of which define the topic covered and the research question raised (which leads to the next step).





**FIGURE 1 RESEACH APPROACH**

The sequence of this thesis is guided by the four research questions described in Figure 1. These research questions are focused on software evolvability quality. The different perspectives of software evolvability that we looked at include what constitutes evolvability quality as defined by existing literature. This perspective covers existing quality assessment models and examines how each of them is addressed.

- RQ1: Can a formal, reusable quality assessment metamodel be developed that is readable by both humans and machines?
  - A thorough examination of the existing quality assessment models is addressed as part of this thesis. Most of the models share a common metamodel representation, but none address the possibility of the formal reusability of the model, i.e., by providing a machine-readable metamodel that could be further extended and reused by other models. The lack of formal representation of the commonalities between the different models introduces challenges, such as compliance to a common structure, terminology, and manual reinvention of the existing work. To address our question, we introduce a novel, evolvable, quality assessment metamodel that is developed using ontological representation to provide a machine-readable format. Our metamodel could be reused and extended by other quality assessment models regardless of the quality or set of qualities the model is addressing. Our ontological metamodel demonstrates the integration of semantic reasoning technology, OWL DL, and SPARQL (Contribution I) [12].
- RQ2: How to address the uncertainties around the assessment score boundaries?
  - Existing quality assessment processes use a set of quality measures (e.g., lines of code and number of open issues) as input for their assessment to determine an assessment score (e.g., excellent or poor quality). Almost all existing approaches (except those using the fuzzy logic process) consider crisp boundaries (e.g., a measure value of  $x\%$  or less is considered to be very poor, whereas as a value of  $x\% + 0.1$  is ranked as only poor). However, using crisp assessment boundaries in the software domain is challenging because one has to deal during the assessment proves often with missing or incomplete knowledge that is being assessed. To address this challenge, we apply the fuzzy logic assessment approach, whereby the assessment results (e.g., quality scores and snapshots) are re-populated into the ontological representation of the assessment metamodel for further interpretation (Contribution II).

- RQ3: Can the interpretation and prediction of evolvability trends be supported by indicators and patterns used traditionally in financial markets?
  - Most of the existing quality assessment models provide an assessment score for users without any further interpretation. Our goal is to monitor the evolvability score patterns over time and then interpret these patterns as a way to predict future scores. This part of our work introduces a novel interdisciplinary research that re-applies the stock market analysis patterns used to interpret and predict stock price evolution on the software evolution (Contribution III)[13, 14].
- RQ4: How can the formal reusability of the metamodel be validated and the integration of assessment results be supported?
  - As a validation of the reusability of our semantic, ontological metamodel introduced to address RQ1, we present a quality model that addresses software evolvability quality as an instance of the quality metamodel. This quality model is extracted from existing quality assessment models (e.g., ISO/IEC 9126 [15] and QUALOSS [16]). The evolvability quality model formally reuses the metamodel ontological representation and semantics. The evolvability quality is assessed using the fuzzy logic process (RQ2), and the assessment results are integrated with a knowledge base of a unified ontological representation of the quality assessment input and output.

### 1.3 ISSUES NOT ADDRESSED IN THIS THESIS

---

The following issues are not addressed as part of this thesis:

- The focus of this thesis is software evolvability as a non-functional (NF) quality. The coverage of any NF software qualities, such as security, portability, performance, etc., is beyond the scope of this thesis. Other generic purpose quality models, which will be described as part of the background chapter, address most of non-evolvability specific qualities.

- The work performed by this thesis is presented in the form of case studies and experimental examples developed using our own assessment framework. The framework does not provide graphical user interface as a visualization of the assessment results, but the results are available in technical formats, such as fuzzy control language files, ontological n-triple files, and command line.
- This work does not introduce new quality assessment measures. We reuse measures and benchmarks from existing quality assessment models.

## 1.4 THESIS OUTLINE

Software quality assessment consists of the three main components: a metamodel, model, and a process (Figure 2). The metamodel is an abstraction, template or a frame that consist of a collection of concepts and relationships between them. A model is a concrete instance of the metamodel that addresses a specific quality composition. Finally, the assessment process is used by the model, which takes a set of quality assessment artifacts as an input and quality score as an output. The derived quality scores can then be further analyzed to detect historical patterns and predict future scores.

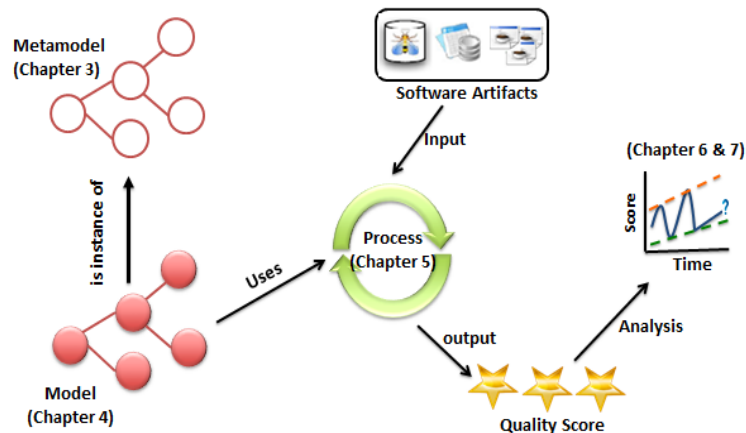


FIGURE 2 QUALITY ASSESSMENT COMPONENTS

The rest of this thesis is divided into eight chapters as mapped in Figure 2:

- Chapter 2: In this chapter, we will thoroughly discuss the terminology and definitions of the main concepts used in this research, such as evolvability and quality. This chapter describes

the existing quality models, coverage for software evolvability quality, and conformance to our syntactic metamodel. Quality assessment processes are described in a section within which we address the steps to provide quality scores and the different scoring systems proposed in the literature. This chapter ends with a summary of the challenges we faced in this research and a comparative study of the major properties of the existing quality models.

- Chapter 3: In this chapter, we introduce the details of our Contribution I (i.e., SE-EQUAM, the Evolvable, QQuality Assessment Metamodel). We describe the core requirements needed for a quality model to be considered evolvable and then outline how each requirement is addressed in our solution.
- Chapter 4: This chapter summarizes the fuzzified quality assessment process approach (Contribution II). It details the steps to benchmark and calculates the quality score on different levels of the quality model.
- Chapter 5: This chapter provides a technical case study to validate our Contribution I, which is presented in Chapter 3. The validation is performed by instantiating the evolvable quality metamodel (Chapter 3) and reusing the fuzzified assessment process (Chapter 4).
- Chapter 6: In this chapter, we first discuss the existing literature of the quality prediction models used in software industry and then we provide a detailed background study on the financial technical analysis for price interpretation and prediction provided by the stock market industry, which we re-apply in our research as part of the next chapter (Chapter 7).
- Chapter 7: In this chapter, we explain our Contribution III, a novel, interdisciplinary approach that re-applies stock market technical analysis patterns, which are used to interpret and predict price evolution on software quality score evolution. This study is performed on the results obtained from reusing our metamodel (Chapter 3), assessment process (Chapter 4), and evolvability model (Chapter 5).
- Chapter 8: This chapter reviews our research questions and explains how they were addressed. It then describes the directions for future work.

## CHAPTER 2: BACKGROUND

### 2.1 EVOLVABILITY DEFINED

---

Software systems require constant modifications in order to meet new and ever-changing requirements [8], increasing software complexities, and maintenance cost, unless change accommodations are rigorously taking into account as part of the software development process [18]. All large and successful software products require continuous evolution and as observed by Brooks in [19]: “As soon as one freezes a design, it becomes obsolete in terms of its concepts.” The term **evolvability** was widely popularized by Lehman [9, 20] as part of his eight laws explained what constitutes software evolvability as continuing change, increasing complexity, and declining quality. Analyzing the maintenance cost of 20 source code releases of the OS/360 empirically proves these laws. In [21] Lehman defined evolvability both as a verb and a noun, but then mainly focused on its use as a verb, which he described as: “how to evolve a software system.”

In [22] Kemerer and Slaughter suggest the following definition of software evolution: “Software evolution refers to the dynamic behavior of software systems as they are maintained and enhanced over their lifetime”. In [23] evolvability was defined as “the capacity of software systems to support adaptation to new requirements and use contexts over time”. Common to these definitions is their description of evolvability as the ability of a software product to react to ongoing changes and the implementation of these changes in order to extend the lifespan of a software product.

More recently, Lehman’s definition of evolvability [21] was further refined by the ISO/IEC and IEEE standards 14764-2006 [24], which used it to define the software maintenance area as: “the totality of activities required to provide cost-effective support to a software system. Activities are performed during the pre-delivery stage as well as the post-delivery stage”.

Given this definition, the terms software evolution and software maintenance are virtually synonymous [25, 26] and will be used interchangeably as a non-functional software quality throughout this study.

## 2.2 QUALITY ASSESSMENT MODELS

---

Quality is a widely used term to evaluate the maturity of development processes in general within an organization from a business point of view. Defining quality allows organizations to specify and determine if a product has met certain non-functional and functional requirements. As Kitchenham [27] states: "quality is hard to define, impossible to measure, easy to recognize"; Gillies et al. [28] derives a similar observation by stating that quality is "transparent when presented, but easily recognized in its absence". These general assumptions resulted in a large body of inconsistencies and often even controversial definitions of what constitutes quality for software products. Unlike functional requirements, where a single analysis technique (e.g., use case modeling) is sufficient to identify essentially all requirements, the same analysis is not appropriate for all quality requirements.

The following section presents some commonly used definitions of what constitutes quality in software. Software quality is often used to specify the degree to which a software product meets specific functional and non-functional quality attributes.

Quality, as defined by the ISO 9000:2000 standard [29], is the "degree to which a set of inherent characteristics fulfills requirements," where a requirement is a "need or expectation that is stated, generally implied or obligatory." There has also been a significant body of work on classifying requirements in order to establish links between software quality and requirements.

Another definition of what constitutes software quality is presented by the ISO standardization committee in ISO 8402:1994 [30], which defines software quality as: "the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs".

Kitchenham [27] also specifies quality as the “fitness for needs,” which covers conformance to the software’s specification (needs) and its ability to address the right problem (fitness). The IEEE 1061 standard [31] interprets software quality as the degree to which a software system fulfills a selected group of attribute requirements, where quality attributes represent features or characteristics that affect an item’s quality. In [32] a quality attribute has been defined as a non-functional characteristic of a component or a system. In the same article, evolvability was described as a non-functional requirement of system and therefore was treated as a characteristic and quality attribute [32].

Given the constantly and rapidly changing technologies (programming paradigms), processes (agile and open source development), and new application domains (distributed and service oriented), as well as social and economic changes (collaborative and global software development), software quality is gaining importance; yet, it must also adapt and evolve to capture these new contexts.

Assessing quality to assure easier handling of evolution problems and further changes to the software in the future has been addressed in existing research through the introduction of various software quality models. These models were introduced to define different software quality dimensions and classified quality areas that affect the development and maintenance of software products.

The ISO/IEC 14598 [33] standard defines quality models as: “The set of characteristics and relationships between them, which provides the basis for specifying quality requirements and evaluating quality.”

Software quality models are artifacts for describing the quality factors of a single software product of different types and domains. In order to allow for a comparison between the existing quality models, a generic metamodel is introduced to establish a common terminology and structure.

### 2.2.1. GENERIC QUALITY ASSESSMENT METAMODEL

Most quality assessment models share a generic structure, template, or frame for assessing software qualities that corresponds to a hierarchy or tree structure with multiple levels and a set of constraints that define the relationship between one level and the next one; in this thesis, this will be



called the quality assessment *metamodel*. Figure 3 represents the common syntactic metamodel that will be used to standardize the terminology and analyze and evaluate the surveyed quality assessments models.

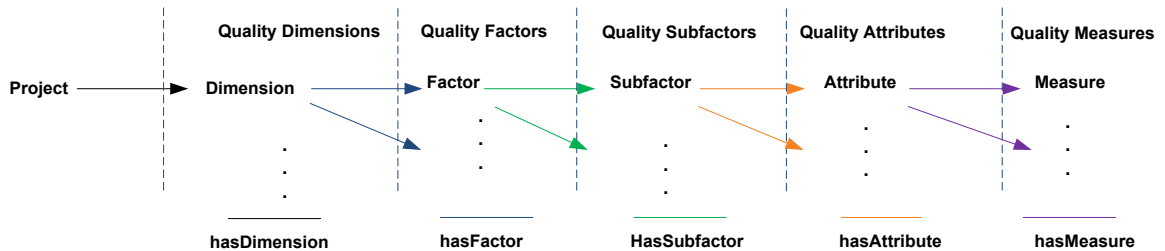


FIGURE 3 EXTRACTED (SYNTACTIC) QUALITY ASSESSMENT METAMODEL

With the exception of a few models that merge some levels, the metamodel is based on a core of five abstraction levels; other models use the optional dimensional level 0.

**Level 0 (Quality Dimension):** The quality dimension is an optional level used by some models to group quality factors used in many quality models. For example, in [34] the software quality of the FLOSS project is viewed from the product, community, process, and tools dimensions. In [35] quality is studied from different FLOSS roles perspectives, such as maintainer, contributor, adopter, and integrator. In [36] quality is either defined by the artifact during specific phases of the software development lifecycle or by one of three domains: application, problem, or solution. The ISO/IEC 9126 [15] looked at quality as external or internal; while in [37] quality dimensions are domain, experience, and process.

**Level 1 (Quality Factor):** In the literature, different terminology exists to describe the term quality factors. The ISO 9000:2005 standard [38] refers to them as *quality characteristics*, being any feature or characteristic of a product/service that is required to satisfy customer needs or achieve fitness for use [29], while *characteristics* themselves are defined as non-measurable quality factors that are used to classify the upper level of sub-characteristics of the ISO model. The IEEE standard [39] introduces *quality factors* as management-oriented attributes of software that contribute to its

overall quality. In the context of this research, we use the IEEE term *quality factors* to represent the upper level quality factors of the surveyed quality models.

**Level 2 (Quality Sub-factor):** As outlined above, various definitions for quality factors exist, including the ISO/IEC standard [38], which refers to them as *quality sub-characteristics* that may be subjectively measured when required and may be decomposed into other sub-characteristics or alternatively into attributes that help in their measurement. The IEEE standard [39] refers to *quality sub-factors* as the decomposition of a quality factor to its technical components.

**Level 3 (Quality Attributes):** In the literature, both the ISO standard [38] and the IEEE standard [39] refer to this level as the *quality attribute*. According to [39], a *quality attribute* is defined as a characteristic of software that is a measurable physical or abstract property of an entity [39].

**Level 4 (Quality Measures):** Called *metrics* by some models, such as QUALOSS [40], SQUALE [41], and SQO-OSS [42], *quality measures* are defined as directly measurable attributes of software and are used to express certain aspects of the product that affect quality [43]. Another definition of *quality measures* is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality [39]. The ISO 9000:2000<sup>4</sup> process standard defines a metric as a numerical measurement of the effectiveness of tasks and activities. In this thesis we use the terms metric and measure interchangeably.

### 2.2.2. QUALITY ASSESSMENT IN TRADITIONAL VERSUS OPEN SOURCE SOFTWARE

For the survey of the quality assessment models, we will differentiate between models created for open source and traditional (non-open) source software. FLOSS, unlike closed-source software, is freely available software whereby users have access to the source code. FLOSS projects typically share a collaborative development approach, with multiple geographically distributed developers (aka contributors) addressing different needs and hardware platforms. The coordination among the

---

<sup>4</sup> [http://en.wikipedia.org/wiki/ISO\\_9000](http://en.wikipedia.org/wiki/ISO_9000)

contributors is effectively based on shared communication protocols and there is no enforced leadership that sets the project guidelines or drives the main decisions regarding the development process. Roles change as FLOSS development grows and leadership arises in a bottom-up investiture, which leads to a greater concentration on development itself. All contributors work on the same source code and have equal opportunity to propose solutions to the same problem, which improves the quality not only by solving errors but also by creating more adaptable software.<sup>5</sup> A major drawback with the FLOSS assessment approach is for projects with often less structured and controllable development processes in place (e.g., open source projects), when a task assignment is becoming a more ad-hoc decision (e.g., for open source projects, it depends completely on the availability of volunteers). For large and successful FLOSS projects this problem does not exist because the community is large and a wider selection is available. In contrast commercial (proprietary or closed source) software is typically recognized by a strictly pre-defined organizational structure with role assignments and reporting hierarchies enforced from the beginning of the development process.

## 2.3 TRADITIONAL ASSESSMENT MODELS

---

The origin of most traditional (non-open source) assessment models dates back to the mid-1970s, prior to widespread use of open source projects. These models originated from the need to assess the quality of software products developed either internally within organizational boundaries or externally as part of outsourcing of the software development to third party developers. Common to these assessment models both their focus on the (software) **product** and the well-defined software development **process** supporting these products. In what follows, we survey the major traditional assessment models, by mapping them to the structure of our generic metamodel in order to illustrate their coverage and allow for a discussion of their benefits and limitations.

For brevity, we represent in this chapter the quality measures level of the models as a black-box. For a detailed coverage of the individual measures used by the model, we refer the reader to the refer-

---

<sup>5</sup> <http://www.opensource.org>

ences associated with each model. Among the more typical measures found in most models are cyclomatic complexity and code size in terms of lines of code.

### 2.3.1. MCCALL QUALITY MODEL

The 1977 McCall quality model [44] was introduced to primarily support the assessment of software developers and development process quality. It attempts to address both users' and developers' perspectives by introducing three main dimensions used to categorize quality attributes [45]. Figure 4 shows the three dimensions: **product revision** (ability to change), **product transition** (adaptability to new environments), and **product operations** (basic operational characteristics). Mapping the McCall model to our generic quality assessment metamodel demonstrates that the McCall model does not have a sub-factors level. The model has been criticized for using non-objective measures [46, 47] and because the *functionality* of a software product is not directly covered as a quality aspect [46]. Figure 4 show that the model has no specific focus on evolvability.

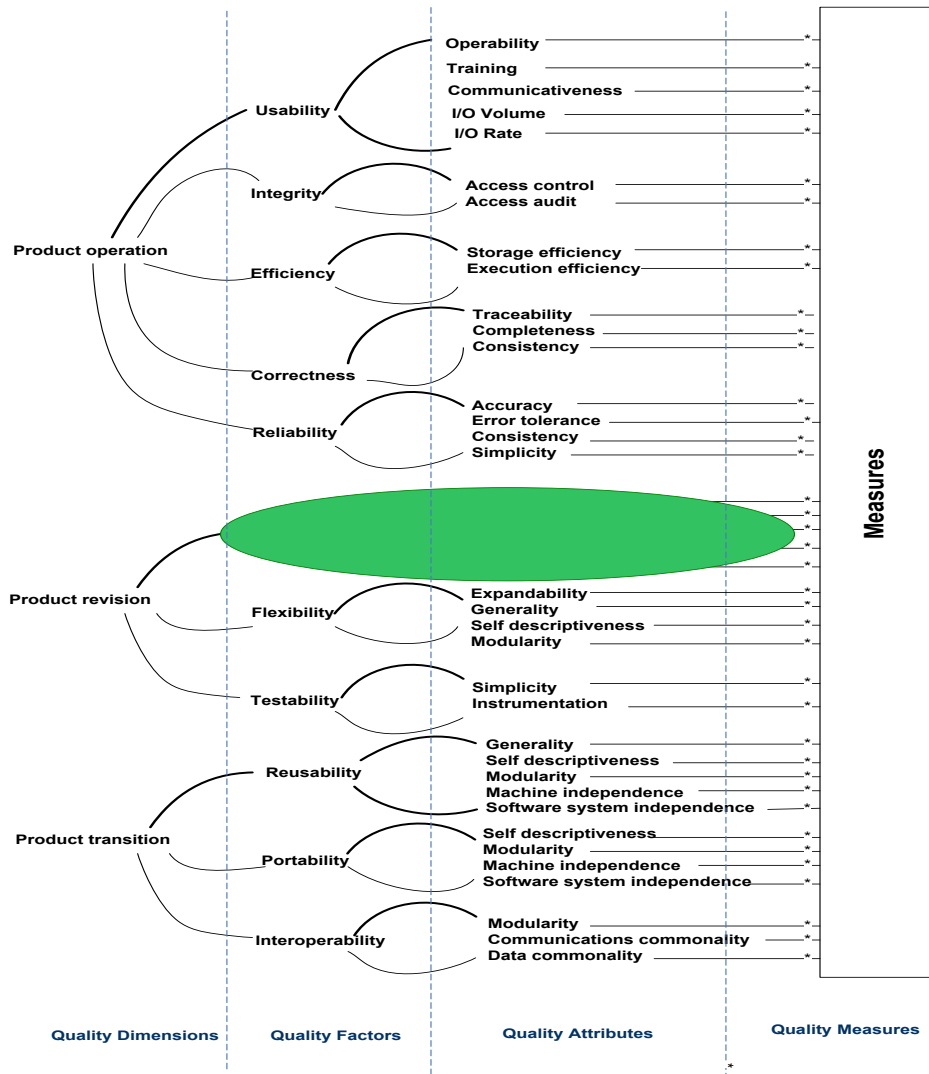


FIGURE 4 MCCALL QUALITY MODEL

### 2.3.2. BOEHM'S QUALITY MODEL

Boehm's 1976 model [48] is one of the early quality assessment models. Overall it is similar to McCall's model but with more emphasis on the *maintainability* aspects of quality [45]. It introduces maintainability sub-factors, such as traceability and comprehensibility, along with associated measures. Furthermore, Boehm's model covers hardware dependencies not addressed by McCall's model. Comparing Boehm's model with our generic metamodel shows that it covers all the quality assessment levels (Figure 5). One of the criticisms of Boehm's model is that the model definitions of

the quality factors are highly technical and difficult for non-technical stakeholders to grasp [47]. Another criticism is that the two quality dimensions are too generic and not well defined.

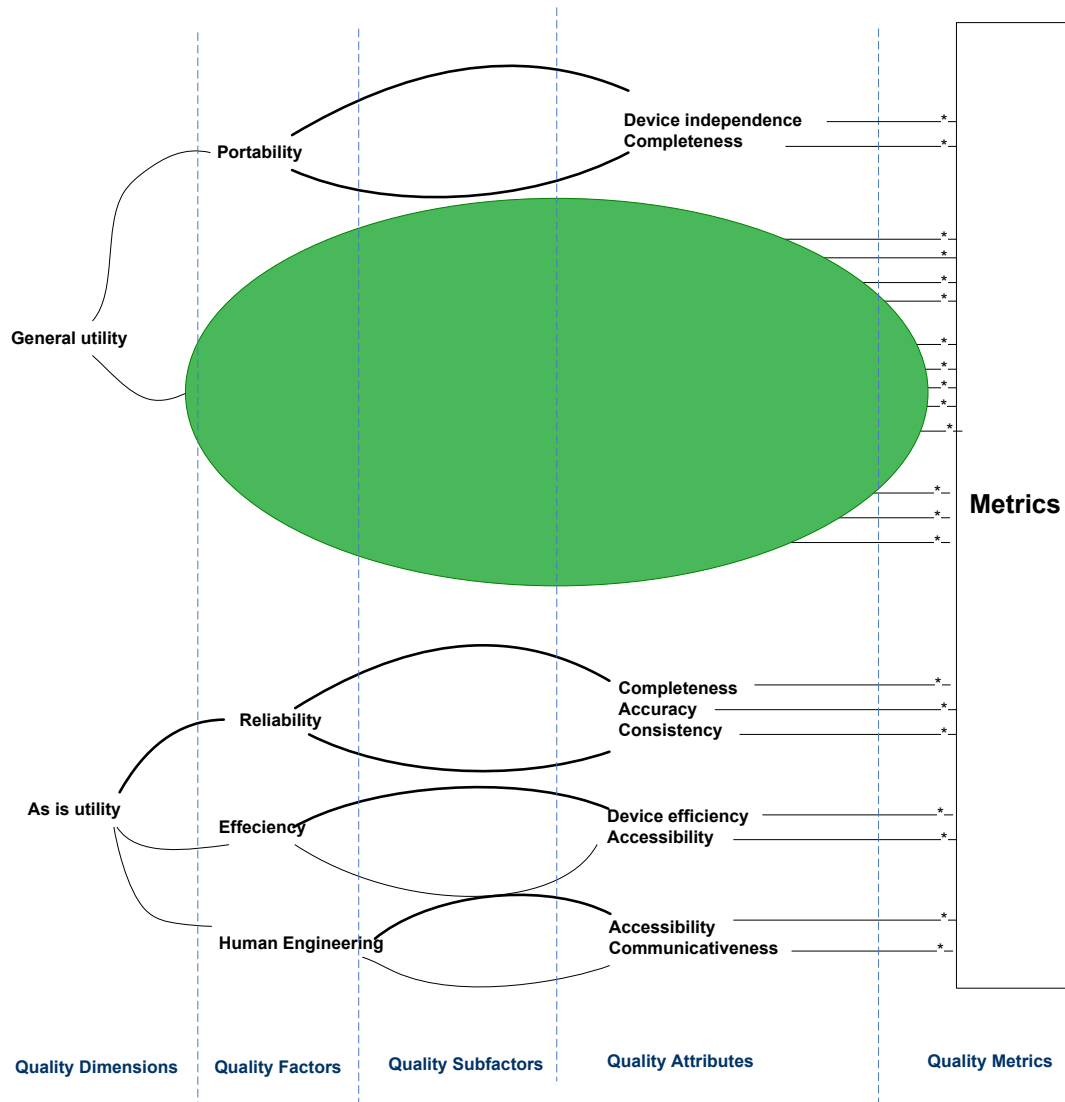


FIGURE 5 BOEHM QUALITY MODEL

### 2.3.3. ISO/IEC 9126 AND SQUARE

In 1991, the ISO committee, together with the International Electro-technical Commission (IEC), established a common ISO/IEC 9126 quality model [15]. There are hundreds of standards developed and maintained by software organizations but the most popular model is ISO/IEC 9126 [49].

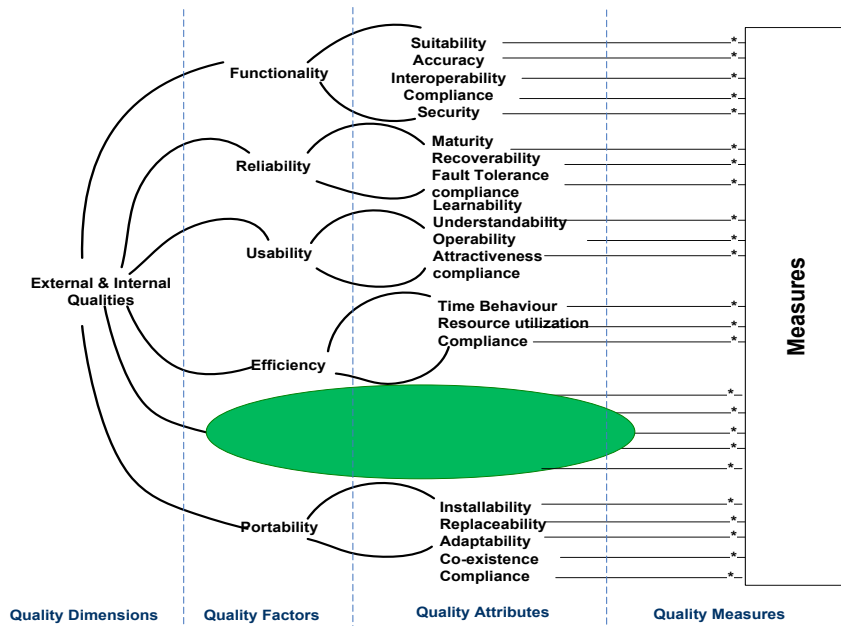


FIGURE 6 ISO/IEC 9126-1 QUALITY MODEL

The objective of this standard was to integrate the many existing perspectives on software quality and establish a worldwide-standardized model, which was further revised in 1998. The ISO/IEC 9126 [15] views quality from two dimensions: **internal** (how the product was developed) and **external** (how the product works in its environment). It also defines six characteristics (quality factors), which are further subdivided into sub-characteristics (quality sub-factors), as shown in the Figure 6. In this quality model, maintainability is treated as a separate quality factor with several quality attributes; however, it never clarifies how the identified factors can be measured [46, 50].

In 2011, ISO/IEC 25010 [51] was introduced to replace ISO/IEC 9126 [15]. ISO/IEC 25010 or SQuARE [51] stands for Systems and Software Quality Requirements and Evaluation. SQuARE revises and extends ISO/IEC 9126 [15], for example by adding security as a characteristic/factor rather than a sub-characteristic; using more accurate names such as modifiability instead of changeability; and adding new characteristics such as compatibility which includes interoperability and co-existence, functional completeness, capacity, and reusability. SQuARE defines two models: *product quality model*, which assesses internal and external software qualities (Figure 7). The second SQuARE defined

model is *Quality in use model*, which assesses the software product qualities that impact stakeholders (Figure 8).

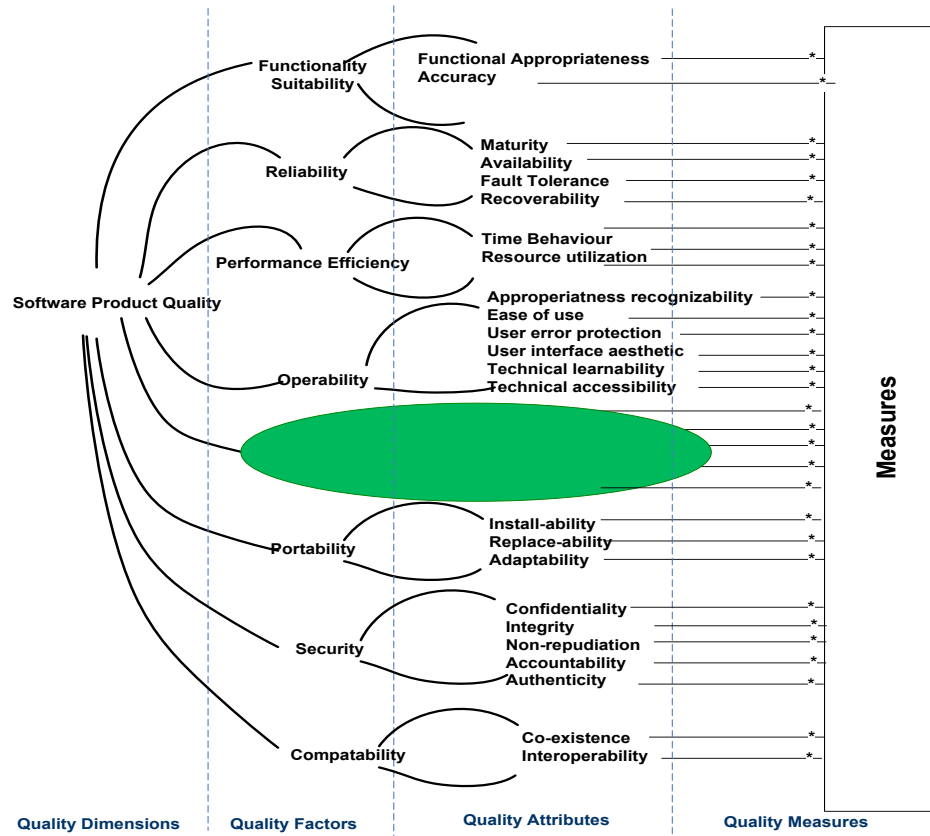


FIGURE 7 PRODUCT QUALITY MODEL AS DEFINED BY THE ISO/IEC 25010:2011

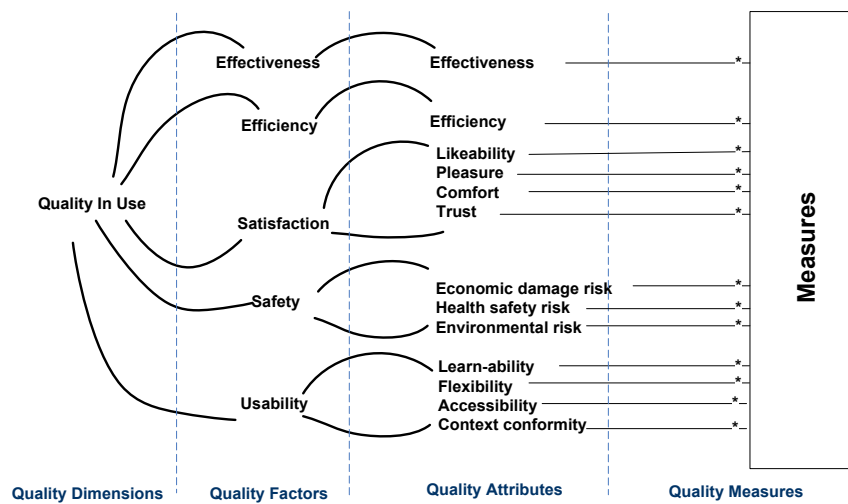
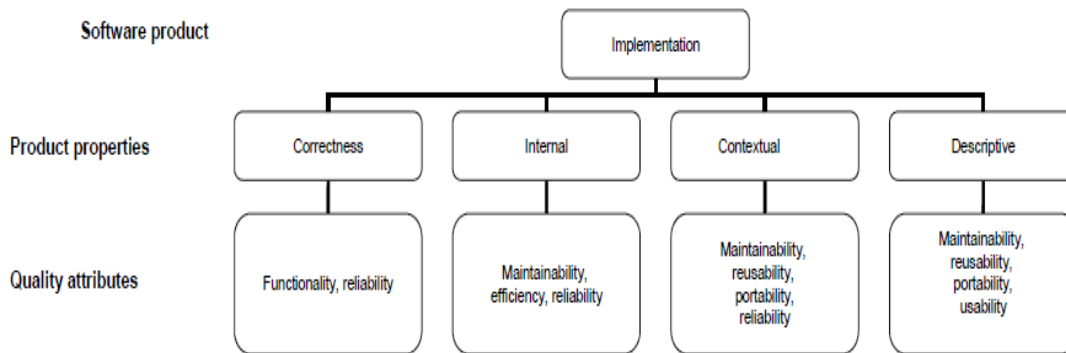


FIGURE 8 QUALITY IN USE MODEL AS DEFINED BY THE ISO/IEC 25010:2011



### 2.3.4. DROMEY'S QUALITY MODEL

In 1996, Dromey's model [52, 53], which is based on McCall and Boehm's models, was introduced. The model links product properties that influence quality with quality attributes [45]. If mapped to our generic metamodel, the software product implementation in the Dromey's model (Figure 9) corresponds to our quality dimension, the four product's properties map to our quality factors, and each factor/property has a set of quality attributes.



**FIGURE 9 DROMEY QUALITY MODEL [64]**

In the quality attributes level, the model identifies eight attributes, with six being the same as in the ISO 9126 model [15]; it additionally defines reusability and process maturity attributes. One of the main disadvantages of Dromey's model is related to its ability to assess reliability and maintainability, since both reliability and maintainability are assessed during the implementation phase, before the system is actually operational [46].

### 2.3.5. NASA SATC QUALITY MODEL

In 1996, the NASA Software Assurance Technology Center SATC developed a quality model in order to help its software managers to identify project risks and assess software quality [54]. The NASA SATC quality model is based on both McCall's [44] and Boehm's [48] earlier models, as well as the early ISO 9126 standard [15]. When mapped to our generic metamodel, the NASA SATC model (Figure 10) covers three levels: quality dimensions, quality attributes, and metrics associated with

attributes. It lacks coverage for the quality factors and sub-factors levels. Also the model is very general in respect to software quality and does not emphasize maintainability and comprehensibility.

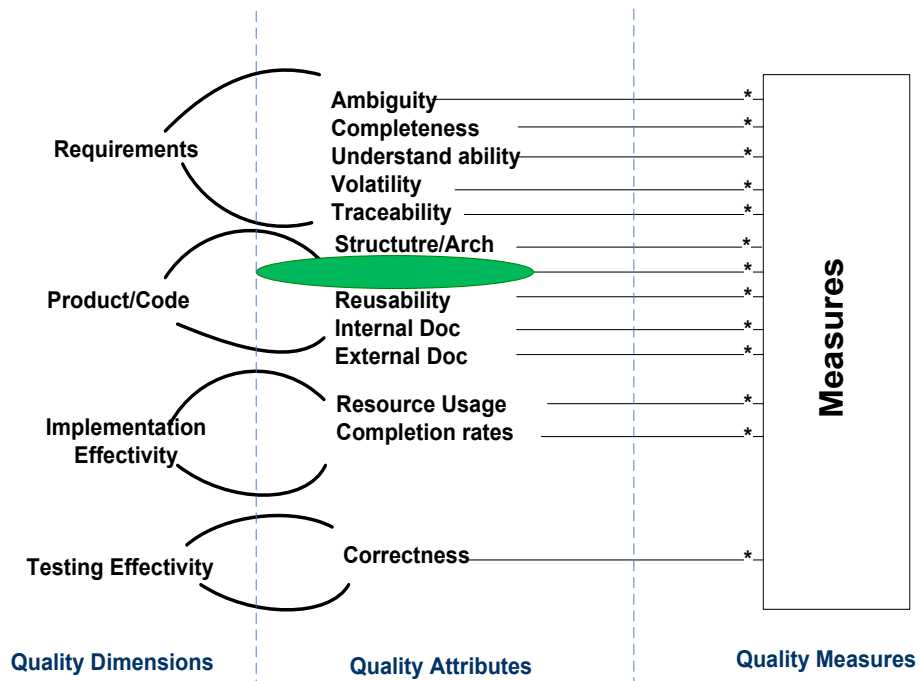


FIGURE 10 NASA SATC QUALITY MODEL

## 2.4 OPEN SOURCE ASSESSMENT MODELS

In what follows, some of the major FLOSS quality assessment models are surveyed by mapping them again to our generic metamodel and discussing the strengths and weaknesses of each model, in particular with respect to their ability to support the assessment of evolvability.

### 2.4.1. QSOS

In 2004, the method for Qualification and Selection of Open Source software (QSOS) [55] was introduced. The QSOS offers a free method for adapters to qualify and select FLOSS software based on the best match with respect to planned/actual and technical/functional requirements. The model uses adjustable weights for the assessment process. Data is provided through manual data import. The model provides a web-based interface to visualize the assessment results. In order to complete the

automatic calculation of the assessment, no incomplete data should exist. The QSOS (Figure 11) supports three levels of our generic metamodel, with the main category being the quality dimensions level, the QSOS criteria being our quality attributes, and the basic criterion corresponding to the measures. The QSOS provides a discrete scoring for the various quality criteria; however, the scoring itself is too restrictive by supporting only values from 0 to 2. It has also been observed that the QSOS missed some of the FLOSS specific quality areas, such as install-ability, security, reference deployment, standard compliance, supporting availability, stability, and performance [56]. One of the major shortcomings of the QSOS is its obsolete scoring, which prevents the model from being tailored to a specific context or user's need. The QSOS is a general quality assessment framework that is not evolvability specific and treats maintenance only as one of many quality factors.

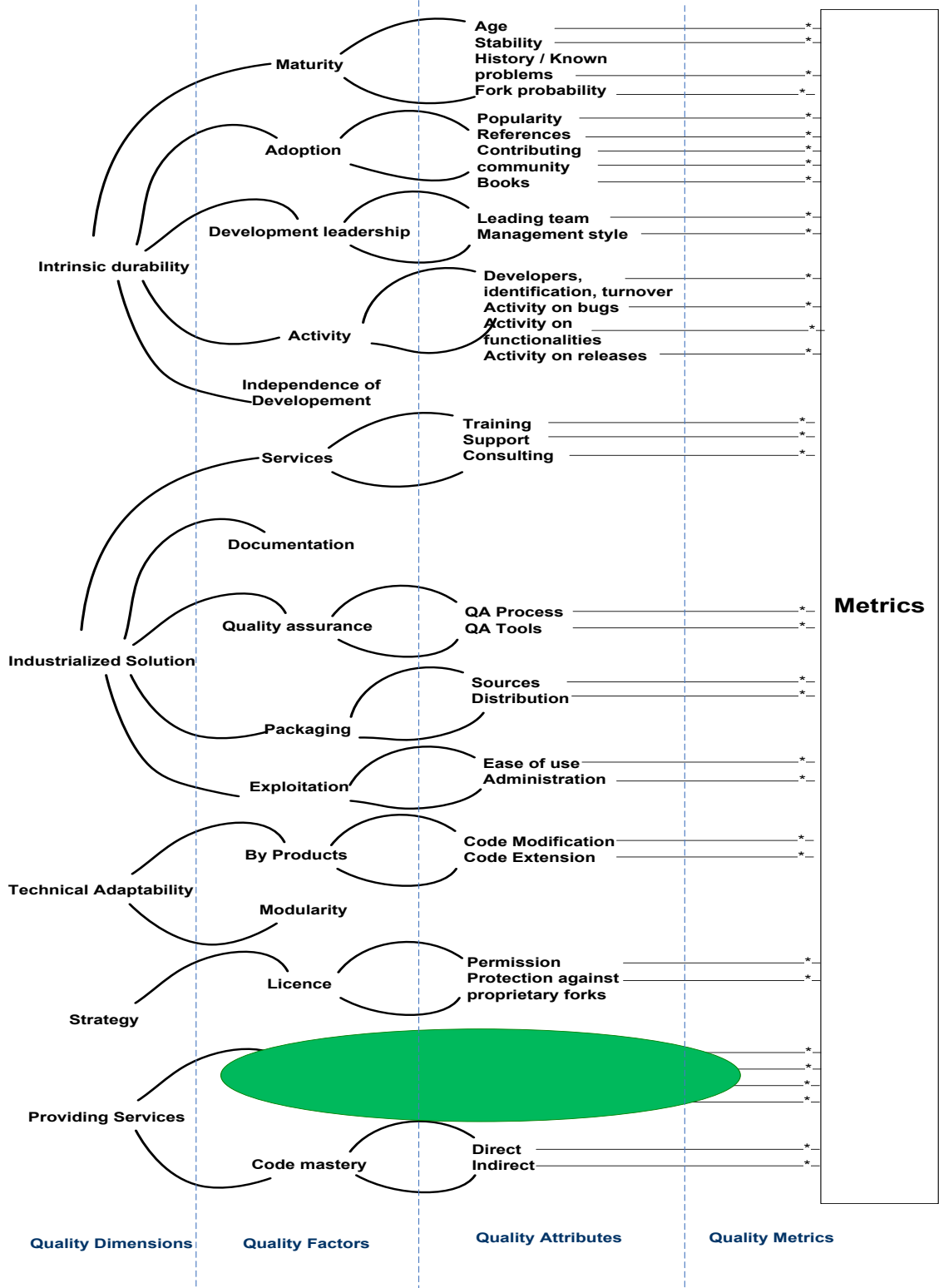


FIGURE 11 QSOS QUALITY MODEL

### 2.4.2. CAPGEMINI OSMM

The Open Source Maturity Model (OSMM) [57] proposed in 2003 by CapGemini was an early attempt to standardize the ad-hoc assessment approaches of FLOSS projects [58]. The OSMM is a commercial product that allows for the evaluation of 12 generic product related characteristics and 15 user related characteristics. Scores range from 1 to 5 and adjustable weights are applied to calculate the final score [57]. If a certain indicator is inapplicable then a score of 3 is assigned, which will not have any positive or negative effect of the final result; this is a unique characteristic of this model that distinguishes it from other models. When mapping the CapGemini OSMM (Figure 12) to our generic meta-model, one can observe that it lacks factors and sub-factors levels. However, while its focus on the assessment of the quality of the development processes, there is no explicit coverage of maintainability within the assessment model.

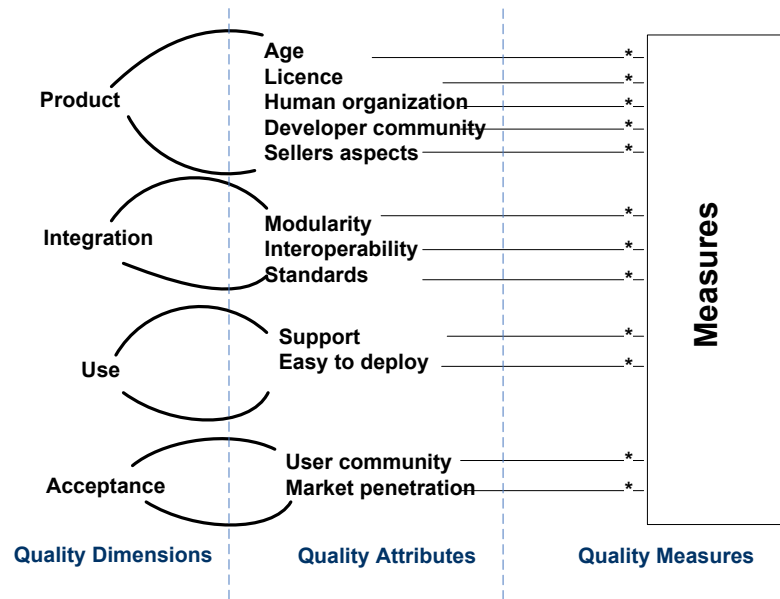


FIGURE 12 CAPGEMINI OSMM QUALITY MODEL

### 2.4.3. OPENBRR

In 2005, the Open Business Readiness Rating (OpenBRR) was introduced [59]. OpenBRR is built upon the existing OSMM models Cap Gemini's OSMM [57]. It defines an open standard assessment model [58] that enables the community to rate FLOSS in order to assess adopters in determining whether

certain software is mature enough to be adopted. The model uses a 1 to 5 score range but only three scores are defined for almost half the metrics. This model can be adapted to the current user needs. When mapped to our generic quality assessment metamodel, OpenBRR (Figure 13) covers only two quality abstraction levels, the quality attributes and metrics, and therefore provides only a limited grouping of the quality aspects.

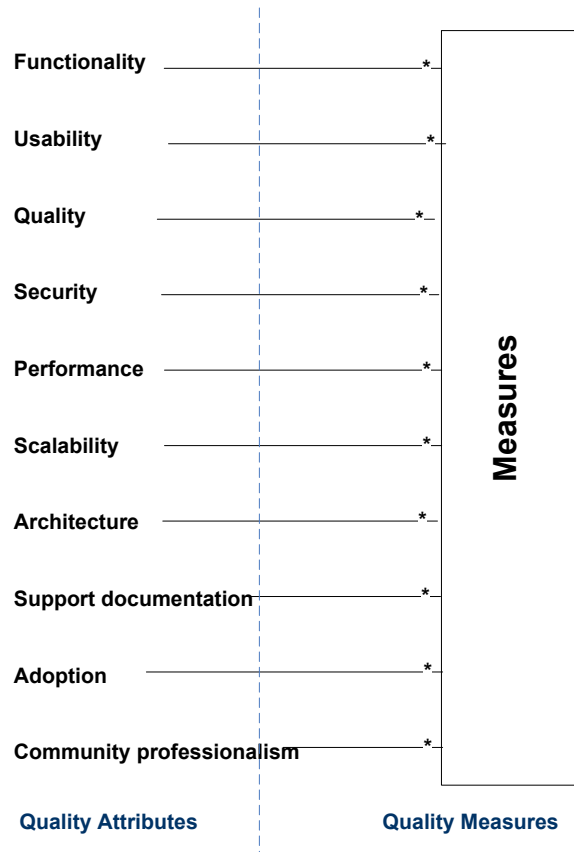


FIGURE 13 OPENBRR QUALITY MODEL

One of the challenges with the model itself is that some of the measures in the model, such as end-user UI experience, performance testing and benchmarks, performance tuning and configuration, and difficulty to enter the core development team, are not well defined. Nine criteria in the model asked for data that is typically unavailable or not easy to obtain, such as time to setup pre-requisites, number of security vulnerabilities in last 6 months, number of security vulnerabilities still open, reference deployment, and design for scalability. Furthermore, the model does not indicate if the evalua-

tion is for any historical release or for the latest version. This approach could either be more open for the user to decide what version to assess or vague and undefined. This model also misses some interesting FLOSS quality areas such as: process quality, strategy and road map, modularity,<sup>6</sup> and in particular maintainability as a quality factor. The OpenBRR [59] website states that the project does not own a thriving community yet and the authors are working on revising it.

#### 2.4.4. SQO-OSS

In 2006, Software Quality Observatory for Open Source Software (SQO-OSS) was introduced [60]. The project provides a framework to automatically evaluate FLOSS source code quality through a comprehensive suite of software quality assessment tools to allow for a more objective analysis and benchmarking of open source software [61]. The objective of the model is to improve code quality by introducing a large range of innovative measures through the use of data mining approach.

Mapping to the generic quality assessment metamodel, the SQO-OSS model covers the quality of FLOSS from two dimensions, product/code quality and community quality, and defines quality factors, attributes, and associated measures levels. From our generic metamodel perspective, it lacks the sub-factors level, as outlined in Figure 14.

Previous models developed for FLOSS evaluation require a substantial effort from the user regarding the rating of the software, while the model presented here asks for limited user interaction. Apart from the model itself, the evaluation process facilitates a profile based evaluation algorithm that is different from the traditional weighted aggregation used by most of the models. The evaluator, if deemed necessary, can alter the profiles used for evaluation. The notion of developer and user communities, communication, and collaboration, however, is not very explicit in the model and therefore does not explicitly support the analysis of relationships between developers and code parts, for example. Also the model only automates source code artifact and proposes an expected result of considering other artifacts [61]. Its coverage for evolvability is restricted to the same attributes offered by the ISO model. Community evolvability is not covered.

---

<sup>6</sup> <http://www.itpro.lk/node/2814>

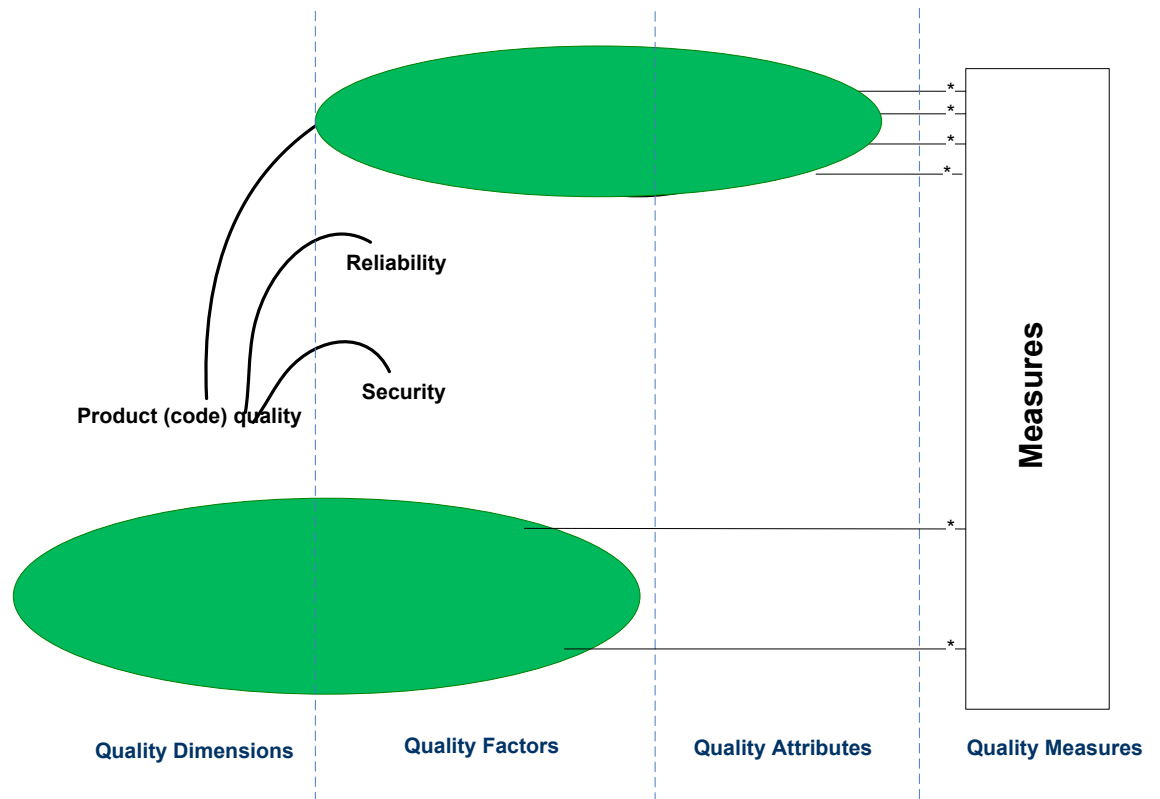


FIGURE 14 SQO-OSS QUALITY MODEL

#### 2.4.5. QUALOSS

In 2007, QUALity of Open Source Software (QUALOSS) [16] was introduced based on the OpenBRR [59] and the QSOS [55] models to implement a FLOSS quality assessment model to assess evolvability and robustness of FLOSS [56, 62, 63]. The objective of this model is to focus on the evolvability aspects of a system. It is built around a high-level assessment methodology that is used to benchmark the quality of FLOSS. The initial assessment is a sequence of manual process of interviewing, writing, and questionnaires that the user needs to do in order to initiate the assessment.

Data sources considered include FLOSS project releases, version control system, bug tracking systems, and mailing list archives. QUALOSS only published score results of a subset of its measures. These measures were extracted from version control system and issue tracker.



Mapped to the generic quality assessment metamodel, the QUALOSS model (Figure 15) misses the sub-factors level. The QUALOSS model covers evolvability and robustness qualities but only evolvability is considered in Figure 15.

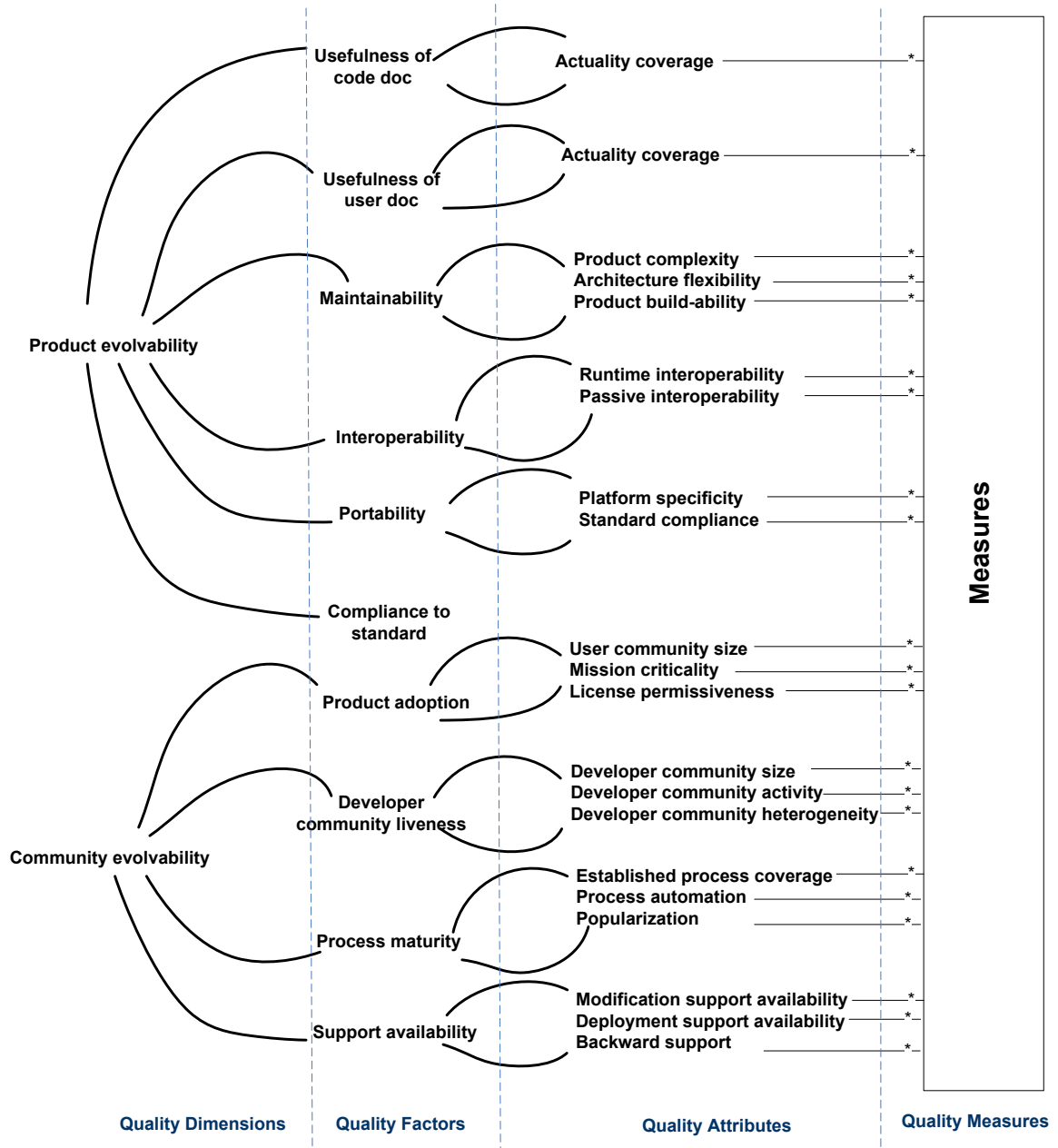


FIGURE 15 QUALOSS QUALITY MODEL

Many of the measures used by the QUALOSS model rely on simple size and completeness measurements and also include ambiguous references to user and community size without specifying any

acceptable benchmarks. Furthermore, many of the proposed measures in the model are not currently supported by automated tools and require manual data extraction/analysis. Some quality attributes do not have any defined measures, such as mission criticality.

#### 2.4.6. SIG MAINTAINABILITY MODEL

In 2007, Heitlager et al. introduced a practical model for rating maintainability quality called the SIG Maintainability Model (SMM)<sup>7</sup>. SIG stands for Software Improvement Group.<sup>8</sup> The objective of the SMM is assessing maintainability quality based on the ISO/IEC 9126 standard [64-66]. It maps four of the maintainability quality attributes to source code properties (Figure 16).

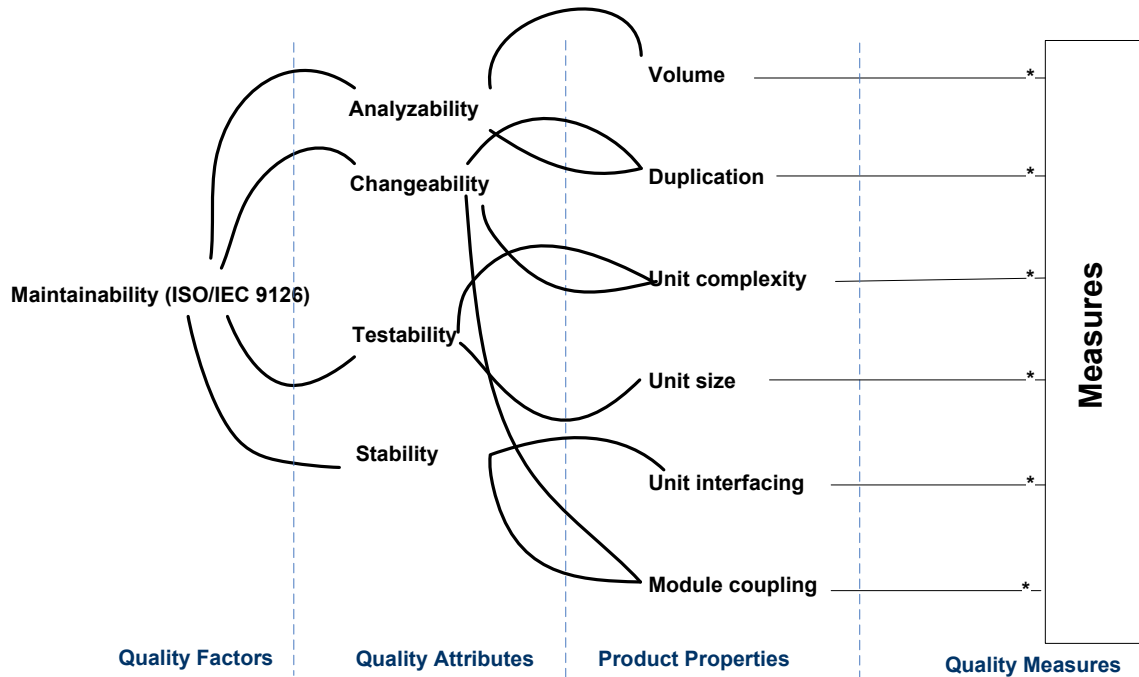


FIGURE 16 SIG MAINTAINABILITY MODEL

<sup>7</sup> [http://www.sig.eu/en/Research/690/\\_Maintainability\\_Model\\_.html](http://www.sig.eu/en/Research/690/_Maintainability_Model_.html)

<sup>8</sup> <http://www.sig.eu/en>

Quality attributes and assigned measures:

- Volume: The overall volume of the source code influences the analyzability of the system (the larger the system the harder it is to analyze). Examples of assigned measures include: LOC, function points, files count, and functional fields counts.
- Complexity per unit: The complexity of the source code units influences the system's changeability and its testability. Examples of assigned measures include: cyclomatic complexity, fan-in, fan-out, coupling, and stability measures.
- Duplication: The degree of source code duplication (also called code cloning) influences analyzability and changeability. An example of assigned measures includes: duplicated blocks over 6 lines.
- Unit size: The size of units influences their analyzability and testability and therefore that of the system as a whole. Examples of assigned measures include: unit LOC, and code size.
- Unit testing: The degree of unit testing influences the analyzability, stability, and testability of the system. Examples of assigned measures include: unit test coverage and number of assert statements.

When mapping SMM to our generic quality metamodel, SMM extracts a subset of ISO/IEC 9126 [15] that omit the dimension and sub-factors levels. SMM added a new layer that maps each quality attribute to one or more of what they call product properties (e.g., volume and duplication). On the other hand, each of these product properties has a 1-1 mapping with a quality measure.

SMM provides a clearly defined model structure along with simple set of quality measure that could be easily extracted and calculated to support full automation. This has been already implemented as a plugin as part of SonarSystem<sup>9</sup>. SMM only considers a single software artifact, i.e., source code. This eliminates any knowledge to be added to the assessment from other artifacts, such as the issue track-

---

<sup>9</sup> <http://www.sonarsource.com/>

er, code comments, or versioning system. It also ignores the software community as a perspective that affects the software maintainability quality.

### 2.4.7. SQUALE

Qualixo first introduced Software Quality Enhancement (SQUALE) in 2008 [41, 67]. It is based on McCall's model [44] and the ISO/IEC 9126 [15] standard. SQUALE's objective is to enhance both technical and economical software qualities by assessing the effort needed for software modifications and identifying healing actions (practices); refer to Figure 17. It provides an opens source tool to assess and improve software quality over time.

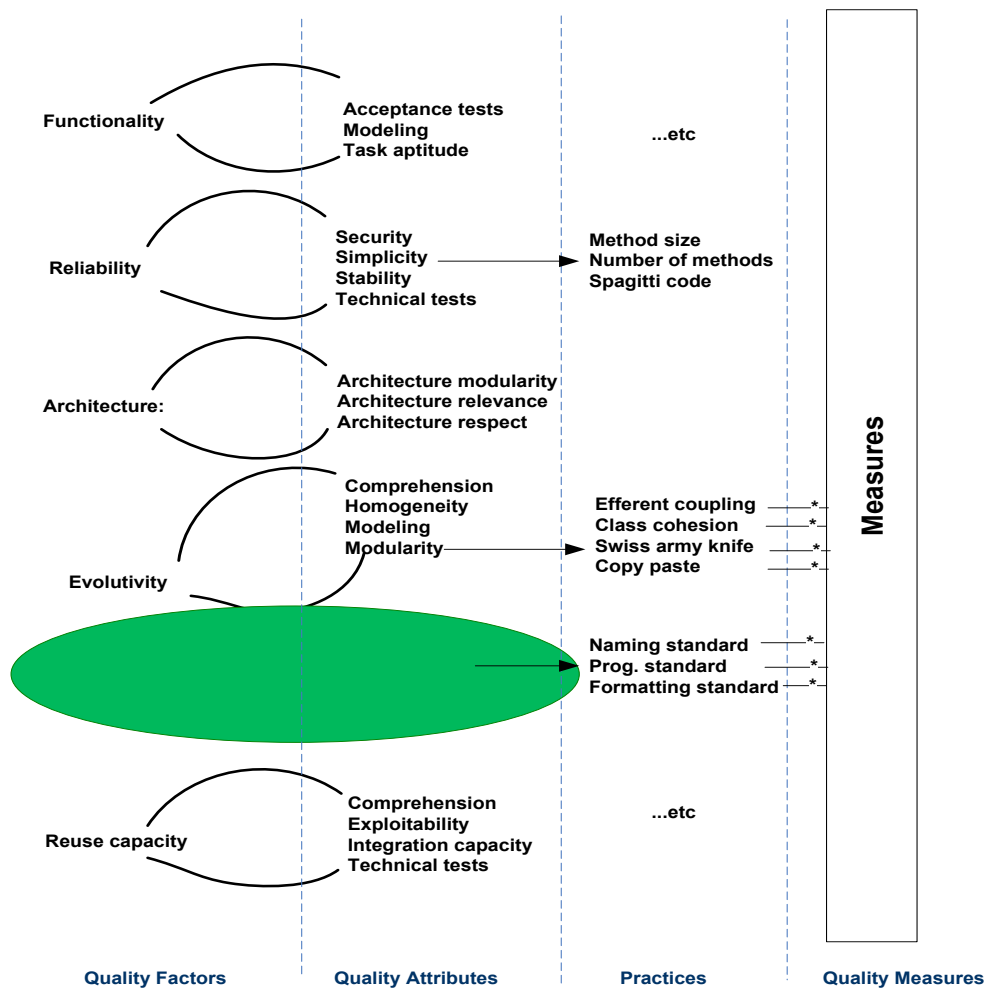


FIGURE 17 SQUALE QUALITY MODEL [68]

When mapped to our generic metamodel, SQALE adds new level, called practices, to bridge the gap between attributes (criterion) and measures. An example of practice is that “complex classes should be more documented than trivial ones.” [68]. The Air France quality standard defined 50 different practices<sup>10</sup>. To calculate a practice, different measures are combined; for example, the comment rate practice combines comment rate per method LOC and the cyclomatic complexity results.

SQALE does not provide explanations for some of the benchmarks/thresholds used in finding the practice scores, e.g., the comment rate practice verifies that a method should include at least 30% of comments. It also depends on human analysis for a subset of the practices, e.g., quality assurance plan availability and functional security.

#### 2.4.8. SQALE

Software Quality Assessment based on Lifecycle Expectation<sup>11</sup> (SQALE) is a quality model that focuses on source code quality based on lifecycle expectations. SQALE defines the quality model as a pyramid of (sub-) factors or (sub-) characteristics rather than the tree view we defined in our generic metamodel. The pyramid is arranged in such a way that the qualities at the bottom correspond to those qualities addressed earlier in the development lifecycle [69] (Figure 18).

SQALE considers bad coding practices as a technical debt. SQALE defines a set of requirements to comply with code non-functional qualities where any incompliance will be estimated as a debt. A remediation function is associated with each of the “right code” compliance requirements, which is another cost defined by the stakeholders and calculated as part of analyzing the current technical debt. Each project or organization can define a target acceptable technical debt, which can be monitored using SQALE. To pay back the debt, incompliance needs to be resolved.

---

<sup>10</sup> [http://www.squale.org/quality-models-site/research-deliverables/WP1.3\\_Practices-in-the-Squale-Quality-Model\\_v2.pdf](http://www.squale.org/quality-models-site/research-deliverables/WP1.3_Practices-in-the-Squale-Quality-Model_v2.pdf)

<sup>11</sup> <http://www.squale.org/>

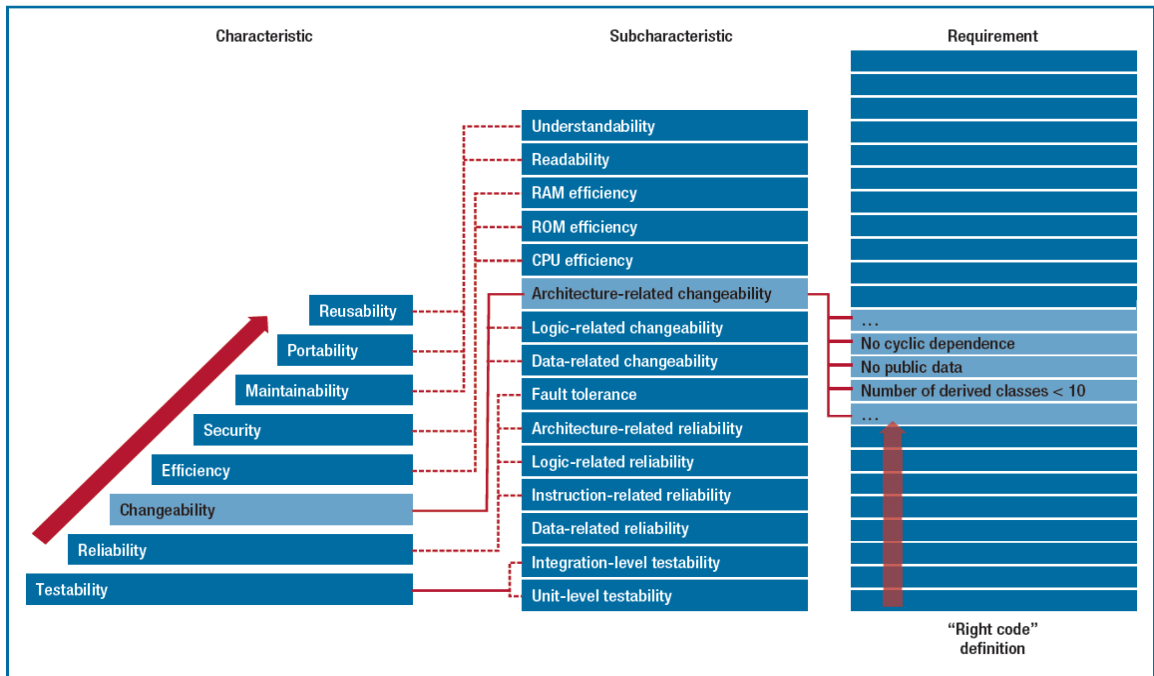


FIGURE 18 SQALE QUALITY MODEL PYRAMID [70]

The sum of all non-remediation costs is defined as the SQALE business impact index. This cost is prioritized based on a pre-defined scale (Figure 19) where the requirements with the highest cost are categorized as high priority (blocking is the highest) so that addressing them will minimize the calculated technical debt.

Category	Description	Sample type	Nonremediation factor
Blocking	Could result in a bug	Division by zero	5,000
High	Will have a high direct impact on the maintainance cost	Copy and paste	250
Medium	Will have a medium potential impact on the maintainance cost	Complex logic	50
Low	Will have a low impact on the maintainance cost	Naming convention	15
Report	Has very low impact—it's just a remediation cost report	Presentation issue	2

FIGURE 19 SAMPLE OF NONREMEDIAION FACTORS ISSUED FROM A SPECIFIC CONTEXT [70]

The SQALE model introduces a new way to look at the software quality and quality incomppliance. The focus of SQALE is the source code artifact only; it does not consider other knowledge artifacts such as project issues, activities in terms of code commits over time, or the software community. A considerable amount of manual work is needed to initiate the assessment process (e.g., [69] the user needs to

define the assessment scope, e.g., reused, reusable, subcontracted, and application internal components). SQALE also requires manual changes to the model, such as selecting the objective of the assessment and the quality measures that are appropriate for the programming language used by the software to be assessed.

#### 2.4.9. OTHER Models

**FURPS+:** The Functionality, Usability, Reliability, Performance, and Supportability (FURPS) model was introduced in by Grady and Gaswel [71] and then extended by IBM<sup>12</sup>. The plus is used to represent all other requirements, such as design or implementation constraints. FURPS+ classification addresses both functional and non-functional requirements [50]. The model does not have a quality dimension level. To assess quality, it starts by identifying quality factors and then identifies sub-factors along with measures. Evolvability is covered as part of the supportability factor (Figure 20).

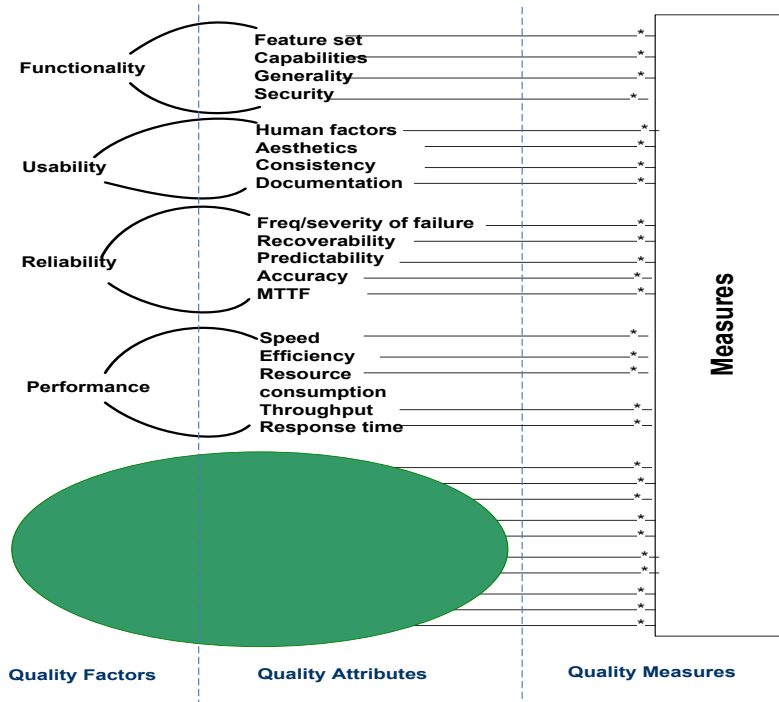


FIGURE 20 FURPS+ QUALITY MODEL

<sup>12</sup>[www.rational.com](http://www.rational.com)

**OpenBQR:** In 2007, the Open Business Quality Rating (OpenBQR) model [72] was introduced as an extension and integration of OpenBRR (2005) [73] and QSOS (2006) [55]. The OpenBQR model assesses FLOSS quality with respect to user needs by taking into consideration some of the limitations of the OpenBRR [73] and QSOS [55] models. Advancements include the use of code size and complexity internal qualities, fault proneness, external qualities, and support availability in the future. The model aims to provide a simple and formal comparison model for fast qualification and selection of FLOSS. The assessment categories supported by the model are: target usage indicators, external qualities, support availability, and functionality. One of the main challenges with the model is that there is only limited documentation about it available and, from our perspective in particular, the model does not explicitly cover evolvability as a quality factor.

**Sonar**<sup>13</sup>: An open source project focusing on analyzing code quality, such as architecture and design, comments, code rules, duplications, unit tests, potential bugs, and complexity, that was first released in 2007. It covers multiple programming languages and provides users with web-based tools and plugins. It is the only project that provides a historical view of the measures quality evolution through a feature called the time machine. Sonar reuses existing code analysis tools such as PMD, CheckStyle, and FindBugs. Sonar integrates results from other models such as the SIG Maintainability Model [64] and SQALE [70, 74].

**Calibre:** In 2004, Co-ordination Action on Libre/Free Open Source Software was introduced to coordinate the study of the characteristics of open source software projects, products, processes, distributed development, and agile methods to help improve the next generation of the software engineering tools and methods as well as establish an European Union (EU) FLOSS research policy forum. It focuses on integrating the interactions between the academic, industrial, and FLOSS communities [75, 76].

---

<sup>13</sup> <http://www.sonarsource.org/>



**FLOSSWorld:** In 2005, the FLOSSWorld<sup>14</sup> project was introduced to strengthen Europe’s leadership in FLOSS and open standards research by building a global constituency with partners from different countries around the world. It aims to enhance the global awareness of FLOSS development, industry, training, standards, and e-government issues for participating regions.

**EDOS:** In 2005, EDOS,<sup>15</sup> which stands for Environment for the development and Distribution of Open Source software, was introduced to study and solve problems associated with the production, management, and distribution of open source software packages, such as dependencies, downloads, quality assurance, and measures.

**QualiPSo:** QualiPSo,<sup>16</sup> which stands for Trust and Quality Platform for Open Source Software, is an integrated project that aims to define and implement technologies, procedures, and policies to facilitate the use of FLOSS projects within industries and governments. It is driven by the need to have trust-worthy FLOSS to make FLOSS development a widely accepted practice.

## 2.5 SOFTWARE QUALITY ASSESSMENT PROCESS

---

In this section we discuss different quality assessment or measurement processes. Process is defined as “set of interrelated or interacting activities which transform inputs into outputs” [38]. The measurement process is the “process for establishing, planning, performing and evaluating measurement within an overall project, enterprise or organizational measurement structure” [77]. The measurement method is defined as a “logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale” [77]. There are two main method types: *subjective*, whereby the measurement is based on human judgment (expert opinion) and *objective*, which is based on quantities rules. The software quality assessment process refers to the set of steps that takes the software knowledge artifacts (e.g., versioning system, issue tracker, and source code) as an

---

<sup>14</sup> <http://www.FLOSSworld.org/>

<sup>15</sup> <http://www.edos-project.org/xwiki/bin/view/Main/WebHome>

<sup>16</sup> <http://www.qualipso.org/>

input and provides a quality assessment score as an output (e.g., Excellent, Green, or 10 out of 10, depending on the quality scale used). The purpose of the quality assessment process is to assess/evaluate the quality status at a certain time stamp or snapshot, predict the future trend of the assessed quality, and improve the software quality of interest. The quality assessment processes are proposed by the existing quality assessment models covered earlier in this chapter or by individual research papers that focus on the assessment of certain metrics/measures, e.g., [78-80].

In software assessment, scale is defined as an ordered set of continuous or discrete values or a set of categories to which the attribute is mapped [77]. There are five main scale types [81]:

1. **Nominal:** classifies measure values by category and without order, e.g., classification data for gender (m/f), where nothing indicates whether m>f and it could have any label as m/f, 0/1, T/F ... another popular example is hair color (blonde, black, red, brown). This scale is often used to find the number of occurrences per category.
2. **Ordinal:** ranks measure values e.g., rate of user's satisfaction levels, and classification of defects by severity. The values are ordered, but the differences are not important. This scale is often used to sort the values in ascending or descending order. For instance, CMM defines five different levels where the order is important (level 1 is the least mature while level 5 indicates the highest level of maturity), but the distance between level 4 and 5 is not defined.
3. **Interval:** where the values of the measure have an order and are separated by equal distances. A constant scale without the true-zero value is used. The differences are not ratios (e.g., for temperature  $30^{\circ}-20^{\circ}=20^{\circ}-10^{\circ}$ , but  $20^{\circ}/10^{\circ}$  is not twice as hot!). This scale permits addition, subtraction, and averaging (e.g., the average peak temperature is  $30^{\circ}$ ), which were not applicable to previous scales.
4. **Ratio:** where the values of the measure have equal distances. This scale permits multiplication and division, e.g., height, weight, age, development cost, time between failures, and schedule length.

5. **Absolute:** a special case of the ratio scale that provides simple counts or frequencies if measured values are on the nominal or ordinal scale, e.g., number of commits, number of bugs where the absolute scale determines the equality of values obtained from all the different scales (all bugs regardless of their severity, for example).

By taking into consideration the above definitions of the types of scales, we will describe the scales used by existing quality assessment models. As per Kitchenham [27], **McCall's quality model** provides a set of questions that corresponds to a quality measure where the answer is either 'Yes' or 'No.' 'Yes' is the preferred higher quality answer. The ratio of all the 'Yes' answers to the total number of answers is then normalized to a range of 0-1. The take on this approach is the subjectivity of individual questions, difficulty to combine measures [27], and the Yes/No scale for the quality rather than more flexible ordinal measurement scale where a middle case is allowed.

The **ISO/IEC 9126 quality standard** proposes the model and the assigned measures but with no clear guidelines on how to perform the assessment process. Kitchenham [27] explains how the ISO/IEC 9126 suggests predicting some of the non-measurable qualities without any guidelines on a good prediction system.

The **QUALOSS quality model** provides a five-step assessment process (Figure 21):



**FIGURE 21 QUALOSS ASSESSMENT PROCESS STEPS**

The QUALOSS provides four ordinal scale values: Green, Yellow, Red, and Black (the details are described as part of QUALOSS project WP4 Quality Models Construction and Validation, Deliverable 4.2: metrics and indicators of the standard QUALOSS assessment<sup>17</sup>). The scale ranges from 100 to -100.

<sup>17</sup> <http://www.cetic.be/IMG/zip/Qualoss-WP4-Quality-Models-Construction-and-Validation.zip>

Green (>50 and <=100) and corresponds to no or minor evolvability/robustness risk where less the 5% of the assessed work provides undesirable or unpredictable results. Yellow (>0 and <=50) and indicates a significant risk where the amount of needed rework is less than 30% of the assessed work. Red (>-50 and <=0) and is a critical risk scale where the assessed work has major flaws and is not evolvable/reusable. This means that at most 70% of the assessed work provides undesirable results. Finally, the black scale (>= -100 and <= -50 inclusive) and indicates a prohibitive risk, which corresponds to “discard and start from scratch.” In this scale, only 5% of the existing work provides desirable results. The scores are assigned at the measures level. Upper model’s levels, such as attributes (sub-) factors scores, are aggregated after assigning weight to each using the weighted mean of the lower levels.

The **QSOS quality model** provides a four-step iterative process (Figure 22). The QSOS evaluation is performed from three different axes:

1. Functional coverage: This coverage is with respect to the functional families grid defined in the definition step. It scores 0, 1, and 2 (which is not wide enough of a range for scoring) according to the rule not covered, partially covered, and completely covered, respectively.
2. Risks from user perspective: It defines 5 main categories for the criteria to be evaluated. Under each category there is a list of criteria and sub-criteria that are scored from 0 to 2 which includes: intrinsic durability, industrialized solution, integration, technical adaptability, and strategy. For example, the user risk category *intrinsic durability* is further refined by the sub-category *maturity*, which is further classified into different *age groups*. Each *age group* has an assigned score from 0 to 2 (<= 3 months score is 0, 3 months- 3 years score is 1, and >=3 years score is 2).
3. Risks from service provider perspective: This axis of evaluation regroups the criteria to estimate risks incurred by a contractor offering services for FLOSS software. It is notable on the basis that its level of commitment can be determined. Service provider risk is the main category that defines maintainability or code mastery as subcategories, which are further divided into for example quality of code. The score for that lowest level is assigned a score of 0, 1 or 2.

The QSOS quality model has three ordinal scale levels spanning over 0, 1, and 2, where 0 is considered the lowest quality and 2 is the highest. The scores will be for the measures level of the quality model and the highest levels' score are evaluated using the calculated scores of the inferior level and the assigned weights.

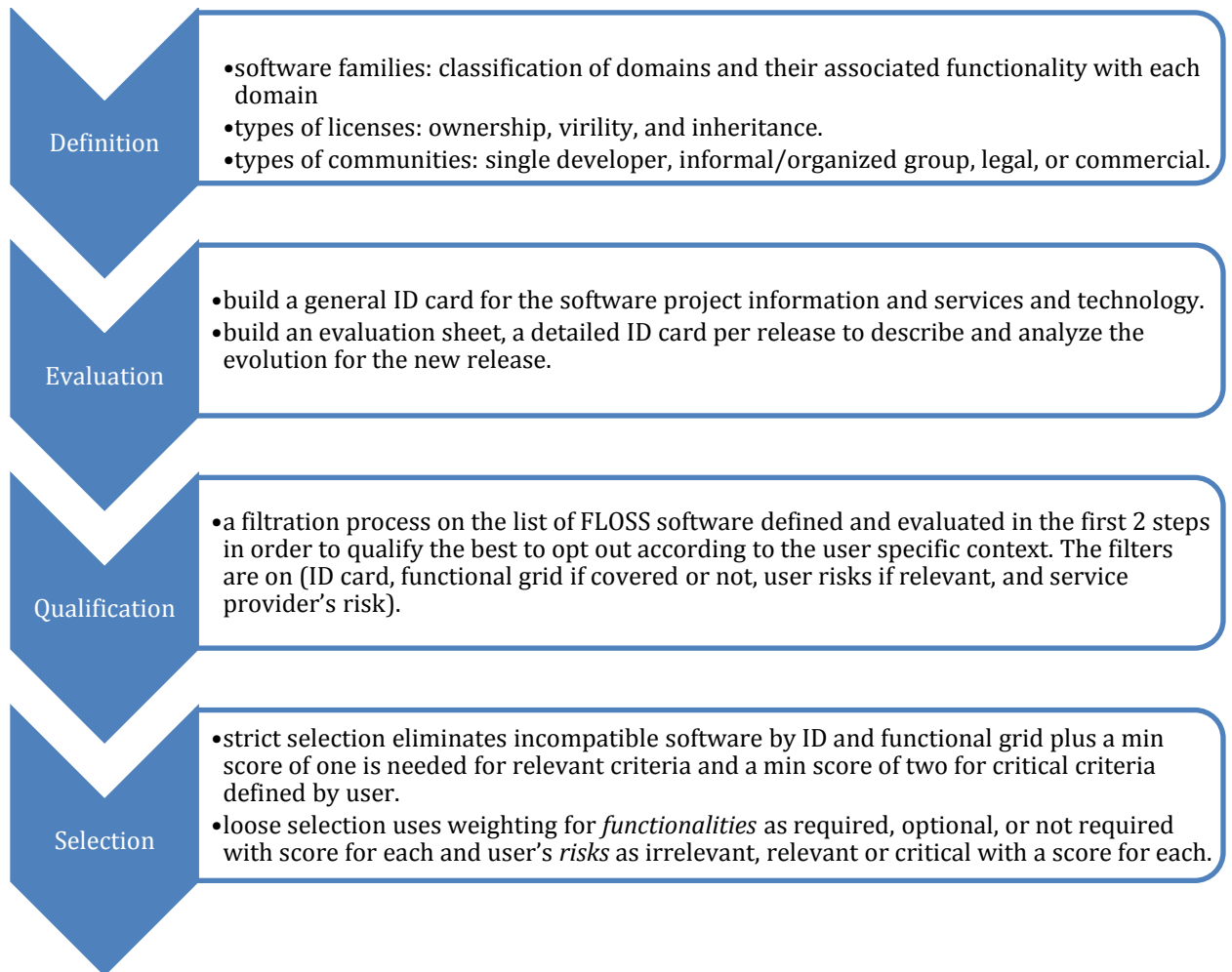
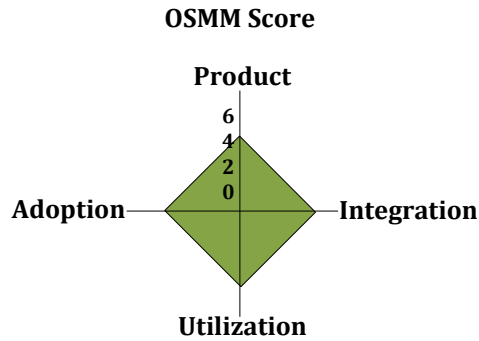


FIGURE 22 QSOS ASSESSMENT PROCESS STEPS <sup>18</sup>

The **CapGemini OSMM** performs the assessment based on two axes and four levels (Figure 23).

There are 12 criteria defined under each level.

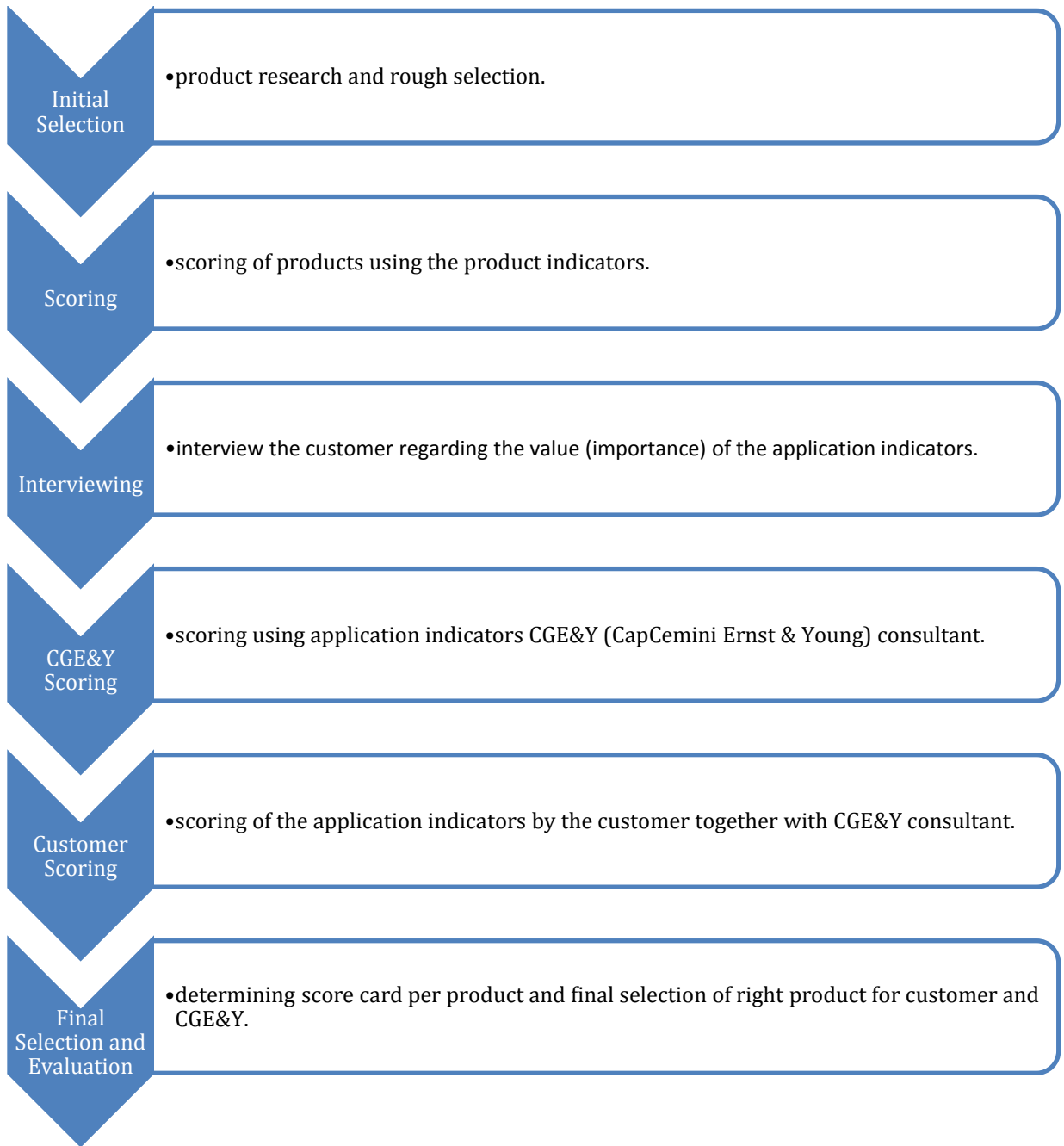
<sup>18</sup> <http://www.qsos.org/Method.html>



**FIGURE 23 THE CAPGEMINI OSMM AXES [57]**

The CapGemini OSMM score ranges from 1-5 (1 for poor, 3 for average, and 5 for excellent), then adjustable weights apply to calculate the final score. For example, the product could be either mature or immature. The assessment is performed by checking elements under product, such as age (refer to Figure 12 above). The age indicates that a product is mature if it has been active for some time and immature if it just started being active. Age scores are 1 if < 2 months, 3 if 1-2 years, and 5 if > 3 years. If a certain measure is inapplicable then a score of 3 is assigned such that it will have neither a positive nor a negative effect on the final result.

The CapGemini OSMM provides the below assessment process steps in sequential order (Figure 24):



**FIGURE 24 CAPGEMINI OSMM ASSESSMENT PROCESS STEPS [57]**

The **SQO-OSS quality model** provides an assessment process that is comprised of 4 categories and does not assign weights [42]. The categories are Excellent (E), Good (G), Fair (F), and Poor (P). Any measure that does not fit in any of the three first categories will be categorized as Poor. Below is an example of the assessment scores of maintainability quality (Table 2) [42]:



**TABLE 2 SQO-OSS MAINTANABILITY QUALITY ASSESSMENT SCORE (SUBSET) [42]**

Attribute	Measure	Profile			Notes
		E	G	F	
Analyzability	Cyclomatic Complexity	4	6	8	Less is better
	Number of statements	10	25	50	Less is better
	Comments frequency	0.5	0.3	0.1	More is better
Changeability	Vocabulary frequency	4	7	10	Less is better
	Number of nested levels	1	3	5	Less is better
Stability	Number of entry nodes	1	2	3	Less is better
	Number of exit nodes	1	1	1	One is better
Testability	Cyclomatic Complexity	4	6	8	Less is better
	Number of nested levels	1	3	5	Less is better

The **SIG Maintainability Quality Model (SMM)** provides a maintainability ordinal scale as: (++) / (+ / o / - / --) or sometimes uses \* to \*\*\*\*\*. SMM provides a 4-step evaluation process, Figure 25.

The SMM defines their measures' scale in order to assign a quality score by taking expert opinions or existing benchmarks (if benchmark exists such as Cyclomatic Complexity [64]) pertaining to the measures boundaries. Figure 26 shows the different benchmarking approaches defined by SMM in order to provide the maintainability quality scale. For example, for duplications to be rated 5 stars, they must not exceed 3%. Another approach, called quality risk profiles (similar to the one used by SQO-OSS [42]), is used for the cyclomatic complexity. Summing the units' LOC and then dividing by the total LOC in all units calculate the relative complexity.

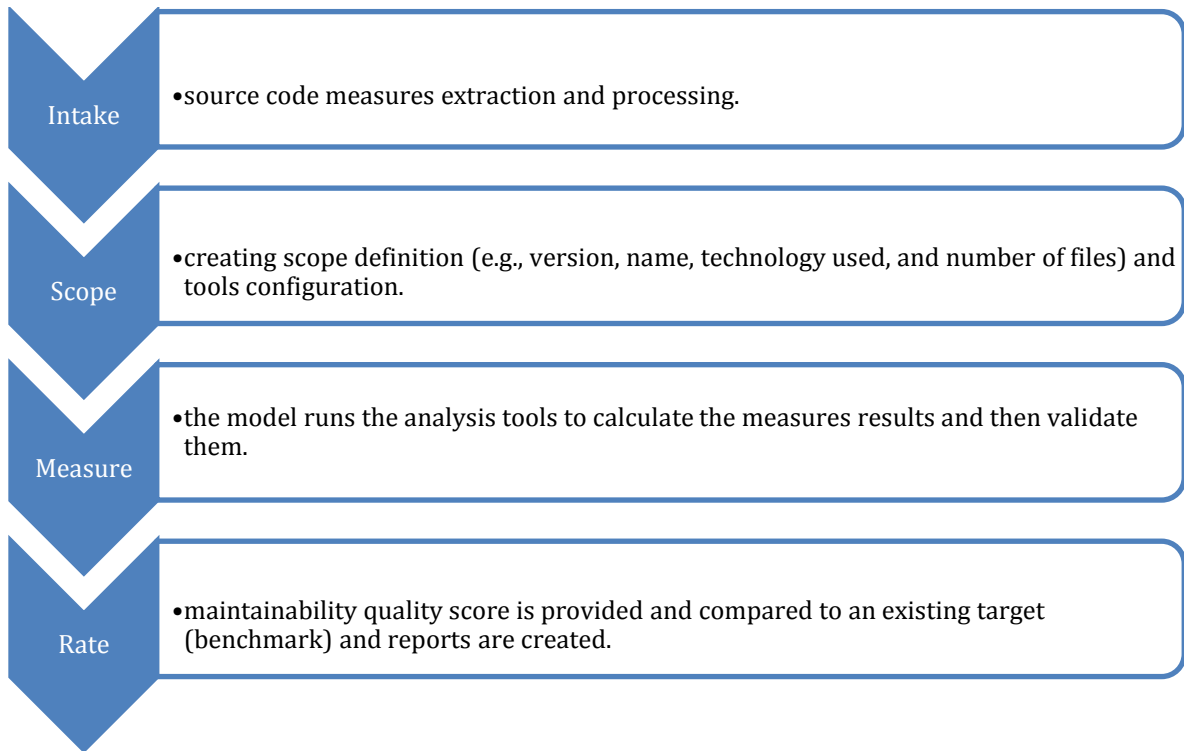


FIGURE 25 SMM ASSESSMENT PROCESS STEPS [66]

Measure	Benchmark	Rating
Duplication	3%	*****
	5%	****
	10%	***
	20%	**
	-	*
Cyclomatic Complexity	1-10	Low
	11-20	Moderate
	21-50	High
	>50	Very High

Rating	Maximum Relative Volume		
	Moderate	High	Very High
*****	25%	0%	0%
****	30%	5%	0%
***	40%	10%	0%
**	50%	15%	5%
*	-	-	-

FIGURE 26 SMM MEASURE BENCHMARKING [64]

The **SQALE quality model** provides a 7-step process (Figure 27):

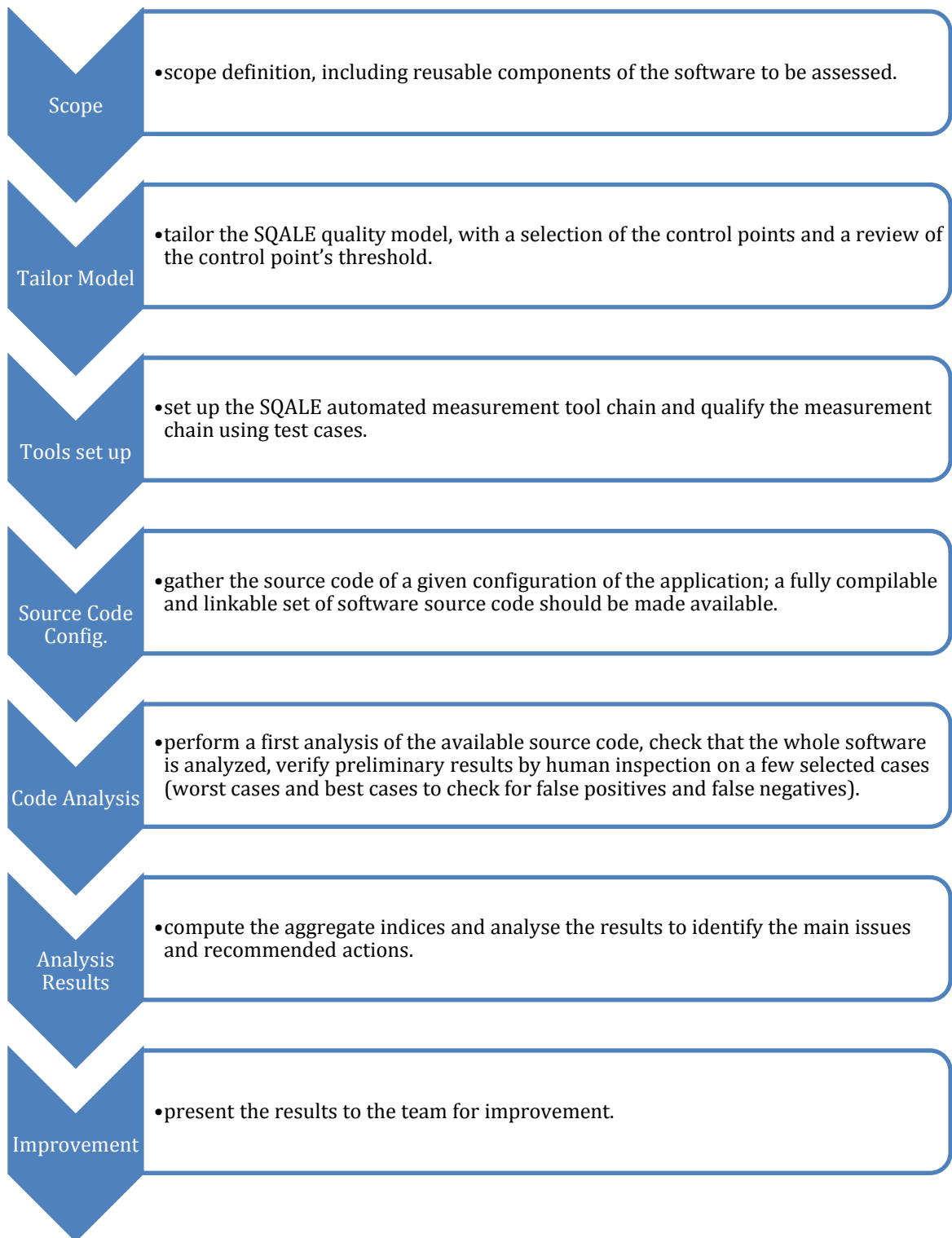


FIGURE 27 SQALE ASSESSMENT PROCESS STEPS [69, 82]

SQALE defines the assessment scale (Figure 28) based on the remediation cost versus team effort to develop the changes from scratch where red corresponds to a remediation cost above 30%. The scale ranges from [0 -> 0.9 -> 3 -> 9 -> 30 -> +∞].

From	To	Rating	Color
<	0.9	A	Light Green
0.9	3	B	Green
3	9	C	Yellow
9	30	D	Orange
30	>	E	Red

FIGURE 28 SQALE SCALE [82]

**Fuzzy Logic** has more recently been used in quality assessment, maintainability quality as in [78, 79, 83, 84], and metrics score visualization as in [85]. Fuzzy logic is a methodology based on fuzzy set theory, which is used to solve problems that are too complex to be understood quantitatively. Fuzzy logic was first introduced in 1965 by Zadeh [86] to deal with uncertainties that are not handled well in traditional mathematics. Sahraoui et al. [80, 87] have applied the fuzzy logic approach on the decision tree model. In the fuzzy logic approach, the numerical values of measures are replaced with linguistic terms (big, small, low, etc.). Any fuzzy value can be thought of as a function, called the membership function (MF), whose domain is usually specified as a set of real numbers, and whose range is the span of positive numbers in the closed interval [0,1]. Membership functions are either trapezoidal, triangular, or ramp or bell shaped. For the purpose of our research, we will use a combination of triangular and ramp shaped membership functions. There is no significant difference between the types of shapes to use.

The triangular fuzzy number is defined using three values for each state or linguistic variable, for example medium, low, and poor. These values represent the left boundary (L), right boundary (R), and modal or core (M), as represented in

Figure 29.

Fuzzy theory is defined as the degree to which a fuzzy number satisfies the given membership functions. The degree of fuzziness is calculated as  $(\text{Fuzziness of } X = (R-L)/2M, 0 < MF < 1)$  [23].  $MF(X)$  is zero if  $X \leq L$  or if  $X \geq R$ , Otherwise  $MF(X)$  is calculated as:

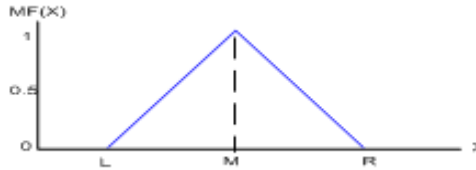


FIGURE 29 LEFT, MODAL, AND RIGHT BOUNDARIES

$$MF(X) = (X-L)/(M-L), L \leq X \leq M$$

$$MF(X) = (R-X)/(R-M), M \leq X \leq R$$

The fuzzy process takes the software measures as an input and then applies **fuzzification**, **inferencing**, and **de-fuzzification** steps to provide quality assessment results (Figure 30).

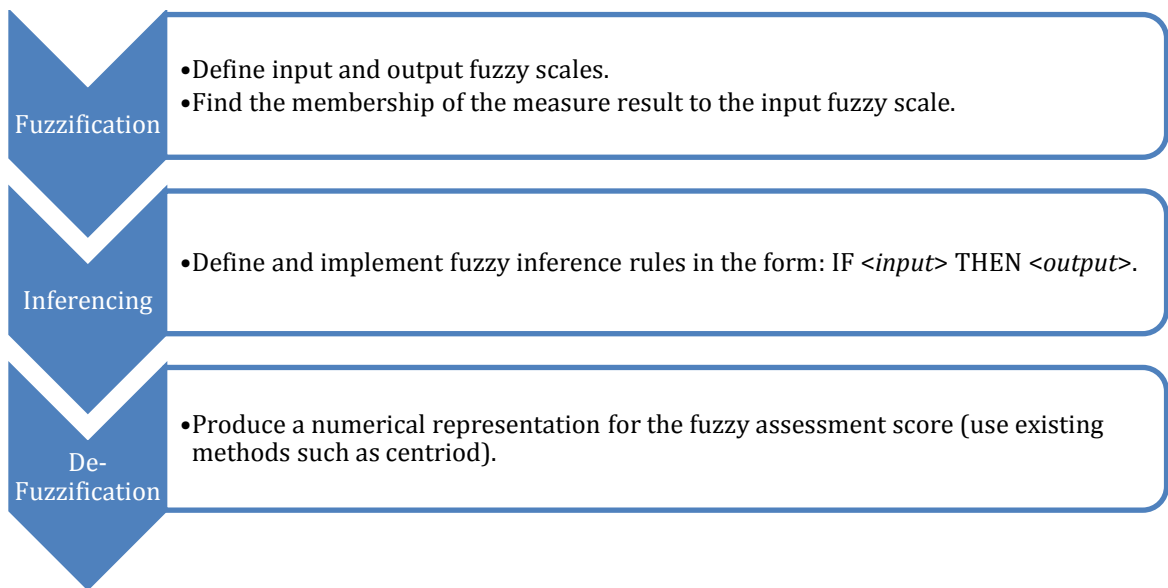


FIGURE 30 FUZZY LOGIC BASES ASSESSMENT PROCESS STEPS

As part of the **fuzzification** steps, two fuzzy scales are defined: the input scale and the output scale. **The input scale** corresponds to the scale of the measures to be fuzzified (e.g., in [84] the average cyclomatic complexity measure has an input fuzzy scale of low/medium/high, Figure 31). **The output scale** corresponds to the quality score scale (e.g., in [84] the maintainability quality score could be very good, good, average, poor, and very poor), as in Figure 32.

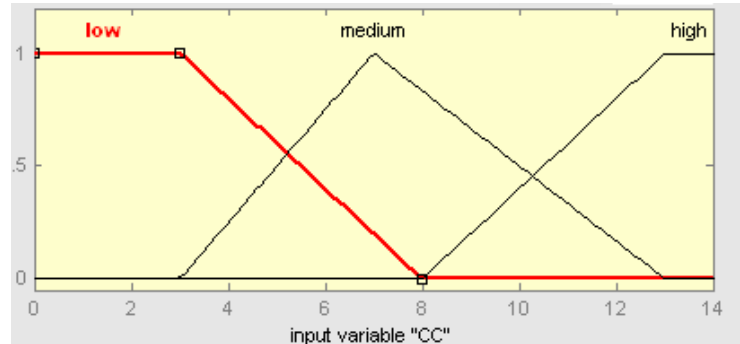


FIGURE 31 FUZZIFICATION OF AVERAGE CYCLOMATIC COMPLEXITY [84]

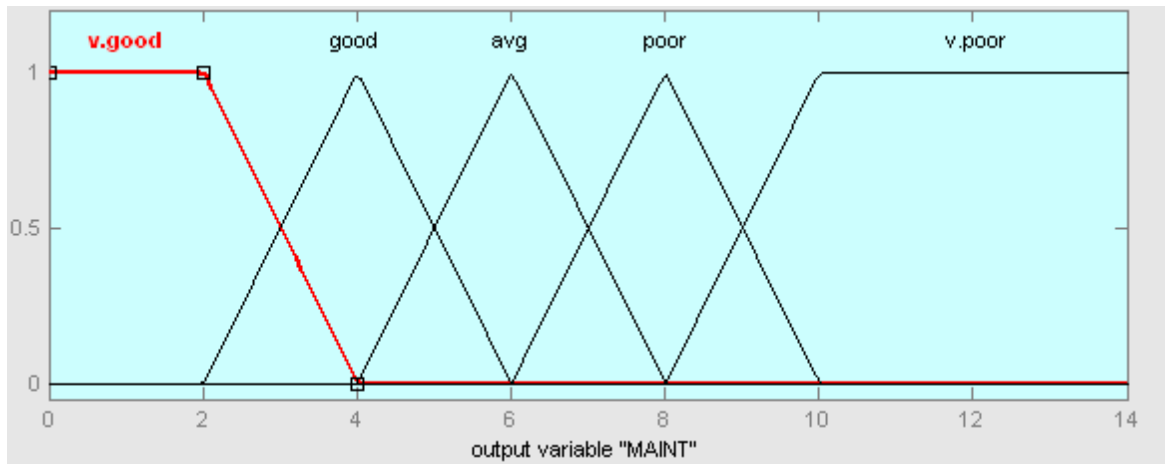


FIGURE 32 FUZZIFICATION OF OUTPUT VARIABLE - MAINTAINABILITY [84]

Fuzzy inference rules should then be implemented (the rules are usually defined by a domain expert, e.g., if the average cyclomatic complexity is low and the comment ratio is low then the maintainability quality is very poor). This step corresponds to the **inferencing** step and could be implemented using existing tools such as [88, 89]. The most popular inference style is the mamdani style [84]. The last

step is **de-fuzzification**, which calculates the center of gravity (centroid) [84] of the assessment result of all input values to finally provide a numerical score value.

## 2.6 REPOSITORY OF REPOSITORIES

---

Repository of Repositories (RoR) are publically available data collection projects used to ease the use of FLOSS data distributed among different projects and their repositories. These RoR allow for divergent representations and access formats. Data in repositories are usually gathered from different locations and converted to some form of unified data format to allow for easy querying of the stored data. Many quality models, such as QUALOSS [40, 90, 91], SQO-OSS [42, 92], and QSOS[55], use these repositories to extract data from version control, issue tracker, source code, and mailing archive repositories. RoR often contain other community related information, such as developers' names and team size.

**FLOSSMETRICS**<sup>19</sup> stands for Free/Libre Opens Source Systems Metrics and Benchmarking. It started in 2006. It is a storage location for data collected from different FLOSS project repositories. It constructs, publishes, visualizes, and analyzes an updated empirical FLOSS database. It extracts data from versioning systems, bug trackers, mailing list, and source code files. The results are provided via a tool called Melquiades<sup>20</sup>, which provides (as of May 2013) 1527 projects analysis for source code management, 581 projects analysis for mailing lists, and 1442 projects analysis for bugs trackers.

**FLOSSMOLE**<sup>21</sup> was introduced in 2006 and aims to provide raw data from FLOSS projects in many formats to help researchers in data collection and integration. It provides tools for researchers to gather their own data. FLOSSMOLE provides data about sourceforge projects and their developer's information, programming languages, and operating systems [93]. Interested users should send a request to the FLOSSMOLE project to get data and access to the database and the basic query tool

---

<sup>19</sup> <http://www.FLOSSmetrics.org/>

<sup>20</sup> <http://melquiades.flossmetrics.org/>

<sup>21</sup> <http://ossmole.sourceforge.net/>, <http://FLOSSmole.org/>

[93]. From 2004 until now, it provides 1 TB of data gathered from over 400 web-based data collections. The site does not provide a specific count for the available projects, but it claims the availability of millions of open source projects<sup>22</sup>.

**FLOSSHUB**,<sup>23</sup> which started in 2008, is a portal FLOSS research resources and discussions. Its goal is to store research papers, FLOSS data, tools, and community information to be made available for researchers and the FLOSS community. Examples of its data sources are FLOSSMetrics<sup>19</sup>, Ohloh<sup>24</sup>, FLOSSmole<sup>22</sup>, and Sourceforge Research Data Archives. FLOSSHUB does not store software data, but it provides information about where to obtain the data.

**OHLOH**<sup>24</sup> is a public wiki of open source software and people. It does not host open source projects in the traditional sense but provides a directory, a community, and an analytics service that creates reports from the extracted historical data. It offers a free API to users to create their own web services based on Ohloh, language comparison, and FLOSS comparison tools. As of May 2013, Ohloh provides data for over 589,584 open source projects.

**SECOLD**<sup>25</sup> is the first online linked data repository to represent software source code artifacts. SECOLD is an online source code crawler that ontologically represents the extracted knowledge. SECOLD introduces the use of unique URI for each of the knowledge resource (e.g., source code file, author, language...etc.). This knowledge is publically shared via linked data<sup>26</sup> endpoint. SECOLD's first release contains data from 1.5 million Java code files [94].

## 2.7 ONTOLOGIES AND MINING SOFTWARE REPOSITORIES

---

Ontologies in computer science have been widely used as “*a formal, explicit specification of a shared conceptualization*” [121]. In this context, *formal* refers to the fact that ontology should be machine-

---

<sup>22</sup> <http://flossmole.org/>

<sup>23</sup> <http://FLOSShub.org/>

<sup>24</sup> <http://www.ohloh.net/>

<sup>25</sup> <http://secold.org/>

<sup>26</sup> [linkeddata.org](http://linkeddata.org)



readable, and *explicit* refers to its ability to define concept types (vocabularies) and constraints (assumptions). *Shared* describes the fact that an ontology captures commonly agreed-upon knowledge, that is, it is not private to an individual but is accepted by a group, and *conceptualization* refers to the abstraction model used in ontologies to represent relevant concepts in a domain. Ontologies are metamodels or conceptual models that are built as a skeletal representation of a knowledge base to be reused in specific domain applications [122, 123]. Ontologies allow for the definition of basic terms and relations comprising the vocabulary of a domain as well as the rules for combining terms and relations as extensions to the vocabulary [124].

Menzies [125] defines the cost benefits of ontologies as (1) interoperability, where two components align via their ontological representation; (2) browsing/searching, where ontologies metadata adds more intellectual querying capabilities; (3) reuse of publically shared knowledge instead of building one from scratch; and (4) structured knowledge modeling using conceptualization. As a result, ontologies have become an important part of the knowledge modeling and sharing domain by acting as a nonproprietary common language. The use of ontologies in software engineering has so far mostly focused on the conceptualization of a domain of discourse and their relations. Ontologies can be applicable in all phases of the software engineering lifecycle e.g., software measurements [126], software processes [127], and conceptualizing the collaborative nature of software engineering [128]. A unified ontological representation has been introduced for the different knowledge resources within a software ecosystem, promoting the reuse and sharing of knowledge and vocabularies in a domain of discourse [129].

While there are some efforts for information sharing, there has not been considerable research progress in sharing the analysis knowledge (i.e., quality score analysis). Among the benefits of this knowledge sharing approach are the reuse of analysis knowledge and its results. In other words, instead of hard coding quality model analysis techniques within the application, they will be modeled as logical expressions (e.g., within an ontology using OWL-DL, to be explained later in this section).

Linked Data is a by-product of the Semantic Web and has been promoted to address interoperability and sharing issues for open and online datasets. In our approach, support for knowledge sharing,

integration, and interoperability are provided by an on the fly approach for inter-linking shared knowledge [94], where a *unique reproducible identifier (URI) generation schema* (Figure 33) is defined. This schema is based on a public API created as part of SeCOLD linked data<sup>27</sup> project that provides URIs for entities at different abstraction levels, such as file, project, snapshot, or multiple projects level. We refer the reader to [94] for a more detailed discussion on the URI generation.

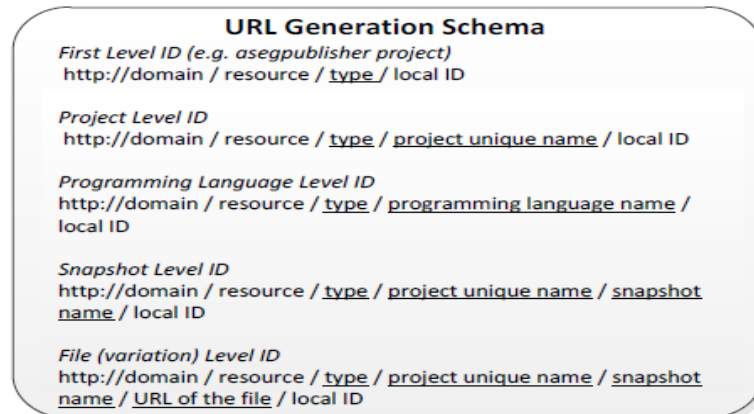


FIGURE 33 URI GENERATION SCHEMA [94]

Using this URI generation schema, extensibility (new concepts/instances and relationships) of ontologies can be achieved without the need for additional synchronization.

Ontologies support both incremental as well as incomplete knowledge modeling. In order for a model to be able to deal safely with incomplete knowledge, the open world assumption (OWA) must hold. That is, we cannot use the lack of information to infer further knowledge. Currently, existing software quality models are based on a closed world assumption, which uses approaches such as relational databases (relational algebra) whose formalism does not support OWA. The dynamic nature of the available knowledge in software ecosystems requires constant knowledge updates, i.e., incremental knowledge population. As a result of the OWA, if one cannot infer that an argument is true from the available knowledge, it can also not be false. If, by adding some knowledge, a conclusion can later be drawn, the value of the argument will be changed.

---

<sup>27</sup> <http://secold.org/>

Description Logic (DL) is a family of logic based knowledge representation formalisms used to formally represent knowledge of an application domain. DL can describe domains in terms of concepts (classes), roles (properties, relationships), and individuals (instances). DL has become a cornerstone of the Semantic Web when designing ontologies. OWL, a DL based description language for ontologies, is a commonly used knowledge representation language for the Semantic Web.

OWL has three different forms with different levels of expressiveness: OWL Lite, OWL DL, and OWL Full. OWL Lite supports primary needs such as classification hierarchy and simple constraints. OWL DL provides the maximum expressiveness, keeps computational completeness, and includes all OWL constructs with certain restrictions. OWL Full also provides maximum expressiveness but without computational guarantees. OWL Full is the union of OWL syntax and RDF. RDF stands for Resource Description Framework, which is a general method for describing and modeling the information. It expresses a statement as a triple (subject-predicate-object expressions). The predicate shows the relationship between the object and the subject. OWL2, an extension of OWL, provides additional features and improvements to OWL.

Ontologies have been used as the underlying technology for Mining Software Repositories (MSR), which is the approach that we followed in our research. First, software repositories are defined as storage locations for software data that are used to manage the progress of software projects. Versioning systems, issue tracking systems, testing repositories, and archived communication are all examples of software repositories [111, 120, 130]. Common to these software repositories is that they contain large amount of information created at various stages of the software lifecycle that is stored at different abstraction level using different semantic models.

The main obstacle of mining different repositories is the interoperability due to the differences in knowledge representation both semantically and syntactically. Syntax related problem could be resolved using data mapping, but semantic mapping is harder to resolve. Mining Software Repositories (MSR) research addresses this challenge by focusing analyzing individual software repositories over periods of time. MSR research attempts to identify relationships among different software repositories in order to support bug prediction, change prediction, program comprehension, process im-

provement and team collaboration, detect source code bad smells [8], reveal hidden dependencies among different artifacts, investigate shortcomings, support software development predication, and plan evolutionary aspects [111].

Other work more closely related to ours is the EvoOnt [131, 132] project, which provides software ontologies for the versioning and issue tracker software artifacts. In their Beatle<sup>28</sup> project they apply their EvoOnt ontologies [132] to support software by querying using limited reasoning to discover dependencies among bugs and their other modeled software artifacts within open source projects. Ontologies were used to represent information in wiki pages in order to locate and reuse information in these wikis across projects [133]. SE-ON<sup>29</sup> [134] defines a pyramid of evolution ontologies for software artifacts at different levels of abstraction such as system, domain, and concept. SE-ON [134] aims to analyze the repositories including issue tracker, source code, and code clones.

QuOnt [135] is an example for the use of ontology-based models to rate the quality of service. SMO (Software Measurement Ontology) [126, 136, 137] is a set of four sub-ontologies that define and conceptualize metrics and quality, as defined by the IEEE Std. 610.12, IEEE Std. 1061-1998, ISO/IEC 14598 series, ISO VIM, ISO/IEC 15939, and some other work in the field. As part of our research, we extend and reuse the SMO specifications. SMO will be discussed in more details later in Chapter 3.

The modeling of software artifacts (including documents) has been addressed in previous work of our research group, with a focus on establishing semantic links among various software artifacts as discussed in [12, 120, 138, 139] to provide process support for the evolution of software systems using OWL ontologies [120]. However, the focus of the previous work was on knowledge integration to support software maintenance tasks rather than on the quality assessment of software products.

---

<sup>28</sup> <http://code.google.com/p/baetle/>

<sup>29</sup> <http://www.se-on.org/>

## 2.8 SUMMARY

---

In the previous sections, different quality assessment models, including both traditional and open source models, were reviewed. The survey shows that traditional models focus mainly on well-defined processes and related aspects, while FLOSS models tend to put more emphasis on open source community aspects rather than on product itself [76].

In what follows, we provide a summary of a comparison of four traditional quality assessment models and five FLOSS quality assessment models that are either closely related or form the foundation of the assessment methodology proposed in this research. The coverage analysis is divided into five categories:

**General**, which compares the seniority of the model considering the year it was introduced and the sponsor/author of the model that could give insight into whether the model was purely academic, industrial, or a combination of these. This category also covers FLOSS specificity, community consideration, and the evolution of each model based on the older existing models. For example QSOS started in 2004; authored by Atos origin, which makes it purely industrial, it addresses FLOSS projects yet the community quality is partially covered as a separate quality (not part of maintainability quality), e.g., community availability and developers turnover rate.

The **Structure** category compares the model content; starting with the number of levels covered by each model against the generic quality assessment metamodel, it details which qualities specifically covered or missed are described under each model sub-section.

The third category summarized the **scoring model** (ranges) used for the measures in the different models to assess the quality attributes and rate the software product. The score range is usually a numeric-value, which is adjustable in most models in order to allow customization to meet stakeholder needs. Modifying the weights assigned to any level of the quality model is one form of customization. McCall [44] and QUALOSS [16, 40] are examples of adjustable models. Some models lack clar-

ification about this feature and are indicated as NA in the table below. Other models do not support such model customization by having fixed scoring rules, such as QSOS [55].

The **data sources** category compares the software artifacts that are considered in that model. This is either directly stated by the model or concluded from the set of measures provided. Source code artifacts are considered in all the models. FLOSS models consider more artifacts compared to traditional models.

The next category determines whether the model provides **tools support** to help users in the assessment process and checks whether or not the assessment process is automated. The only model that claims full automation is SQO-OSS [42]. QUALOSS claims that it provides a quality assessment tool but no implementation is provided so far [16, 40].

Finally, we provide five criteria for **model evolvability**, such as providing a formally reusable structure and knowledge representation such that new users could automatically reuse, customize, and extend the model. One form of formal representation is providing a machine-readable format for the model, such as ontology. The open world assumption is a criterion provided by ontologies and it allows incremental knowledge population in cases where the extracted knowledge necessary for the assessment is missing and incomplete at the assessment time. Finally, user defined queries provide the users with the ability to further interpret the assessment results for their own purpose (e.g., the user might be interested in only source code violations or style compliance, in which case the user can query the results of these measures separately or compare different projects from that perspective).

Furthermore, the SQO-OSS [42, 145] design covers different artifacts but the measures included in their project deliverables only cover source code artifacts. Compared to other FLOSS models, the SQO-OSS model [145] does not cover community quality, e.g., licensing, turnover, support availability, references, or books.

Despite being the only model that takes evolvability as its main focus, the QUALOSS model [40, 90, 91] does not cover some of the evolvability qualities, such as traceability, consistency, install-ability,

adaptability, and modularity. Furthermore, among the large subset of measures most of them are only manual or semi-automatic (147 manual versus 65 automatic measures). QUALOSS [40, 90, 91] included evolvability factors that have undefined assessments measures, such as mission criticality. Finally, while the QUALOSS project proposed a tool to automate the assessment, nothing is available on their website [16].

Common to most reviewed quality models is that they rely on simplified measures, such as counting instances within software artifacts. Evolvability as a quality factor has not been a major focus in most models (except for QUALOSS [91]). Existing models do not provide any guidance on the improvement of the artifact itself or the enhancement of the software development processes in order to produce higher quality artifacts and therefore a higher quality product that is easier to maintain.

**TABLE 3 COMPARATIVE STUDY FOR EXISTING QUALITY ASSESSMENT MODELS**

Comparison Criteria		BOEHM 1	MCCALL 2	FURPS+ 3	ISO/IEC 9126 4	QSOS 5	SQO-OSS 6	QUALOSS 7	SQUALE 8	SMM 9
Generality	Year	1976	1977	1987	1991	2004	2006	2007	2009	2007
	Sponsors/Author	Barry Boehm	Jim McCall	Grady & Caswell	ISO/IEC Org	Atos Origin	Athens Univ. of Economics and Business, Aristotle Univ. of Thessaloniki, ProSyst GmbH, Sirius Corporation, Klarälvdalens Datakonsult AB, KDE e.V.	CETIC, Univ. of Namur, Universidad Rey Juan Carlos, Fraunhofer IESE, Zea Partners, MERIT, Adacore, PEPITe	Ile-de-France regional council, General Directorate for Competitiveness, Industry and Services, Zerturna-round company	Software Improvement Group (SIG)
	Based-on	X	X	X	1,2	X	1,2,4	4, 5	2,4,7	4
	FLOSS specific	No	No	No	No	Yes	Yes	Yes	No	No
	Community considered	No	No	No	No	Partial	Yes	Yes	No	No
Structure	# of levels	5	4	3	4	4	4	4	5	4
	Dimension	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	No
	Factor	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Subfactor	Yes	No	No	No	No	No	No	No	No
	Attribute	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

	Measure	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Others	No	No	No	No	No	No	No	Yes (Practices)	Yes (Product properties)
Scoring model	Score range	Matrix	0-6	NA	NA	0-2	Excellent /Good /Fair	Green /Yellow /Red /Black	0-3	* to *****
	Adjustable?	No	Yes	NA	NA	No	Yes	Yes	Yes	No
	Uses weights	Yes	NA	NA	NA	Yes	No	Yes	Yes	No
Data sources	Source code	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Issue trackers	X	X	NA	NA	NA	Yes	Yes	No	No
	Version control	X	X	NA	NA	Yes	Yes	Yes	No	No
	Inline docs	Yes	X	NA	NA	No	Partial	Yes	Yes	No
	External tools	X	X	NA	No	No	No	Yes	Yes	No
Tool support	Tools availability	No	No	No	No	O3S	Alitheia Core	No	SQUALE	plugin on SonarSys
	Full automation	No	No	No	No	No	Yes	No	Yes	No
Evolvability	Formal meta model	No	No	No	No	No	No	No	No	No
	Formal knowledge sharing	No	No	No	No	NA	NA	NA	NA	No
	Custom queries	No	No	No	No	NA	NA	NA	NA	No

Here we summarize the main challenges we faced while studying the existing quality assessment models:

- Knowledge availability: some quality models, such as OpenBRR [73], QSOS [55], and QUALOSS [16], request some knowledge that would most likely be unavailable (e.g., time to set up pre-requisites, time to vanilla installation/configuration, performance testing and benchmarks, difficulty to enter the development team, developer identification, history/known problem, and independence of development).
- Knowledge consistency and incompleteness: quality assessment is based on the extraction of knowledge from a diverse set of knowledge artifacts/repositories, such as versioning systems and issue trackers. To check the inconsistency of the extracted and populated



knowledge using traditional approaches, such as relational database, is not suitable because of the closed world assumption that does not provide a separation between the knowledge completeness and consistency. (If the knowledge is unavailable then it's assumed to be false.) This approach is imprecise, giving a high probability of missing or incomplete knowledge (e.g., late adoption of version control system or common issues that are not reported as part of the issue tracker) [146].

- Addressing uncertainty: As part of either their scoring procedures or assessment scales, most of the existing quality models opt for a scale with crisp boundaries (e.g., the quality score could either be poor or very poor but nothing in between). However, if one considers the fact that input knowledge artifacts (e.g., version control system, issue tracker, source code etc.) could be missing (e.g., the version control system is only available two years after the real development started) or contain incomplete data (e.g., there are some issues/bugs that the community is aware of but never recorded in the issue tracking system), a quality scale should be able to deal with these uncertainties as part of the assessment/measurement process.
- Knowledge sharing and integration: most of the existing quality models do not provide their assessment results (e.g., quality scores) in a format that the community could further analyze or reuse, e.g., McCall [44], ISO/IEC 9126 [15], QUALOSS [91], and Boehm [48].
- Score evolution: most of the existing models (except the Sonar System tool<sup>9</sup>) assess the quality of the latest snapshot of the software of interest. Since we are dealing with the concept of “release early, release often,” [147] providing a single assessment at one point in time is insufficient to make a decision about software evolvability quality.
- Subjectivity: most of the existing models (e.g., QUALOSS [91]) depend on expert opinions and manual approaches to define the quality assessment scales (i.e., to determine what is

considered excellent or poor). This approach is subjective and makes it difficult to compare the assessment results.

- Model reusability: existing quality model design and process are usually documented in textual paper format, such as SQO-OSS [42], and/or online webpages, such as QSOS<sup>30</sup>, which hinders the ability to reuse and extend without re-implementing the model from scratch.
- Interpretation and prediction patterns: most of the models, such as McCall [44], QUALOSS [91], and SQO-OSS [42], provide the user with the quality assessment score without further interpretation of the current results or the ability to predict score trends. Evolution patterns represent recurring trends in the past and their effect on the near future trends. Based on the observation that history repeats itself, these recurring patterns are indicators of the future. For example, if we observe that whenever the evolution trend of a certain quality of interest dropped below a value X, a strong improvement followed, we can conclude that the same behavior will reoccur. This helps us proactively analyze that pattern for process improvement purposes.

---

<sup>30</sup> <http://www.qsos.org/Method.html>

## CHAPTER 3: SE-EQUAM, AN EVOLVABLE QUALITY METAMODEL

A *domain model* is a conceptualization of a domain problem in terms of its entities, properties, relationships, and constraints. In software, several domain models exist that are capable of representing and assessing predefined sets of qualities, e.g., McCall [44], ISO/IEC 9126 [15], and QUALOSS [91]. All these domain models share a common, while informal (not machine-readable), structural representation of the qualities they are assessing. While capable of assessing qualities in a given context, they lack the formalism and semantics (inferring implicit knowledge that could be extracted from the assessment and future prediction) required to allow these models to evolve and therefore make them reusable for different assessment contexts.

As Lehman [9] already stated in his law of continuous change, a software system needs to adapt to continuous changes otherwise it becomes less satisfactory and its quality declines. This law of continuous change is also a key motivation for SE-EQUAM, an Evolvable QUALity Metamodel that addresses the need to derive a formal (machine-readable) domain model that can adapt to changes, in our case changes to the software artifacts that are assessed.

A key objective of this thesis is to derive a quality metamodel that is not only capable of dealing with continuous change but also allows for its reuse by simplifying the instantiation of new domain model instances. The reusability of SE-EQUAM is supported by its ontological representation. Ontologies are used to conceptualize the structure of SE-EQUAM (quality factors, sub-factors, attributes, measures, weight, and relationships); its input artifacts, such as version control system and issue tracker; and its outputs, such as the quality assessment score. Ontologies not only provide a formal way to represent knowledge but also eliminate ambiguity, enable validation, and provide a consistency-checking approach [148].

SE-EQUAM uses semantic reasoners, in our work Pellet [149], to infer hidden relationships between domain model attributes. More details about the semantic reasoning are provided in the next section.

Our approach incorporates the unified formal representation of knowledge resources [120] and the Semantic Web (such as ontologies with added semantics and reasoning for implicit knowledge) to create SE-EQUAM [12], a quality metamodel that has been designed from the ground up to support domain model evolvability by reusing, integrating, and sharing existing models and their knowledge. SE-EQUAM is based on a formal ontological modelling that allows for a semantic rich and human-interpretable representation. Semantic richness is the machine's ability to interpret and reason more complex meaning by shifting from the weaker taxonomical modeling to a richer, logic based knowledge representation [150]. The main contribution of the introduced metamodel design is to provide a first step towards the use of a formal assessment model in quality domain.

As part of SE-EQUAM [12], we identified the following set of complementary core requirements necessary for a model to be considered an evolvable model: *Model Reusability, Knowledge Modeling, Knowledge Population, and Knowledge Exploration* [12]. In what follows, we will describe each of these requirements in more detail:

### 3.1. METAMODEL REUSABILITY REQUIREMENTS

---

In [151] software reuse has been defined as using existing software knowledge to derive new knowledge. In [152] this definition has been extended to a set of characteristics that promote model reusability. These characteristics of model reusability include: expressiveness, reusability scope, transferability (sharable), and formalism (machine represent-ability). We define model reusability therefore as *the ability to extract sub-models that can be further refined and extended without violating the consistency and soundness of the original structure of the model from which they were derived.*

In [153] the four factors that affect the reusability quality are outlined as: *portability, adaptability, understandability, and confidence.* Here, we provide a definition for each of the reusability factors and outline how they are addressed by SE-EQUAM:

- *Portability* is the ease of transferring components from one environment to another [154].  
Ontologies, used in representing SE-EQUAM, are considered to be one of the richest formal

representations of machine-readable systems, as described by Obrst in [150]. In our implementation, ontologies are stored and transmitted using a line-based, textual format called n-triple, which is a machine independent format.

- *Adaptability* is the ease of change in a different context or functionality [154]. This is validated by the ability of a specific quality model to reuse SE-EQUAM in its own context needs. Below, we will describe how the user can reuse/instantiate (throughout the thesis, the term reuse and instantiation are used interchangeably) SE-EQUAM and extend it (by adding one more level to the SQUALE [68] model).
- *Understandability* is how easy it is for the user to understand the design in order to reuse it [154]. In reusability, the user needs to interface with the reusable component(s) as-is or must make modifications before reusing them. In SE-EQUAM, the user can perform both actions. The user needs to be familiar with ontologies in order to be able to understand and reuse SE-EQUAM.
- *Confidence* is the probability that the reused components will function as expected in the new environment [154]. One form of confidence that our solution provides is through the inconsistency checks provided by the inference engine (in our case Pellet [149]), which detects contradictory facts and reports any found inconsistencies. SE-EQUAM is also query-able, for example the user can write a query to determine which sub-factors of a specific quality factor (such as evolvability) need to be validated. The query language used is called SPARQL<sup>31</sup>. A domain expert is needed to write the SPARQL test queries.

*For a domain model to be reusable, its underlying structure and representation have to be based on a (semi-) formal representation in order to support the four factors defined above. Existing domain models like McCall [44], ISO/IEC 9126 [15], and QUALOSS [91] lack this formalism. Their model structures are represented by an informal textual specification, which does not allow for the automated instantiation of new domain models and, because of re-implementing the textual specification of the*

---

<sup>31</sup> <http://www.w3.org/TR/rdf-sparql-query/>

model structure, has limited model reuse to manual reuse. SE-EQUAM addresses this model reuse issue by defining a formal (machine-readable) ontological *metamodel* that can be instantiated and customized for the purpose of reuse.

### 3.1.1. SE-EQUAM REUSABILITY PROCESS STEPS

In order to reuse SE-EQUAM, we followed the guidelines provided by the *methontology* method [155]. Methontology is a structured method used to build ontologies [155]. Figure 34 defines each of the steps used in this method. A concrete example of each of these steps is provided in a later section.

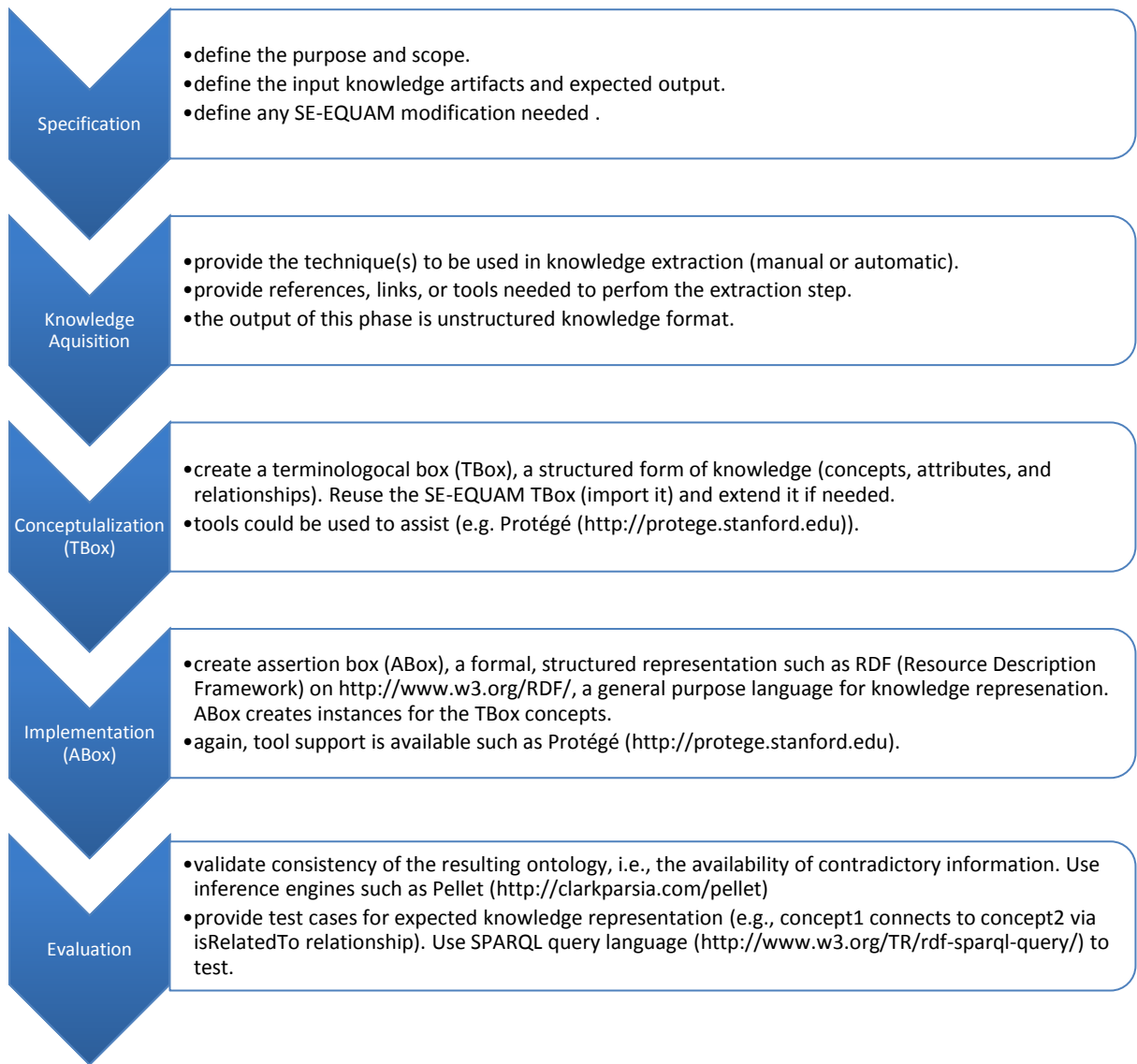


FIGURE 34 SE-EQUAM REUSABILITY STEPS

### 3.1.2 SE-EQUAM: RELATIONSHIPS BETWEEN METAMODEL, MODEL, AND ONTOLOGY

SE-EQUAM introduces a semantic mapping between the syntactic metamodel (described in Section 2.2.1) and their corresponding semantic models. Domain specific model ontology can be instantiated using the ontological metamodel [123]. Figure 35 illustrates the relationship between the different syntactical and the semantic models used in our approach.

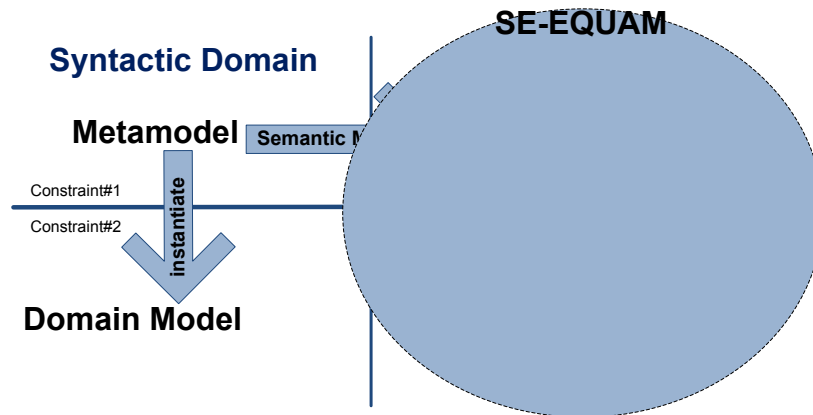


FIGURE 35 RELATIONSHIPS BETWEEN MODELS, METAMODELS, AND ONTOLOGY [123]

In our approach, we first abstract a *syntactical metamodel* for quality models by extracting a generic metamodel structure from a set of existing quality models (*domain models*), such as ISO/IEC 9126 [15] or QUALOSS [91] (Figure 36). These domain models are considered instantiations of the syntactical metamodel.

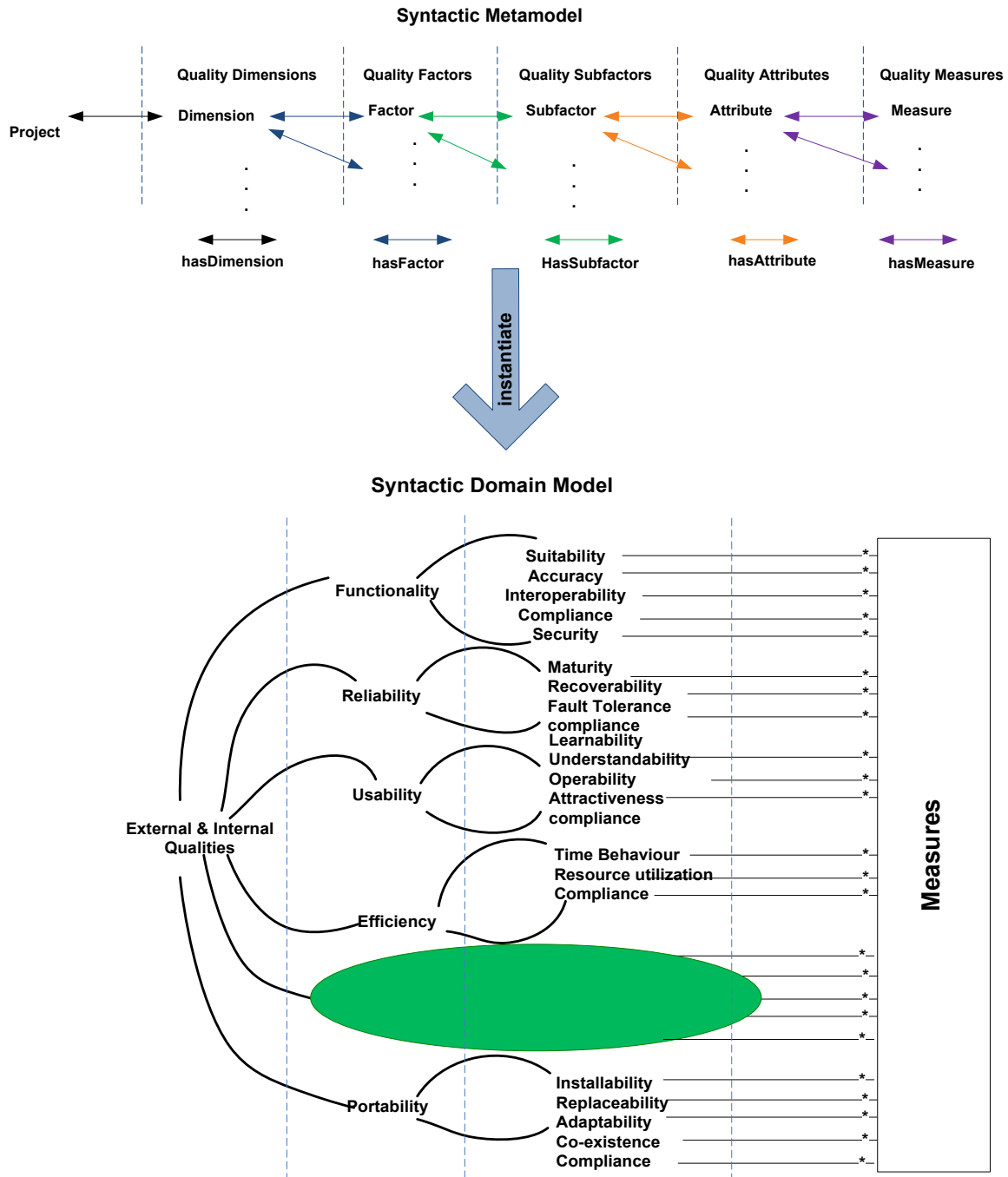


FIGURE 36 SYNTACTIC METAMODEL AND DOMAIN MODLE INSTANCE (ISO/IEC 9126) EXAMPLE

Existing approaches perform syntactic instantiation in an informal, mostly textual, form. In our approach, and in order to support model evolution and future reuse, we perform a semantic mapping from a syntactic to a semantic form. *Semantic mapping* (Figure 37) refers to the formal and unified conceptualization of the syntactic metamodel as an ontology metamodel with more semantics added



to infer new relationships between the different metamodel entities. These semantic, inferred relationships are defined using OWL DL and semantic reasoners (more details are described in the next section entitled Knowledge Modeling Requirements).

Ontology reuse has been supported by multiple case studies in the literature, which prove its cost effectiveness [157]. In our approach, the semantic mapping of the existing syntactic metamodels resulted in QUAMON (QUALity Metamodel ONtology). QUAMON is built such that it can also be reused; QUAMON itself reuses SMO [126] (Software Measurement Ontology) and its conceptualization (T-Box) of measure, attribute, and scale.

Enabling and being able to support a semantic mapping between the syntactical and semantic ontology metamodel is an essential requirement of our approach. The ontological metamodel allows us to take advantage of ontologies and their semantic expressiveness to support model evolvability.

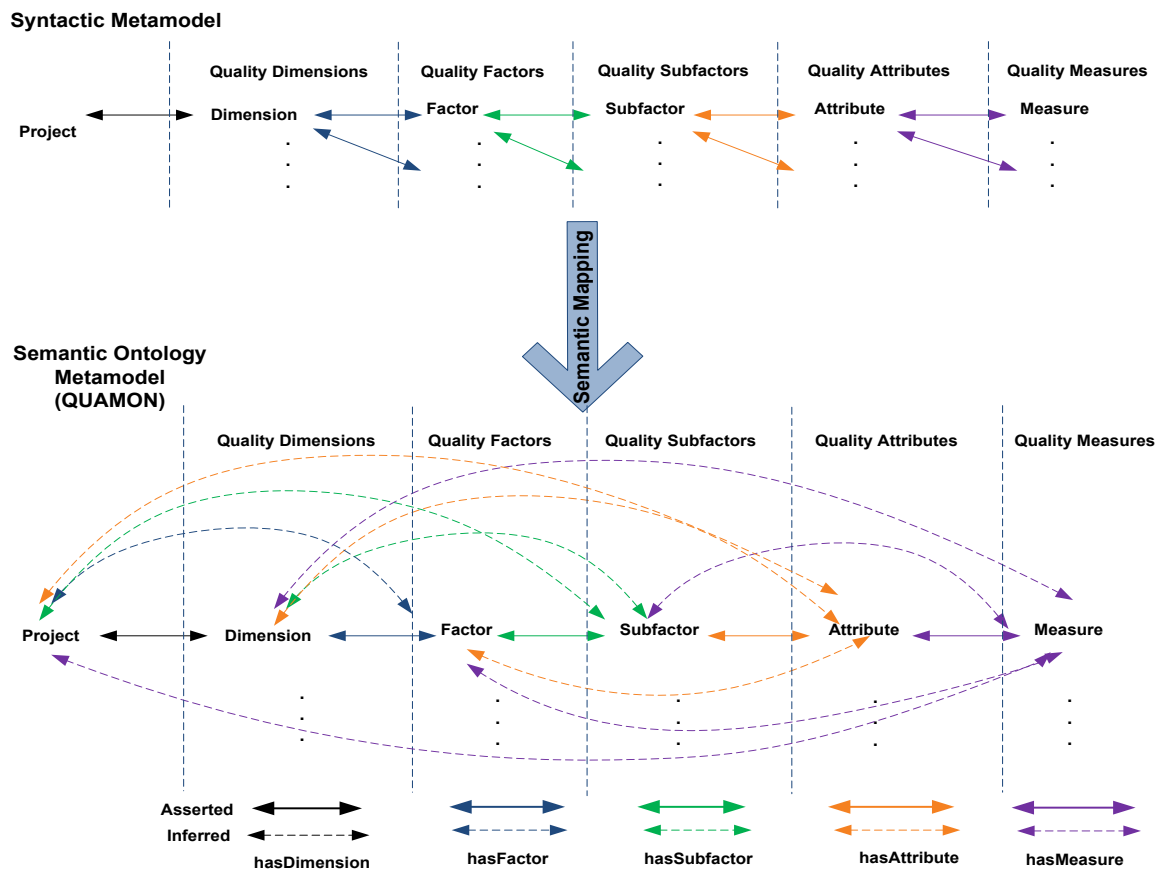
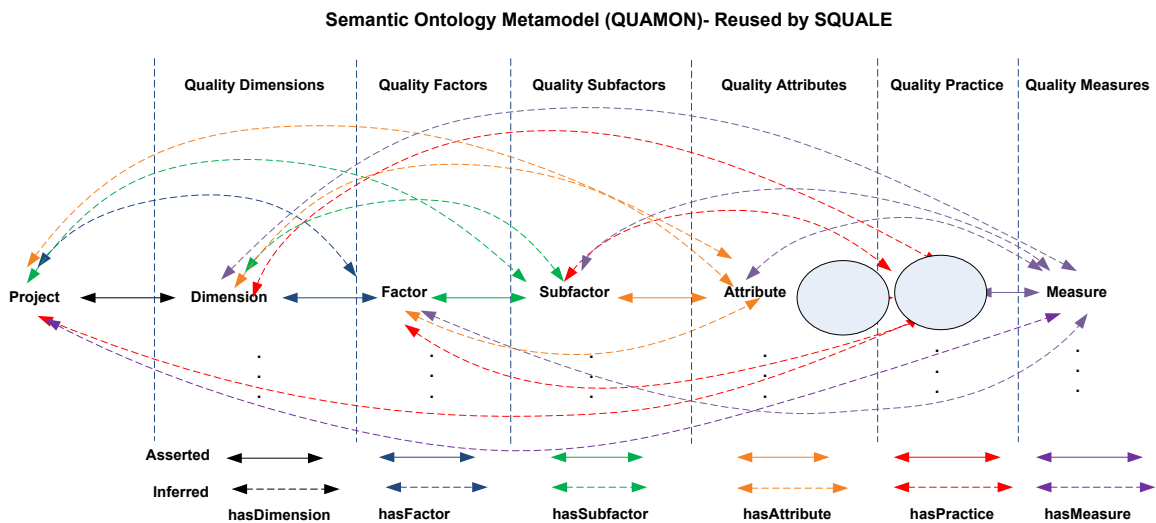


FIGURE 37 SEMANTIC MAPPING TO ONTOLOGY METAMODEL

### 3.1.3 EXAMPLES OF SE-EQUAM REUSABILITY

A domain model such as the SQUALE model [41, 67] described in Chapter 2 extends our generic metamodel by adding an additional level, called Practice, between the Attributes and Measures levels. For the SQUALE model [41, 67] to reuse our QUAMON semantic ontology metamodel (Figure 38), all that it needs to do is clone the Attribute concept to create the Practice concept and clone the hasMeasure (asserted) relationship to create the hasPractice relationship as part of the T-Box (as defined above in Figure 34). As part of the A-Box, populating individuals for the new model replaces the  $\langle :att1, quamon:hasMeasure, :m1 \rangle$  with  $\langle :att1, quamon:hasPractice, :p1 \rangle$ , and  $\langle :p1, quamon:hasMeasure, :m1 \rangle$ , where  $:att1$ ,  $:m1$  and  $:p1$  are individuals of Attribute, Measure and Practice concepts respectively. All of the *existing* asserted relationships and properties are recovered as part of QUAMON semantic modeling. For the newly added relationship i.e. hasPractice, the user must define the *new* inferred relationships (dotted lines in Figure 38) following the same approach used by hasMeasure (details are provided in the next section as part of the OWL2 property chain constructs).



**FIGURE 38 SAMPLE SQUALE REUSE OF QUAMON**

Another example of reusability would be eliminating Subfactors from QUAMON to meet ISO/IEC 25010:2011[51] model requirements. Following the same approach used by SQUALE [41, 67], any user-specific domain model could reuse QUAMON by tailoring it to its needs. Both examples mentioned here, if implemented, should not violate the consistency and soundness of the existing ontolo-

gy metamodel. A consistency check is automatically provided through ontology reasoners, such as Pellet [149].

Aside from the ability to customize of the structural design of the metamodel given in the previous two examples, metamodel reusability could be established by populating QUAMON with new user defined instances/individuals in order to instantiate and customizes new domain model ontology to create a domain specific instance of our quality ontology metamodel, for example, a domain model that addresses security or robustness qualities.

Assume that a user needs to create a domain model ontology for assessing *f1* as a quality factor, with *f1* consisting of a Subfactor *sf1* concept, which in turn has a concept *att1* as an Attribute. The concept *att1* is further refined by a concept *m1* as a Measure. The user can populate QUAMON by creating individuals (instances) of QUAMON concepts then linking these individuals using QUAMON's predefined relationships, such as  $\langle :f1, \text{quamon.hasSubfactor}, :sf1 \rangle$  and  $\langle :sf1, \text{quamon.hasAttribute}, :att1 \rangle$  and  $\langle :att1, \text{quamon.hasMeasure}, :m1 \rangle$ .

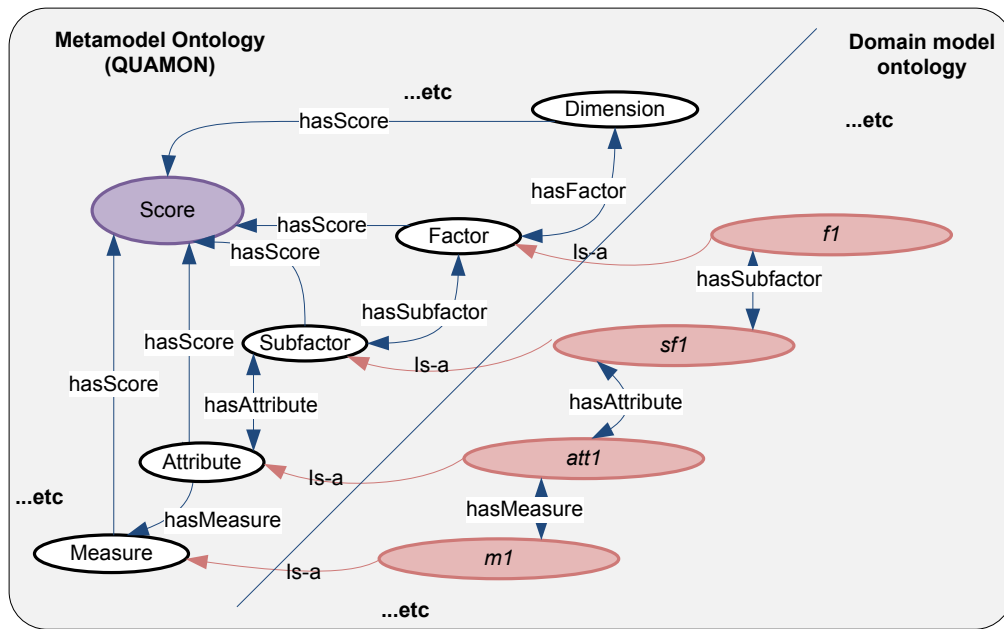


FIGURE 39 QUAMON REUSABILITY USING A DOMAIN MODEL ONTOLOGYEXAMPLE (SUBSET)

This modeling approach allows users' defined *f1* factor to automatically reuse all the inferred relationships QUAMON added to the ontological model, such as *hasSubfactor*, *hasAttribute*, *hasMeasure*, and *hasScore*.

Semantic reasoners (e.g., Pellet [149]) can be used for consistency checking to ensure the absence of type or constraint contradiction that might have been introduced by the user when instantiating a QUAMON domain model instance. Another scenario where the built-in consistency checks are beneficial is when introducing concepts in the ontology as being disjoint such as *Attribute* and *Measure* concepts. *Attribute* and *Measure* are disjoint when an individual/instance can either be a *Measure* or an *Attribute* but not both. Users might assert that *m1* (an individual of *Measure*) is the same individual as *att1* (an individual of *Attribute*) that will introduce an inconsistency that can be detected by the Pellet reasoner [149].

### 3.2. KNOWLEDGE MODELING REQUIREMENT

---

Software ecosystems involve a broad range of heterogeneous software components and artifacts at various levels of abstraction and semantics. These systems are not static and must evolve and adapt to the rapidly changing technologies (such as programming paradigms) and processes (such as agile and global source code development). Given that they are distributed, it is often impossible to identify and create a single functional reference model for software components [158]. Furthermore, cultural, social, and technological factors impact the qualities used to assess software artifacts/knowledge resources.

As a result, for a quality assessment model to be evolvable, the underlying model has to facilitate the evolvability of the knowledge base through extendibility. Knowledge modeling here refers to the conceptualization (T-Box) modeling of knowledge. Figure 40 shows a subset of QUAMON T-Box (concepts and relationships):

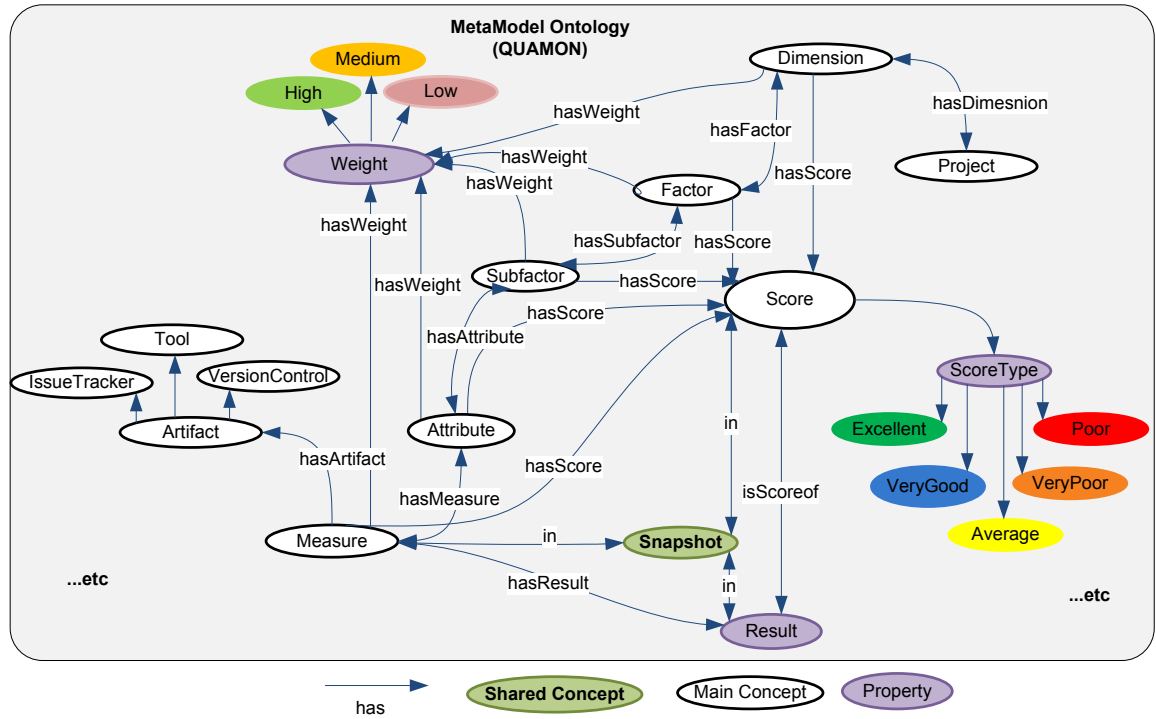


FIGURE 40 QUAMON- METAMODEL ONTOLOGY (SUBSET)

Below we provide more detailed descriptions of QUAMON's concepts (Table 4), attributes (Table 5), and relationships (Table 6). QUAMON reuses the Software Measurement Ontology (SMO) [36, 159, 160] to define certain concepts, such as score and measure. A detailed description of SMO and its reuse is provided at the end of this chapter, in the Related Work section.

The building blocks of QUAMON are the quality model hierarchy (e.g., dimension, factors, and measures), the relationships between the levels (e.g., isAttributeOf, hasMeasure), and the attributes (e.g., score has type excellent, very good, average, poor, and very poor). One of the aspects that are not represented in QUAMON is the description of the information about the model such as its scope, objective, and problems. SMO sub-ontology, called characteristics and objectives, could be reused for this purpose<sup>32</sup>. In QUAMON, the Measure concept is not subdivided into types, such as base/derived measures or direct/indirect measures. On the other hand, QUAMON applies weights for each measure; in this case, if a certain measure has a higher priority, it is assigned a higher weight. Figure 40 shows a sample of weight values as low, medium, or high.

<sup>32</sup> <http://alarcos.inf-cr.uclm.es/ontologies/smo/>

In QUAMON, we define the concept of Score. The score represents the quality rank. The score has a type and a value. We considered that these two separate properties cover the numeric value representation of score for models that rank quality in a numerical range such as 1-5, while score type is used with models that rank quality in textual terms such as red/green, good/bad, or good/poor. Figure 40 shows sample score values.

Snapshot is one of the most important concepts in QUAMON. As our research revolves around evolvability and software evolution, studying quality over time drives our solution. For any quality, the evolution of its score is tagged by a snapshot of time. The snapshot has a type that could be every 10 days, 6 months, or year. The snapshot is tagged by its end date, for example, 2008-08-28 at 20:25:28. Multiple concepts are associated with a snapshot, for example, a score is per snapshot, a contributor could be active in a specific snapshot, and a bug is issued in a snapshot.

**TABLE 4 DEFINITIONS OF QUAMON CONCEPTS**

<b>Concepts</b>	<b>Super Concept</b>	<b>Description</b>
Dimension	Thing	A way to group factors together (e.g., community or product dimensions).
Factor	Thing	A feature or characteristic of a product/service that is required to satisfy customer needs or achieve fitness for use (e.g., evolvability).
Sub-factor	Thing	The decomposition of a factor to its technical components (e.g., comprehensibility).
Attribute	Thing	Reuses SMO [36, 160]; a characteristic of a product; a measurable physical or abstract property of an entity (e.g. analyzability or testability).
Measure	Thing	Reuses SMO [36, 160] ; a measurable property of the software that contributes to assessing quality (e.g., ratio of files with code duplications).
Snapshot	Thing	A predefined period of time to extract datasets, e.g., every 6 months.
Score	Thing	The rating of quality on a predefined scale, e.g., very poor, poor, average, very good, and excellent.
Measure Value	Thing	The numerical value of the measure calculation.
Item Value	Thing	The numerical value of dimension, factor, sub-factor, or attribute calculation
Snapshot File	Thing	A version of a source code file in a specific snapshot of time.
Artifact	Thing	Artifact could be a version control system, issue tracker, or an external tool. These represent knowledge resources to extract software measures.

Project	Thing	Reuses DOAP ontology <sup>33</sup> ; represents the definition of project to be assessed.
---------	-------	---

**TABLE 5 DEFINITIONS OF QUAMON ATTRIBUTES**

Concepts	Attributes	Description
Dimension, Factor, Sub-factor, Attribute, Measure	ID	A unique identifier for each entity.
	Name	A human readable name for the asserted individual.
	Weight	The priority of the item, defaulted to the ISO standard but customizable by user (Low, Medium, or High).
Snapshot	Number	The count of the number of snapshots in a project's lifetime, e.g., a five-year-old projects with snapshot type = 12 months will have 5 snapshots.
	Type	Snapshots could be x number of days, weeks, months, or years.
	End Date	Each snapshot is tagged with the end date (yyyy-mm-dd hh:mm:ss) of the snapshot.
Score	Type	Could be good, bad, average, red, green, etc., depending on the scale.
	Value	The numerical value of the type, e.g., for a scale 0-9, 0-3 is very poor and 7-9 is excellent, etc.
	Membership	The degree of membership to a scale in a fuzzy scale system, e.g., 40% is very poor and is 60% poor.
	Assessed Value	The value for which the score is assigned, e.g., if the calculated measure m1 value >30 then the score is average.

**TABLE 6 DEFINITIONS OF QUAMON RELATIONSHIPS**

Relationship	Concepts	Description
hasName	Measure-String	The name of the measure.
hasID	Measure-String	The unique ID of the measure.
hasArtifact	Measure-String	The artifact name from which the measure value is extracted, e.g., issue tracker, external tool, version control system, etc.
hasScore	Entity-Score	The score of an entity (entity could be Dimension, Factor, Sub-factor, Attribute, or Measure).
inSnapshot	Measure-Snapshot	The snapshot in which the measure/score is calculated.
	Score-Snapshot	
hasValue	Measure-Number	The numerical value of the measure result.

<sup>33</sup> <http://usefulinc.com/ns/doap>

hasScoreValue	Score-Number	The de-fuzzified/crisp number score value.
hasScoreType	Score-String	The score type, e.g., poor, excellent, green, red, etc.
hasScoreMembership	Score-String	The score membership to the type (in fuzzy system), e.g., 40% is poor and 60% is average.
hasSnapshotEndDate	Snapshot-DateTime	The end date tag of snapshot in the format yyyy-mm-dd hh:mm:ss.
hasPreviousSnapshot	Snapshot-Snapshot	A snapshot correlate to the previous snapshot.
hasSnapshotType	Snapshot-Type	The snapshot type could be in days, weeks, months, or years.
isSnapshotOf	SnapshotFile-LogicalFile	A version of a file in a specific snapshot is correlated to its logical representation as part of VERON, which conceptualizes the version control system artifact as ontology.
hasDimension isDimensionOf	Project-Dimension Dimension-Project	The Project concept is defined as part of DOAP external ontology. Each project has one or more dimensions (perspectives) to categorize its qualities. isDimensionOf is the inverse relationship.
hasFactor isFactorOf	Dimension-Factor Factor-Dimension	Each Dimension defines one or more Factors, which are inversely related to the Dimension.
hasSubfactor isSubfactorOf	Factor-Subfactor Subfactor-Factor	Each Factor has Subfactor(s) that inversely correlate back to the Factor concept.
hasAttribute isAttributeOf	Subfactor-Attribute Attribute-Subfactor	Each Subfactor is subdivided into a set of Attributes. Each attribute is inversely correlated to one or more Subfactors.
hasMeasure isMeasureOf	Attribute-Measure Measure-Attribute	One or more measures calculate each Attribute. The measure is used to calculate one or more attributes.
hasMeasureResult	Measure-Entity	The measure result could be any concept defined by artifact ontology such as file, commit, issue, contributor, violation, style error, etc..
hasMeasureValue	Measure-Number	Some measures have their results as numbers rather than entities, e.g., LOC or issue count/ratio, and average commits over time.
hasAssessedValue	Score-Number	The score is correlated to the numerical value of the assessed entity (measure, attribute, factor ...etc.).
hasWeight	Entity-String	Dimension, Factor, Subfactor, Attribute, or Measure (entity) all have a weight assigned. The weight could be Low, Medium, or High.
hasWeightValue	Entity-Number	Each entity has a weight that corresponds to a numerical range (e.g., low is 0-3, while high is 7-9). The scale is quality model specific.
hasArtifact	Measure-Artifact	Each measure input data is extracted from an artifact. The artifact could be a version control system, an issue tracker, or an external tool. The input knowledge artifacts are external ontologies with which QUAMON integrates.



In our research, the focus is on software evolvability quality. Understanding software dependencies and relationships among heterogeneous components distributed over multiple software artifacts is a major challenge [161]. Without traceability links, knowledge is disconnected and it will require greater effort and cost to comprehend these connections and perform software engineering tasks such as reverse engineering and maintenance[161]. Ontologies support several techniques to enrich the model, by establishing traceability links across multiple ontologies. In what follows, we explore and describe three of the approaches that we use to find two types of links: (1) hidden/implicit links that we infer from explicit knowledge using description logic axioms and (2) traceability links that connect independent ontologies sharing a common knowledge.

***Linking using SPARQL queries:*** Our ontological knowledge modeling supports the use of SPARQL queries. One of our uses of SPARQL is to establish traceability links among different ontologies. One possible scenario is finding the correlation between two different concepts that share a common property. For example, the time a source code change is made to the time an issue about this file is reported. Concrete examples are provided as part of the case study in the next chapter.

***Linking through inference rules:*** Another approach for establishing traceability links between multiple ontologies. Inference rules provide an approach to infer implicit knowledge based on existing knowledge. As mentioned previously, we use Pellet reasoner [149] to fire the inference rules and enrich the ontology with new semantic links.

***Traceability links through DL axioms:*** OWL DL axioms provide semantically rich properties that can be used to infer implicit traceability links, and, similar to inference rules, the new links are recovered using a semantic reasoner, such as Pellet [149] in our case. Below, we provide a concrete example of implementing DL axioms on the metamodel ontology. In this example, metamodel levels are linked automatically to lower levels using DL property chains. The result of this linking at the metamodel level is shown in Figure 37 (above), including the added semantic based on asserted (explicit) knowledge and the inferred (implicit) knowledge that can be extracted from QUAMON.

In our approach, we also take advantage of the new OWL2 *Property Chain* construct to model knowledge inference through a chain of connections between a series of individuals. For SE-EQUAM, we asserted five types of relationships, on which we apply the construct of property chain to infer the inter-relationships among the quality model levels (Table 7).

**TABLE 7 OWL2 PROPERTY CHAIN USAGE**

Inferred Relationships	Property Chain Constructor
Each <i>Project</i> has one or more Factors, Subfactors, Attributes, and Measures.	SubPropertyOf( ObjectPropertyChain( :Project :hasFactor ) :Factor )
	SubPropertyOf( ObjectPropertyChain( :Project:hasSubfactor ) :Subfactor )
	SubPropertyOf( ObjectPropertyChain( :Project :hasAttribute ) :Attribute )
	SubPropertyOf( ObjectPropertyChain( :Project :hasMeasure ) :Measure )
Each <i>Dimension</i> has one or more Subfactors, Attributes, and Measures.	SubPropertyOf( ObjectPropertyChain( :Dimension :hasSubfactor ) :Subfactor )
	SubPropertyOf( ObjectPropertyChain( :Dimension :hasAttribute ) :Attribute )
	SubPropertyOf( ObjectPropertyChain( :Dimension :hasMeasure ) :Measure )
Each <i>Factor</i> has one or more Attributes and Measures.	SubPropertyOf( ObjectPropertyChain( :Factor :hasAttribute ) :Attribute )
	SubPropertyOf( ObjectPropertyChain( :Factor :hasMeasure ) :Measure )
Each <i>Subfactor</i> has one or more Measures.	SubPropertyOf( ObjectPropertyChain( :Subfactor :hasMeasure ) :Measure )

Here is an example of an OWL representation of the hasSubfactor property; it shows the chain of properties for the hasSubfactor:

```
<!-- http://aseg.cs.concordia.ca/ontologies/2010/11/quamon/hasSubfactor -->
<!-- quamon "http://aseg.cs.concordia.ca/ontologies/2010/11/quamon" -->
<owl:ObjectProperty rdf:about="&quamon;/hasSubfactor">
  <owl:propertyChainAxiom rdf:parseType="Collection">
    <rdf:Description rdf:about="&quamon;/hasDimension"/>
    <rdf:Description rdf:about="&quamon;/hasFactor"/>
    <rdf:Description rdf:about="&quamon;/hasSubfactor"/>
  </owl:propertyChainAxiom>
  <owl:propertyChainAxiom rdf:parseType="Collection">
    <rdf:Description rdf:about="&quamon;/hasFactor"/>
    <rdf:Description rdf:about="&quamon;/hasSubfactor"/>
  </owl:propertyChainAxiom>
</owl:ObjectProperty>
```

Given the semantic linking and support for T-Box reasoning of the metamodel, users can now instantiate domain model ontology by populating our QUAMON T-Box with the domain-specific individuals, which will allow users to inherit all the semantic properties captured by the metamodel. In addition,

users can then further refine their domain ontology by adding new inference rules and relationships to the model.

### 3.3. KNOWLEDGE POPULATION REQUIREMENT

---

Knowledge Population in SE-EQUAM refers to the instantiation or assertion of individuals to the A-Box i.e. ontology population. SE-EQUAM supports knowledge modeling, population, integration, and sharing by means of shared concepts and the use of our URI generation schema [94]. The URI generation schema provides a unique identifier for each software repository entity in the ontological model. It allows us to share common concepts among different knowledge artifacts (sub-ontologies); accordingly, a considerable number of traceability links are automatically provided at design time on the ontology metamodel level. As explained in Chapter 2, the URI generation schema defines entities at different abstraction levels [94]:

- Server level for individuals shared among different projects, such as programming language or version control system type.
- Project level for individuals related to a certain project, such as a file or contributor name.
- Snapshot level for individuals defined during a specific time interval of the project's life. An individual could be a quality score or a file version at that snapshot for example if the project is populated each 6 months then the end date of the sixth month is the snapshot ID.

For example, each entity on a project level has the project's unique name as part of its URI, which automatically correlates them together as a logical file. The snapshot level entity has the project and the snapshot unique names as part of the URI, which automatically recovers the inSnapshot link. One more example is correlating a file in the project to its various snapshots over time using the shared local ID (marked in blue in Figure 41). This automated link recovery integrates different ontologies together, in our example QUAMON, METON (metadata ontology for shared common software con-

cepts such as files that we reused) [120], and DOAP (description of a software project ontology that we reused) [120], as long as they are all uploaded under the same domain.

Another advantage that the URI generation schema provides is the versioning of the concepts using the month and the year as part of the URI design.

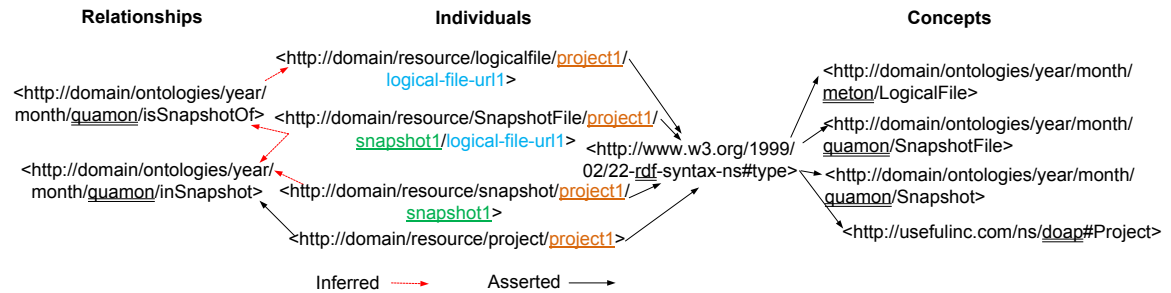


FIGURE 41 SAMPLE SET OF URI GENERATION SCHEMA APPLICATION

The URI schema provides on-the-fly knowledge integration with no added effort for synchronization [120]. Also, the URI generation schema, also called reproducible identifiers [94], provides ontological entities at any time and by any tool that further supports our metamodel design evolvability and re-usability. The ontological model could be easily extended and enriched with more knowledge, such as external knowledge resources or analysis tools results.

Given the heterogeneity and globalization of knowledge in software ecosystems, one can no longer assume that all knowledge will be available at assessment time. Domain models will have to ensure model and knowledge consistency, while having to deal with either missing or incomplete knowledge. As a result, an evolvable model should support an open world assumption [162] to deal with both incremental and missing knowledge while populating and assessing knowledge captured by the model.

**Incremental knowledge population** is a key requirement for an evolvable quality assessment. This is in particular important since not all knowledge resources might be available or complete at assessment time where new knowledge is introduced continuously, older knowledge might be missing (e.g., late adoption of version control system) and existing knowledge could be modified (e.g., new comments added to an issue report, bug status changed, or piece of code documented). The incre-

mental knowledge population is used for our continuously evolving knowledge base, e.g., adding a new input artifact to such as mailing list or an external resource such as static analysis tool results. The incremental knowledge population is supported by the ontology alignment to map the newly added knowledge with the existing knowledge through establishing semantic links and defining shared concepts (e.g., the contributor who posts or comments in the mailing list is mapped to the same contributor who resolved an issue or made a modified a file in the versioning system).

Semantic web technologies provide an enabling technology that lets us deal with both incomplete and missing knowledge. In software industry, the domain is evolvable and completeness is hard to measure. SE-EQUAM provides a step-by-step population process for the ontological knowledge resources in a concrete example in Chapter 4.

The ability to deal with an incremental population of ontologies, along with the semantic reasoners ability to provide type consistency checks, allow us to deal with the knowledge population and exploration uncertainties (uncertainty in this context refers to the lack of preciseness in the quality assessment as a result of the evolvable nature of the software ecosystem).

### 3.4. KNOWLEDGE EXPLORATION REQUIREMENT

---

An evolvable assessment model has to support different assessment perspectives to meet various stakeholder needs. An evolvable model has to allow for the customization of what (knowledge) and how (measures, factors, and quality scores) can be utilized. As a result, knowledge exploration has to go beyond answering a predefined set of common questions (queries). The model has to support both the customization of the assessment processes as well as the knowledge exploration to support model reuse and evolvability.

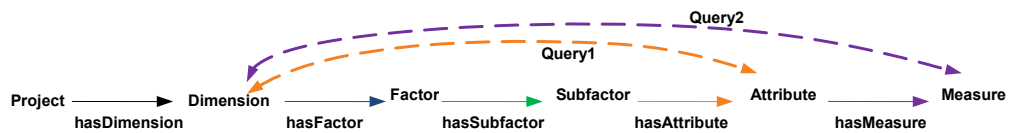
SE-EQUAM provides public access for all the populated knowledge resources online by integrating with SECOLD.org, an online linked data repository that provides a human readable representation of ontologies. As part of SECOLD, users can fetch/download data in RDF, XML, and other formats. Users can also query the provided knowledge using the query endpoint. SE-EQUAM allows for: (1) a prede-

defined set of interesting queries to answer frequent quality assessment related questions and (2) provides users with the ability to formulate their own SPARQL queries to explore knowledge from the ontology metamodel. Knowledge exploration will be supported by a SPARQL endpoint where the user can write their queries related to their specific needs.

The predefined set of queries is divided into the following categories:

**Structural queries:** These queries address knowledge exploration to answer model structure/hierarchy related questions that only use the direct/asserted relationships in the quality metamodel (e.g., relationship between factors and subfactors or attributes and measures).

**Advanced structural queries:** These queries take advantage of ontology reasoning abilities in order to infer indirect/non-asserted knowledge and relationships (e.g., relationship between dimensions and attributes or factors and measures).



**FIGURE 42 ASSERTED VERSUS INFERRED QUERY KNOWLEDGE**

Figure 42 shows an example of the asserted relationship queries (solid line arrows) versus the advanced inferred queries (dotted arrows) we are trying to answer. For example, QUAMON does not assert any explicit relationship between dimensions and measures, but we provide a set of OWL2 property chain axioms to define sub-relationships based on asserted relationships. Then, the reasoner (in our case, Pellet [149]) infers the relationship between dimensions and measures. The query, in return, includes the inferred relationships as part of their results and treats them as if they were asserted. This approach hides the reasoning details from the user.

**Assessment queries:** These queries obtain assessment related facts. For example: “What is the assessment score of a specific quality factor?” or “What is the assessment score for a specific quality attribute at a certain point of time?” “What is the assessment score for the list of X items that scored S?” or “What is the assessment score for the list of X items that scored S at Y point of time?” **Ad-**

**vanced assessment queries:** These queries go beyond the scope of the simple assessment queries by taking advantage of ontology reasoning to classify the effect of specific quality attribute on the quality score. For example: What are the models' item(s) that positively/neutral/negatively affect X? This query requires a relationship between the measures and the quality factor levels of the model. This is not an asserted relationship, but using OWL2 property chain; we can infer this relationship and answer the assessment related question using a simple SPARQL query.

The difference between the advanced structural and the advanced assessment queries is that the structural queries are interested in relationships that constitute the model's structure, while the advanced assessment queries are about score results and their effect. Both are advanced in the sense that the query results will not be available to users without the added semantic reasoning.

### 3.5. SE-EQUAM RELATED WORK

---

There are quite a few existing initiatives that propose an ontology-based approach to handle software quality assessment or measurement. The closest work to ours, and the one on which we based the definitions of our QUAMON main concepts, is the Software Measurement Ontology (SMO) [36, 136, 159]. The SMO proposal aimed to unify and formalize the definitions provided in multiple existing standards, such as IEEE Std. 610.12: "Standard Glossary of Software Engineering Terminology," IEEE Std. 1061-1998: "IEEE Standard for a Software Quality Measures Methodology," and ISO/IEC 15939: "Software engineering – Software measurement process." [36, 136, 159].

SMO [36, 136, 159] analyzed and compared the terminologies used in several international standards in order to consolidate and conceptualize the domains by providing a unified, common vocabulary (T-Box) for future use. The SMO research resulted in four sub-ontologies (Figure 43)<sup>34</sup>:

- Software measurement characterization and objectives: provide information about the measurement goal and context. This sub-ontology is about documenting a readable form of

---

<sup>34</sup> <http://alarcos.inf-cr.uclm.es/ontologies/smo/>

the measurement metadata, such as what is a quality model or what is an attribute. QUAMON does not include objectives, scope, or definitions as such as part of the model, yet QUAMON annotates all the concepts with human-readable definitions using `rdfs:Label`<sup>35</sup> and `rdfs:Comment`<sup>36</sup>. The user can import the software measurement characterization and objectives sub-ontology to enrich their representation.

- Software measures: define the measure's terminology, such as the type and scale of measure's values. QUAMON reuses the measure and measure value definition from here but it does not sub-divide the measures into base and derived types as in SMO. On the other hand, QUAMON defines several other ways to group measures by:
  - Weight: for each measure; this way measures could be grouped by their weight (Low, Medium, and High). The weight in QUAMON represents the importance or priority of the measure: the higher the more effect it has on the quality assessment. SMO does not define the weight concept as part of their design <sup>34</sup>.
  - Data sources: such as versioning system, issue tracker, or tools such as PMD [163] and CheckStyle [164]. These sources are used to extract measures used in assessment. SMO does not define data sources or knowledge artifacts as part of their design <sup>34</sup>.
  - Parent (upper) level: measures in QUAMON are the lowest level in the model hierarchy. Grouping could be by quality attributes, subfactors, factors, and dimensions that they measure. In SMO, everything aside from measure and attribute is a measurable concept [159]. We argue that attributes and measures are also measurable concepts, yet they are selectively defined as their own concepts while factors and subfactors are not. QUAMON defines each of the measurable concepts separately.

---

<sup>35</sup> [http://www.w3.org/TR/rdf-schema/#ch\\_label](http://www.w3.org/TR/rdf-schema/#ch_label)

<sup>36</sup> [http://www.w3.org/TR/rdf-schema/#ch\\_comment](http://www.w3.org/TR/rdf-schema/#ch_comment)



Users can omit any quality concept (by assigning a weight of zero to it) or extend it by adding new concepts to the model.

- Score: in QUAMON design, each measure has an associated quality score. The score could be, for example, red/green/yellow/black, such as in QUALOSS [40]. Measures then could be grouped by their score value.
- Measurement approach: defines the ways to obtain measurement results. Here the SMO defines different approaches (method, function, and analysis model) for different types of measures (base, derived, and indicator) [159]. QUAMON defines the concept measure without further detailing subtypes; hence the concept measurement approach does not apply to QUAMON. In QUAMON, the approach to calculate measures, which we will describe in the next chapter, is either the tool-based approach, where we reuse existing libraries to calculate popular software measures such as PMD [163] and CheckStyle[164], or the SPARQL-based approach, where the measure is calculated using our own SPARQL queries which run against the software knowledge base. In QUAMON, we did not define the concepts for these approaches, but the user can still find which measures are tool-based. (This is available as part of the concept URI, as we will describe in the next chapter). If the measure is not tool-based, then it is SPARQL-based.
- Measurement: defines concepts related to the measurement process, such as the approach used to calculate a measurement result. In SMO, there is an approach per each type of measure, such as based or derived [159]. In QUAMON, we model the process input (such as the knowledge artifacts used to extract measures and weights) and output (such as the quality score) but we do not design the relationship between the approach and the measurement result as part of the ontology. The reason is that we look at types of measures from a different perspective. In QUAMON, measures could have different levels of importance in the assessment process or could affect the quality positively/negatively based on their score,

which is why the approach and its relationship with the measurement result as defined in SMO does not apply to QUAMON.

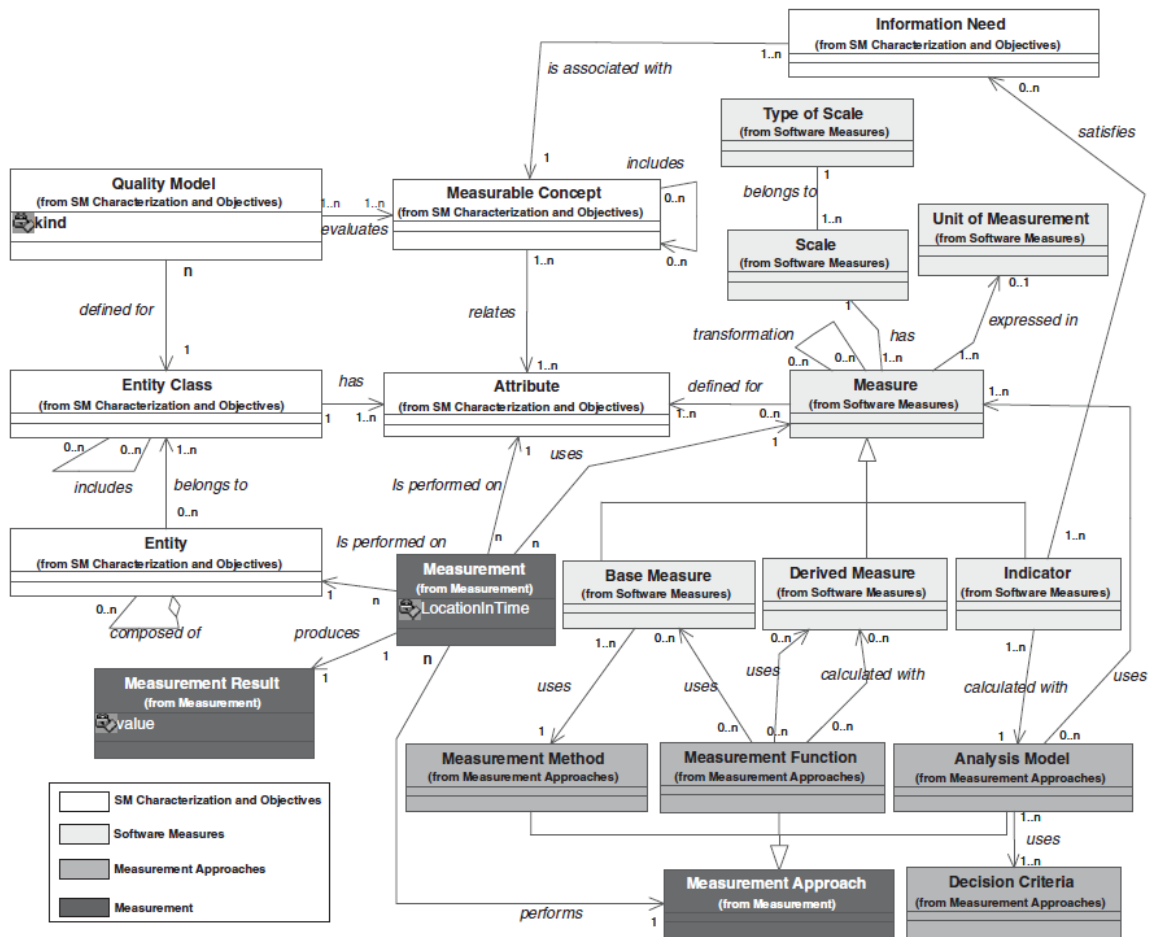


FIGURE 43 SOFTWARE MEASUREMENT ONTOLOGY (SMO) [159]

Our QUAMON ontology metamodel reuses and extends two of the SMO sub-ontologies (software measures and software measurement characterization and objectives)[36, 159]. QUAMON does not define software assessment/measurement process (e.g., measurement method, function, and analysis model and decision criteria) as part of its ontological metamodel representation, yet the assessment process output (e.g., score and measure result) is part of QUAMON. The main difference between QUAMON and SMO is that our metamodel infers indirect relationships among all the quality metamodel levels. In QUAMON, the user can choose any quality of interest and check its relationships with any other quality in the model directly and indirectly related to it. SMO lacks the indirect (in-

ferred) relationship definition. SMO does not distinguish between the concepts (dimension, factor, and subfactor), all of which it defines as a measurable concept [159]. As a result, tracing back structural model information such as measures used in assessing a certain project are complicated, if even possible, using SMO.

QUAMON defines another relationship from attribute to measure (`hasMeasure`); the same applies from attribute to measurable concept (`isAttributeOf`).

SMO defines Measurement Result concept as the value resulted from a measurement where every measurement produces only one result (refer to Measure Action sub-ontology<sup>37</sup>). Generally, in quality assessment, the measurement result is a numerical value resulting from a calculation such as counting LOC or the ratio of active to idle contributors. This numerical result is then assigned a score based on a predefined scale (e.g., if measure *m1* value > x then the score is Excellent or Green). In QUAMON, for a richer and more comprehensive ontological modeling, both values (the number and the corresponding score) should be represented. SMO allows only one of these two measurement results to be represented [159], while QUAMON defined both.

Another missing concept that QUAMON added to SMO is the weight for measure, attributes, or measurable concepts. The weight concept or property is defined as part of the ISO/IEC 9126 standard [15] and it represents the priority or importance of this quality item. Besides quality score and weight concepts, SMO lacks a representation for a date-time stamp, snapshot, or version for any of its concepts such as attribute, measure, or measurable concept [36, 159]. This is a significant concept for quality models that assess software evolution or for any model that performs multiple assessments and then compares them to older snapshots. The missing representation of score and snapshot concepts from SMO hinders further interpretation of the measurement results such as trend analysis.

SMO provides an over-generalized definition for some concepts such as decision criteria and information need. For example, in [159] paper the authors indicate that a decision criteria represents a fuzzy-scale of low, medium, and high (e.g. in

---

<sup>37</sup> <http://alarcos.inf-cr.uclm.es/ontologies/smo/>

Figure 44, the medium fuzzy term is defined by three points  $((0, 0.4), (1, 0.55), \text{ and } (0, 0.7))$ , but they do not provide details about how to populate three fuzzy terms in a single concept.

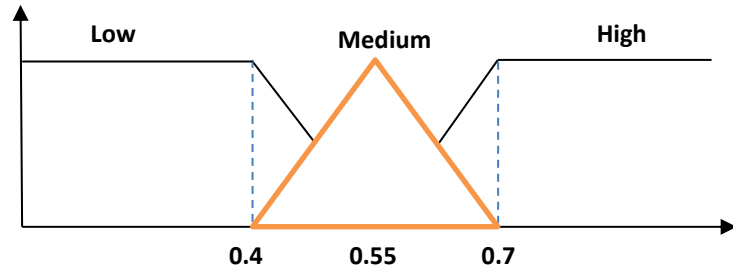


FIGURE 44 SMO DECISION CRITERIA [159]

QUAMON integrates with existing ontological knowledge artifacts (VERON, METON, SOCON, ISSUEON, and TOOLON [120]) with auto-populated links using URI generation schema [94]. SMO does not provide details about how to integrate with the input knowledge artifacts, such as source code repository or issue tracker for example. Supporting the integration with input knowledge artifacts enriches the ontological model to provide more comprehensive exploration capabilities.

Interestingly enough, in 2010 [126] there has been another research group who proposed another SMO under the same name but a different objective; we will refer to it as SMO-II. SMO-II aims to evaluate the suitability of the measurement used in organizations based on their maturity level (the maturity level is determined based on the Capability Maturity Model Integration (CMMI) with level 1 as the lowest maturity and level 5 as the highest) [137]. SMO-II also provides software measurement recommendations [126]. SMO-II claims that the existing SMO is not expressive because it is not based on a unified foundational ontology or UFO [126]. (UFO is developed based on formal ontology; philosophical logics; and language, linguistic, and cognitive psychology in order to provide a basis to develop a domain specific ontology)[126]. SMO-II is not implemented in formal modeling language such as OWL and it has been manually evaluated which makes SMO-II harder to reuse [126].

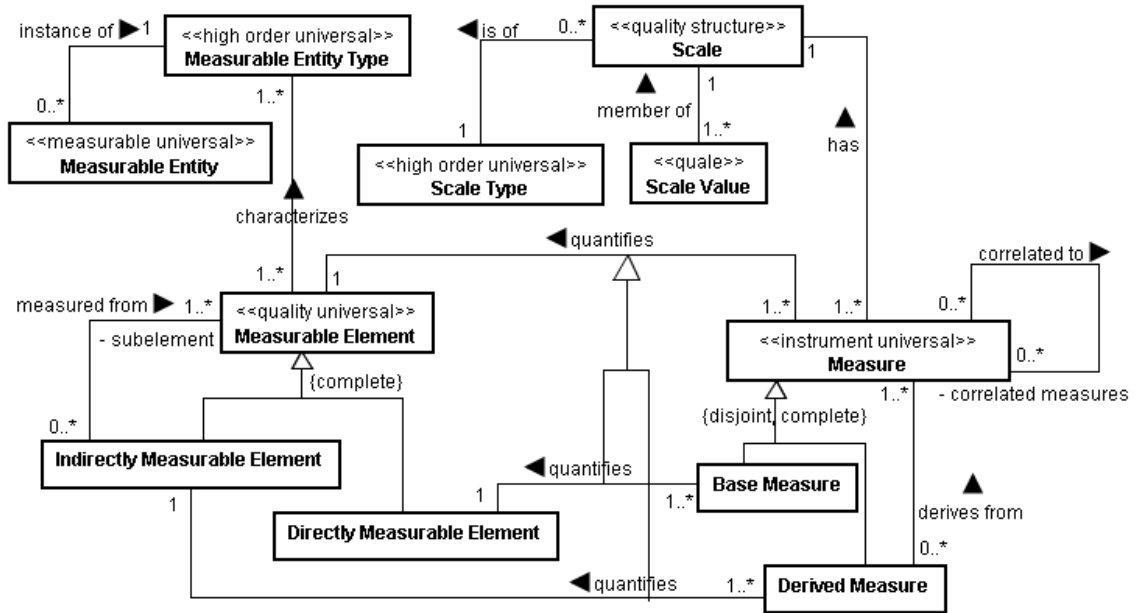


FIGURE 45 SMO-II (SUBSET) [126]

Despite following a different approach that is based on UFO, SMO-II (Figure 45) does not provide a significant change to the older SMO when it comes to the measure specific conceptualization that is relevant to our work in QUAMON. SMO-II shifts its focus to the measurement process analysis in high maturity software organizations (classified as Levels 4 or 5 based on the CMMI) [126, 137]. Software process behavior analysis is not relevant to QUAMON.

Another ontological model (Figure 46) has been proposed in [165] that aim to support automated software evaluation. The interesting part of this work is the representation of the knowledge artifacts and the way it associates artifacts with artifact versions. It also defines a Tool concept as a knowledge artifact. A comparison of the QUAMON design with the SMO design in Figure 46 shows that the QUAMON design shares the definition of concepts such as Artifact and Tool. However, in QUAMON the Tool concept corresponds to the Snapshot (e.g., if the user obtains the tool's results against annual snapshots of a software, then an x-year-old software will have x number of snapshots, each of which is dated by the end of the year date-time) and does currently no capture a version (as in SMO). The version of the tool/artifact could be added as a property of the Artifact concept. QUAMON also defines other kinds of software artifacts, such as issue tracker and version control systems.

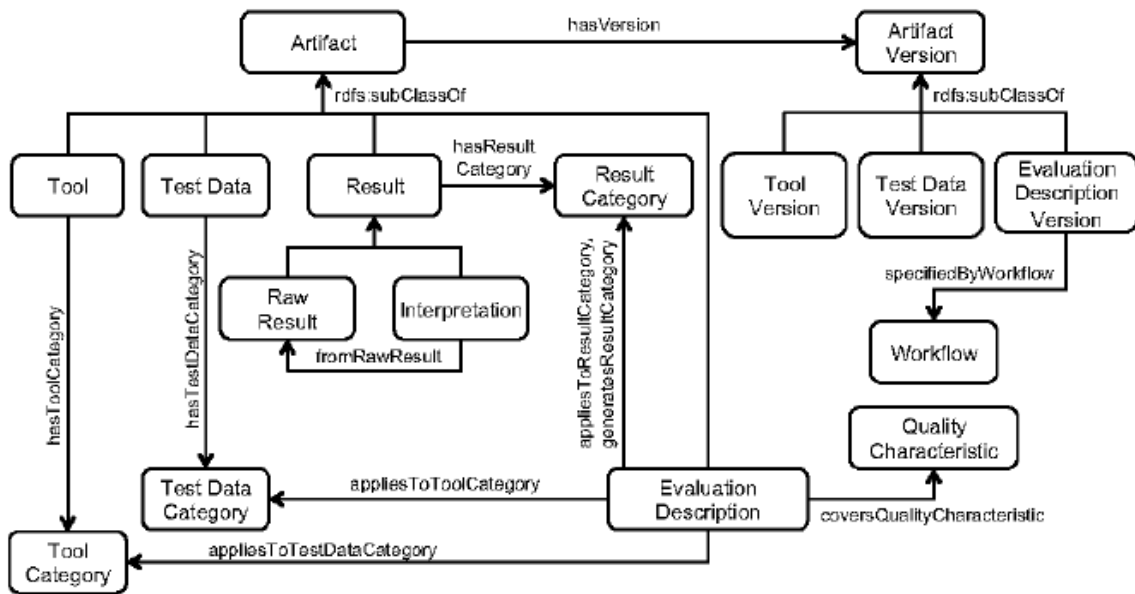


FIGURE 46 ONTOLOGY MODELING FOR SOFTWARE EVALUATIONS [165]

Martin and Oslina [166, 167] introduced ontologies for software metrics (measures) (MO-Ontology) for cataloguing web systems. MO-Ontology is implemented for WebQEM [168], a methodology with a tool support that aims to measure the quality of Web applications. MO-Ontology is a subset of SMO [166] that focuses on software measures conceptualization; therefore the same discussion of SMO applies (such as missing score, weight, and snapshot representation or the break-down of the quality model structural levels such as factor, sub-factor, and dimension). Martin and Oslina maintained a communication with the SMO [166] group in an effort to merge both of their work. Similar to the ontological model for software evaluation (Figure 46), MO-Ontology defines the Software Tool as a method to calculate Measure/Metric, which corresponds to QUAMON integration with TOOLON ontology as a knowledge artifact that provides also measures' results. (More details about TOOLON are to be provided in the next chapter).

### 3.6. SUMMARY

---

This chapter introduces SE-EQUAM, an ontology based, reusable quality assessment metamodel. SE-EQUAM provides a formal, evolvable representation, which is machine readable and includes a se-

semantic mapping, of the common core dimensions and factors of existing quality assessment models. We first identified a set of complementary core requirements necessary for a model to be considered an evolvable model: *Model Reusability, Knowledge Modeling, Knowledge Population, and Knowledge Exploration* [12]. Each of these requirements is discussed in more details to provide a foundation for adopters when reusing the metamodel. Furthermore, a discussion on related work on ontologies in quality assessment is provided, comparing both commonalities and differences among these existing approaches and our approach.

## CHAPTER 4: ONTEQAM: ONTOLOGICAL EVOLVABILITY QUALITY ASSESSMENT MODEL

**Case study objectives:** The objective of this chapter is to provide a case study to illustrate and validate that our approach is capable of meeting our requirements for a reusable quality assessment metamodel.

**Case study setting:** For the case study, we reuse the SE-EQUAM ontology metamodel presented in Chapter 3 to instantiate a domain model ontology, i.e., our Ontology-based Evolvability Quality Assessment Model (OntEQAM) [17]. OntEQAM is designed to capture evolvability quality aspects based on existing quality assessment models, such as ISO/IEC 9126 [15], SIG Maintainability Model<sup>38</sup>, and QUALOSS [91]. OntEQAM structure could be extended such as adding new subfactors, attributes or measures' assignment by extending the ontological representation. OntEQAM focuses on the integration, semantic analysis, and assessment of knowledge resources typically found in software ecosystems. We illustrate how our OntEQAM model takes advantage of the unified knowledge representation and associated semantic modeling provided by the SE-EQUAM metamodel [12] in order to assess the evolvability quality of the software ecosystems through addressing the metamodel requirements.

**Case study expected outcomes:** Firstly, we will explain the reuse of the SE-EQUAM metamodel [12] ontology, including its ontology design (concepts, relations, and semantic links) by populating QUAMON (metamodel ontology) with individuals (instances) from our ontological domain model (i.e., the OntEQAM model). This population process (which creates an A-Box) corresponds to creating a concrete instance of SE-EQUAM. In the second part of the case study, we will illustrate how users can take advantage of the semantic modeling approach used in SE-EQUAM to enhance knowledge population and exploration to answer advanced, user-specific queries about the OntEQAM instances.

Figure 47 provides an overview of the complete case study and its expected outcomes. Following a traditional approach of deriving a quality assessment model (left side of Figure 47), a syntactic quali-

---

<sup>38</sup> <http://www.sig.eu/en/Research/690/ Maintainability Model .html>



ty metamodel is extracted from existing quality assessment models. In this approach, the textual specification of the quality metamodel is then manually transformed into a domain-specific model by re-implementing the metamodel and extending it with domain-specific information.

In contrast, the semantic approach (right side of Figure 47) corresponds to SE-EQUAM and its support for model reuse and evolvability. SE-EQUAM ontological representation provides a formal (machine-readable), reusable metamodel. This ontological representation also allows for additional semantic modeling at the metamodel level by using DL axioms (such as the property chain axiom) to infer new implicit relationships in the model.

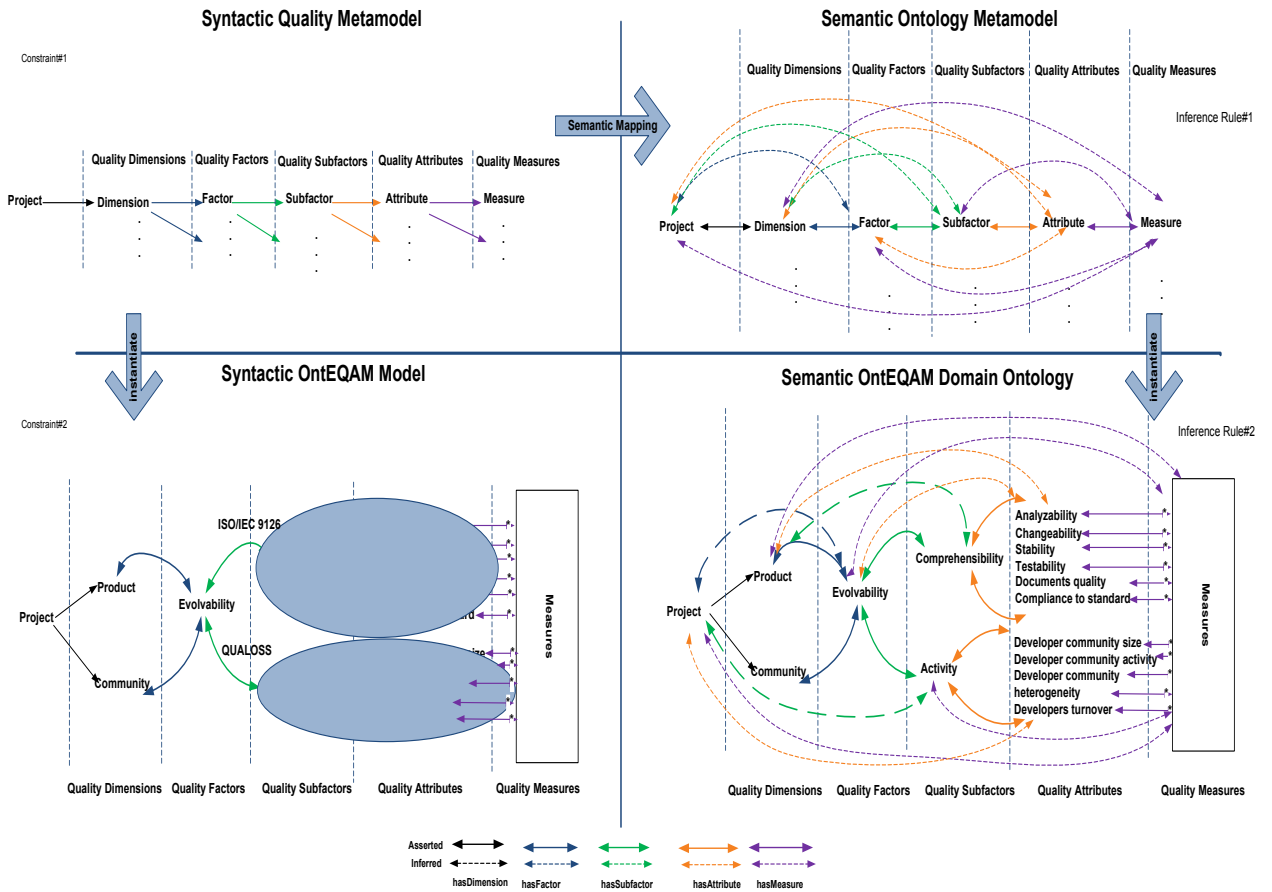


FIGURE 47 REUSE OF SE-EQUAM METAMODEL TO INSTANTIATE A DOMAIN MODEL ONTOLOGY (ONTEQAM)[12]

Below we illustrate each of the four requirements of SE-EQUAM with concrete examples to show their applicability to the evolvability domain-specific quality model (i.e., OntEQAM).

#### 4.1. ONTEQAM ADDRESSES METAMODEL REUSE REQUIREMENTS

---

The main goal of creating a metamodel is to provide a reusable framework for user-specific domain models. Reusing SE-EQUAM provides the user's domain model with a common, standardized terminological box (T-Box) that represent the basic structural relationships as exists in literature, semantically richer links, as well as the integration with input knowledge artifacts.

Below, we break down Figure 47 into Figure 48 and Figure 49 to show that when reusing the SE-EQUAM ontology metamodel, not only is the conceptual design (model structure) reused, as is done in the traditional, existing approach (Figure 48), but also the added semantic knowledge is automatically instantiated (Figure 49).

Figure 48 shows the mapping from the generic quality metamodel, as defined in the background chapter, to a concrete instance of the OntEQAM quality model [17]. OntEQAM defines the evolvability quality model structure based on the evolvability definitions of two sources (ISO/IEC 9126 [15] and QUALOSS [16, 169]). The mapping is a syntactic mapping where OntEQAM uses the definition of the asserted relationships between one level and the subsequent lower level (dimension -> factor -> sub-factor -> attribute -> measure). Basic queries (such as: What are the sub-factors associated with product evolvability?) can be answered through this mapping.

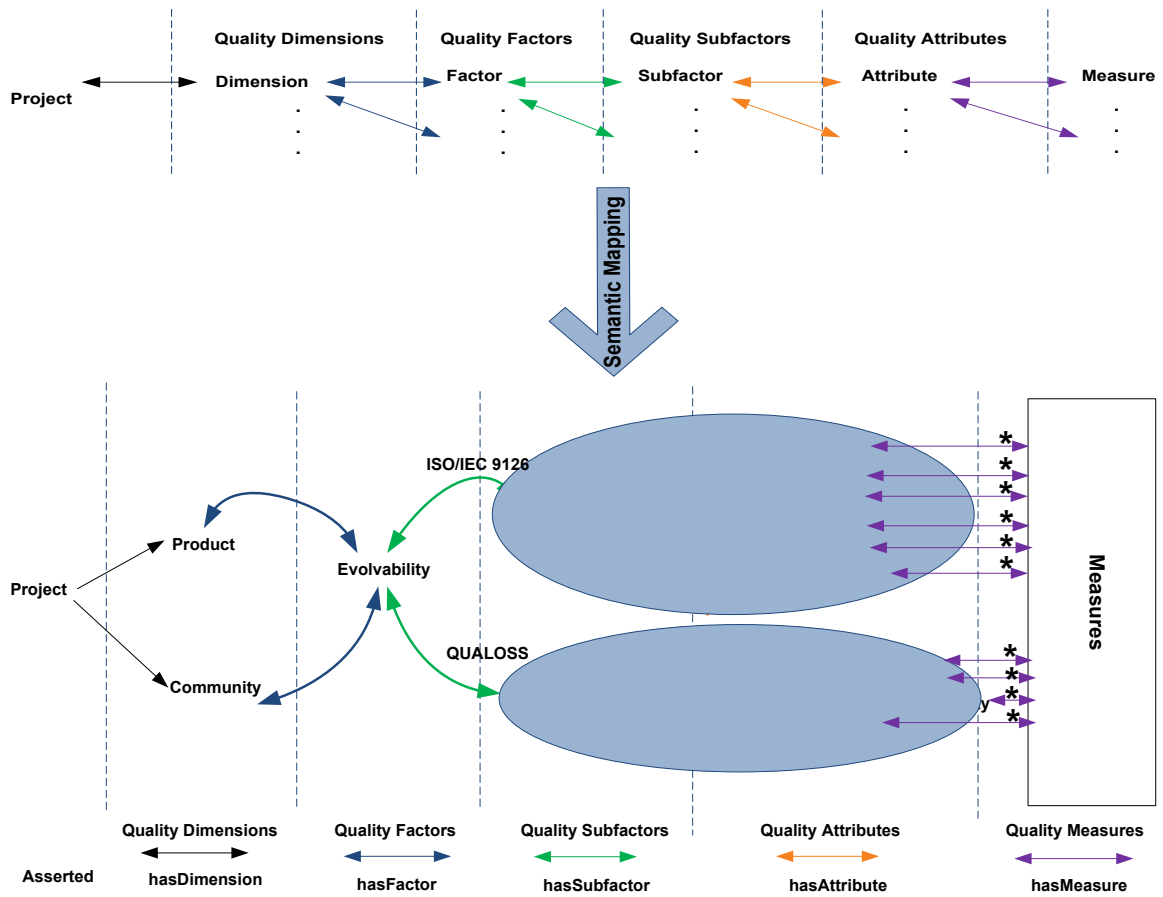


FIGURE 48 THE SYNTACTIC METAMODEL AND ITS INSTANCE, THE DOMAIN MODEL (ONTEQAM)

Figure 49 shows the asserted (explicitly modeled) knowledge in the final OntEQAM assessment model, as well as the inferable knowledge links (dashed lines), which are a result of the reuse of our ontological metamodel. In our case study, we want the OntEQAM model to inherit the syntactical and the semantic aspects of the SE-EQUAM ontology metamodel.

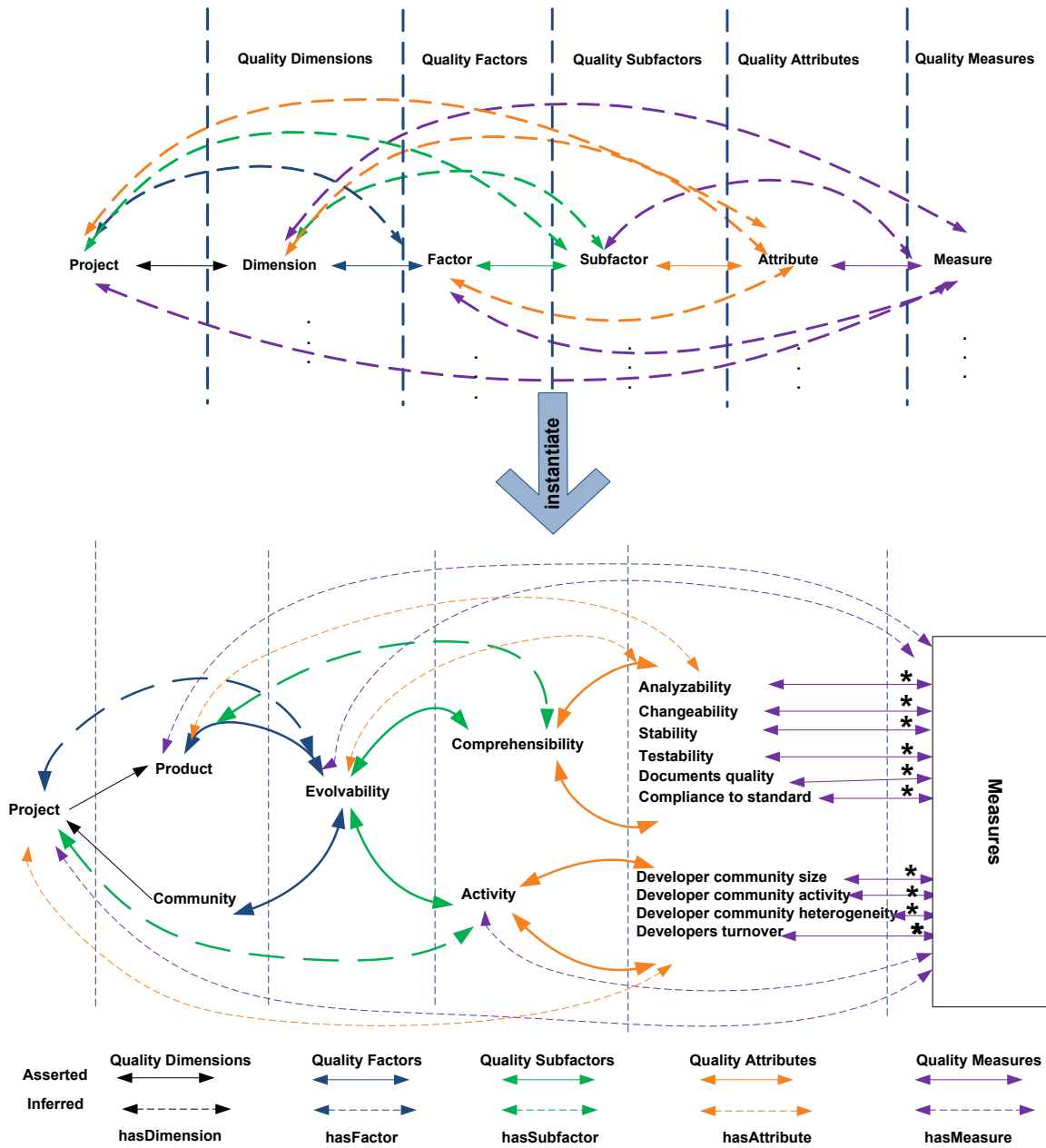


FIGURE 49 SE-EQUAM SEMANTIC METAMODEL AND ITS INSTANCE, THE DOMAIN MODEL (ONTEQAM)

Below we use a concrete example to describe how this can be achieved. In this chapter, we will be using the following prefixes in the examples of our ontology triples (triple here refers to the subject-predicate-object representation of the ontology individuals):

- doap: <<http://usefulinc.com/ns/doap#>>
- issueon: <<http://aseg.cs.concordia.ca/ontologies/2010/11/issueon/>>

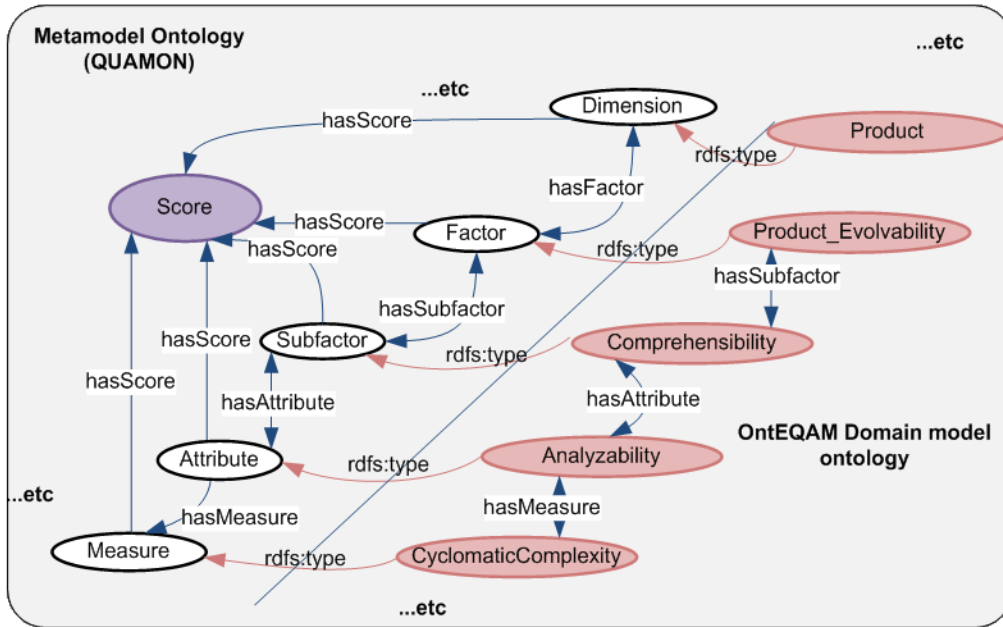
- toolon: <http://aseg.cs.concordia.ca/ontologies/2010/11/toolon/>
- veron: <http://aseg.cs.concordia.ca/ontologies/2010/11/veron/>
- meton: <http://aseg.cs.concordia.ca/ontologies/2010/11/meton/>
- quamon: <http://aseg.cs.concordia.ca/ontologies/2010/11/quamon/>
- rdfs: <http://www.w3.org/2000/01/rdf-schema#>
- rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
- xsd: <http://www.w3.org/2001/XMLSchema#>
- fn: <http://www.w3.org/2005/xpath-functions#>
- afn: <http://jena.hpl.hp.com/ARQ/function#>

As part of the QUAMON ontology, we first define that each software project has an associated quality dimension <doap:Project> <quamon:hasDimension> <quamon:Dimension>. This definition reuses an existing doap<sup>39</sup> ontology that is publically available to share the description of the software project in an ontological format. This is the first step to connect the software project concept with the quality model. QUAMON defines the quality metamodel concepts and relationships, such as <quamon:hasFactor>,<quamon:Factor>,<quamon:hasSubfactor>,<quamon:Subfactor>, <quamon:hasAttribute>,<quamon:Attribute>,<quamon:hasMeasure>,<quamon:Measure>, <quamon:hasScore>,<quamon:Score>, <quamon:inSnapshot><quamon:Snapshot>, etc.

OntEQAM reuses the QUAMON ontology by populating QUAMON with concrete individuals. The following provides an example of instantiating QUMAON.

---

<sup>39</sup> <https://github.com/edumbill/doap/wiki>



**FIGURE 50 ONTEQAM DOMAIN MODEL REUSES QUAMON METAMODEL ONTOLOGY (SUBSET)**

Figure 50 visualizes the reuse approach for the analyzability attribute and one of its measures, i.e., cyclomatic complexity. Steps 1 to 5 are applied here to obtain this result:

1. Define the *product* and *community* dimensions:

```
<quamon:Product> <rdfs:type> <quamon:Dimension> and <quamon:Community>
<rdfs:type> <quamon:Dimension>
```

Figure 50 shows the example of the product dimension.

2. Define *product\_evolvability* quality as a factor that is associated with the *product* dimension.

```
<quamon:Product_Evolvability> <quamon:hasFactor> <quamon:Product> and <quamon:Product_Evolvability>
<rdfs:type> <quamon:Factor>.
```

3. Following the same approach, OntEQAM defines *comprehensibility* as a subfactor of *product\_evolvability* which is associated with the *analyzability* quality attribute.

```
<quamon:Product_Evolvability> <quamon:hasSubfactor><quamon:Comprehensibility> and
<quamon:Comprehensibility> <rdfs:type> <quamon:Subfactor>.
```

```
<quamon:Comprehensibility> <quamon:hasAttribute> <quamon:Analyzability> and <quamon:Analyzability>
<rdfs:type> <quamon:Attribute>.
```

4. To complete the model structure, let's say that OntEQAM defines files with the *cyclomatic complexity* measure to assess the *analyzability* quality attribute.

<quamon:Analyzability> <quamon:hasMeasure> <quamon:CyclomaticComplexity> and <quamon:CyclomaticComplexity> <rdfs:type> <quamon:Measure>, Figure 50.

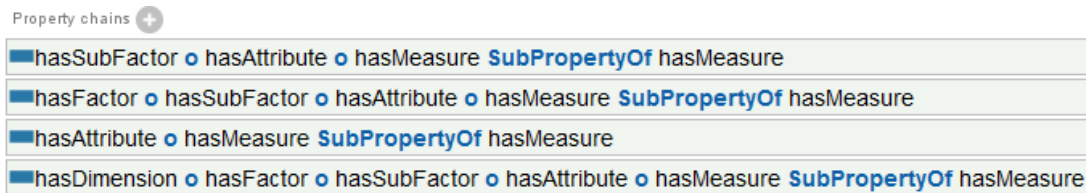
At this stage, from points 1 to 5, queries (such as: What measures are associated with the product\_evolvability dimension? Or what attributes scored poorly under the community dimension?) will not be answered because the only relationships that are asserted are the direct relationships between one level and the next level down (dimension -> factor -> subfactor -> attribute -> measure).

To resolve this issue, OntEQAM reuses QUAMON-added semantics of how the metamodel's relationship chain to each other (e.g., hasDimension relationship with hasSubfactor and hasMeasure). Here is an example of the OWL2 property chain axioms defined to address the inferred metamodel relationships. These axioms represent the dotted (inferred) lines in Figure 49. These inferred relationships are recovered using the Pellet reasoner [149].

- Project-related OWL2 property chain constructs:
  - SubPropertyOf( ObjectPropertyChain( :Project :hasFactor ) :Factor )
  - SubPropertyOf( ObjectPropertyChain( :Project :hasSubfactor ) :Subfactor )
  - SubPropertyOf( ObjectPropertyChain( :Project :hasAttribute ) :Attribute )
  - SubPropertyOf( ObjectPropertyChain( :Project :hasMeasure ) :Measure )
- Dimension-related OWL2 property chain constructs:
  - SubPropertyOf( ObjectPropertyChain( :Dimension :hasSubfactor ) :Subfactor )
  - SubPropertyOf( ObjectPropertyChain( :Dimension :hasAttribute ) :Attribute )
  - SubPropertyOf( ObjectPropertyChain( :Dimension :hasMeasure ) :Measure )
- Factor-related OWL2 property chain constructs:
  - SubPropertyOf( ObjectPropertyChain( :Factor :hasAttribute ) :Attribute )
  - SubPropertyOf( ObjectPropertyChain( :Factor :hasMeasure ) :Measure )
- Subfactor-related OWL2 property chain constructs:
  - SubPropertyOf( ObjectPropertyChain( :Subfactor :hasMeasure ) :Measure )

When the user imports this OWL2 constructs file, the queries mentioned above are answered. The question that could be asked here is: Can users define these inferred relationships without the property chain? While it would be possible to assert all inferred relationships this would not only require a significant amount of changes to the actual model and might become a source of design inconsistencies. For example, in a model that defines 20 measures, the user would need to assert 80 triples

(four relationships for each measure for the measure indirect relationship with projects, dimension, factor, and subfactor) instead of having four constructs only (Figure 51) independent of the number of measures defined in the model. The same issue applies for inferred relationships related to attributes, subfactors, and dimension, which leads to a higher complexity of the resulted ontology and hence a harder to maintain solution. The reason is that whenever a new individual is created in the model (e.g., a new measure is added), all the inferred relationships need to be asserted again by the user.



**FIGURE 51 HASMEASURE-RELATED PROPERTY CHAIN CONSTRUCTS (PROTEGE VIEW)**

Besides recovering semantic links, the Pellet reasoner [149] is also used for consistency checking before the actual reasoning takes place. The consistency checking ensures that there is no type or constraint contradiction being introduced by the user to the QUAMON. For example if the Attribute and Measure concepts are defined as disjoint (i.e. an instance can either be a measure or attribute but not both), then after the reuse, if the user mistakenly asserts that *m1* (individual of Measure) is also an individual of Attribute then the reasoner issues an inconsistency.

The OntEQAM model case study also shows how our SE-EQUAM metamodeling approach supports the evolvability of the quality model by allowing users to instantiate domain model ontology for their specific application context (in this example, this is a quality assessment model for evaluating the evolvability of software systems). The same reusability approach could be applied in any new context or domain.



## 4.2. ONTEQAM ADDRESSES KNOWLEDGE MODELING REQUIREMENT

---

Knowledge modeling refers to the T-Box modeling for knowledge artifacts. Chapter 3 focused on the representation of QUAMON, our metamodel ontology. As part of this case study, to assess software evolvability quality, we will reuse and integrate QUAMON with other ontological software knowledge artifacts needed for OntEQAM. Figure 52 shows an example of such ontological integration of different knowledge artifacts. These artifacts include (but are not limited to) Version Control Repository (VERON),<sup>40</sup> which models activities performed by different source code contributors over time. VERON ontology conceptualizes commits made by contributors to the version control repository, tagged with the date and time. Properties such as commit message and type of modification made (add/delete/change) are also part of VERON. The Issue Tracker Repository (ISSUEON)<sup>41</sup> is designed to model activities related to bug, and change and feature requests. The ISSUEON models concepts such as the reported issue, date and time, assignee, comments made, and the status of the issue. METON<sup>42</sup> is the metadata ontology, and it defines the shared concepts among existing ontologies such as source code files, contributors, and project information. Detailed descriptions of VERON, METON, and ISSUEON ontologies are available here: <https://sites.google.com/site/asegsecold/ontology>.

The ontological representation of the quality model (QUAMON) is used for modeling the assessment model entities (e.g, factors, attributes, measures, scores, and weights). As part of our approach, we also integrate results obtained from third-party source code analysis tools. SE-EQUAM currently integrates the results from the following four source code measurement tools: (1) PMD<sup>43</sup>, which analyzes code for possible bugs and dead and overcomplicated code; (2) Checkstyle<sup>44</sup>, which detects Javadoc style errors. Checkstyle configuration checks violations to defined standards such as: the Java

---

<sup>40</sup> <http://aseg.cs.concordia.ca/ontologies/2010/11/veron.rdf>

<sup>41</sup> <https://sites.google.com/site/asegsecold/ontology/Issueon.owl>

<sup>42</sup> <http://aseg.cs.concordia.ca/ontologies/2010/11/meton.rdf>

<sup>43</sup> <http://pmd.sourceforge.net/>

<sup>44</sup> <http://checkstyle.sourceforge.net/>

Language Specification<sup>45</sup>, the Sun Code Conventions<sup>46</sup>, the Javadoc guidelines<sup>47</sup>, and the JDK API<sup>48</sup>; (3) Simian<sup>49</sup>, which reports duplicated code, i.e., file similarity (similar file names, start, and end line); and (4) JDepend<sup>50</sup>, which reports package level dependencies and instability measures.

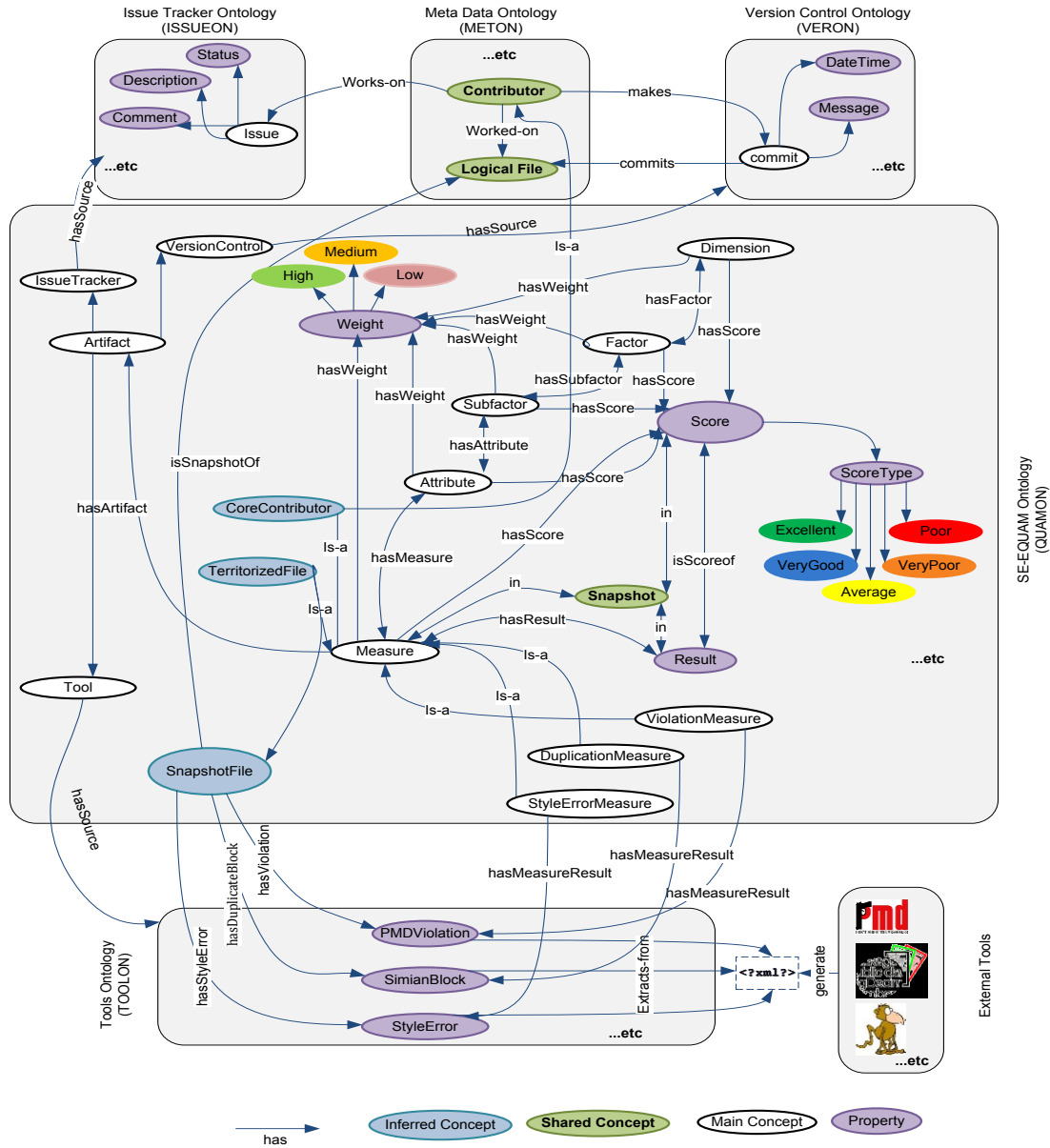


FIGURE 52 SE-EQUAM KNOWLEDGE ARTIFACTS

<sup>45</sup> [http://java.sun.com/docs/books/jls/second edition/html/index.html](http://java.sun.com/docs/books/jls/second%20edition/html/index.html)  
<sup>46</sup> <http://java.sun.com/docs/codeconv/>  
<sup>47</sup> <http://java.sun.com/j2se/javadoc/writingdoccomments/index.html>  
<sup>48</sup> <http://java.sun.com/j2se/docs/api/index.html>  
<sup>49</sup> <http://www.harukizaemon.com/simian/>  
<sup>50</sup> <http://clarkware.com/software/JDepend.html>

We import the tool-specific XML measure reports and parse them to populate our corresponding ontologies. Quality measures and results obtained from external analysis tools are captured by the TOOLON ontology. Below we define each of TOOLON concepts (Table 8) and relationship (Table 9):

**TABLE 8 DEFINITIONS OF TOOLON CONCEPTS**

Concepts	Tool Source	Description
Style Error	Checkstyle	Source code comment style error as defined by standards, e.g., Javadoc.
Style Error Source	Checkstyle	The source name as defined by Checkstyle tool; each style error has a source,
Simian Block	Simian	Block of duplicated lines in a source code file.
Simian Set	Simian	Each set has one or more duplicated blocks within the same file.
Simian Check	Simian	Each simian check defines one or more sets of duplications on a project level.
PMD Violation	PMD	Each source code file has one or more PMD violation.
PMD Rule Set	PMD	Each violation is related to a named PMD rule set, e.g., design rules.
PMD Rule	PMD	Each rule set defines one or more rules, e.g., missing break in switch statement is a design rule violation.
Package	JDepend	Source code package (set of related class files).
Snapshot Package	JDepend	A version of the package concept in a specific snapshot of time.
Cycle	JDepend	A package level cycle indicates two or more packages that depend on each other and so should be changed and released together. A cycle is detected when the package p1 depends on package p2, which in return depends on p1. Dependency here indicates the import of that package path.
Class	JDepend	Each package has one or more classes. This concept is linked to the LogicalFile in METON.
Abstract Class	JDepend	The class that does not provide implementation for all of its methods.
Concrete Class	JDepend	A derived class that provides all method implementations missing in its base class.

**TABLE 9 DEFINITIONS OF TOOLON RELATIONSHIPS**

Relationship	Concepts	Description
hasStyleError	SnapshotFile-StyleError	Each source code file has one or more style (code comments) errors at a certain snapshot of time.
hasStyleErrorSource	StyleError-StyleErrorSource	Each style error has a source (Checkstyle API path) that defines the details of the error.
hasStyleErrorMessage	StyleError-String	This relationship provides a human readable description of the detected style

		error.
hasSet	SimianCheck-SimianSet	Each check of duplication in the source code defines one or more duplication sets. The set includes the count of the duplicated lines.
hasLineCount	SimianSet-Number	The number of duplicated source code lines in a set.
hasDuplicateBlock	SimianSet-SimianBlock	Each set of duplication defines one or more duplication blocks that define the location of duplicated lines within a file.
hasSourceFile	SimianBlock-SnapshotFile	Each block of duplicated source code lines are related to a file version at a snapshot x.
hasStartLineNumber	SimianBlock-Number	The start line number for the duplicated source code block.
hasEndLineNumber	SimianBlock-Number	The end line number for the duplicated source code block.
hasViolation	SnapshotFile-PMDViolation	Each file has one or more code violations (as defined by PMD tool) at a certain snapshot x.
hasViolationDescription	PMDViolation-String	This relationship provides a human readable description of the detected violation.
hasRuleSet	PMDViolation-PMDRuleSet	Each violation is related to a rule set (a category for related violation rules).
hasRule	PMDRuleSet - PMDRule	The rule set defines one or more related violation rules.
hasPackageName	Project-SnapshotPackage	Each software project defines one or more packages at a certain time snapshot x.
hasTotalClasses	SnapshotPackage-Number	Defines the total number of classes under a package at snapshot x.
hasConcreteClasses	SnapshotPackage-Number	Defines the number of concrete classes under a package at time snapshot x.
hasAbstractClasses	SnapshotPackage-Number	Defines the number of abstract classes under a package at time snapshot x.
hasAfferentCoupling	SnapshotPackage-Number	Afferent coupling is detected when packages depend on classes of another package, which increases the package responsibility. This relationship counts the number of packages that depend on package p.
hasEfferentCoupling	SnapshotPackage-Number	Efferent coupling means the number of other packages that package p depends on.
has Abstractness	SnapshotPackage-Number	The ratio of abstract to concrete classes in a package. Ranges from zero to one. Zero indicates the absence of abstract classes, and one means that package only defines abstract classes.
hasInstability	SnapshotPackage-Number	Defines a package's resilience to change based on its coupling. $Instability = efferent\ coupling / (efferent\ coupling + afferent\ coupling)$ . Ranges from zero to one. Zero indicates a completely stable package, and one means a completely

		instable package.
hasClassFile	SnapshotPackage- Class	Defines classes/files under each package at a time snapshot x.
dependsOn	SnapshotPackage- SnapshotPackage	Defines the dependency from package p on other packages at a time snapshot x. Dependency occurs when package p uses classes from other packages.
usedBy	SnapshotPackage- SnapshotPackage	The inverse of dependsOn.
hasCycleWith	SnapshotPackage- SnapshotPackage	Defines the cycle dependency between packages. Refer to the definition of Cycle concept in the above table.

The following is an example of the shared concepts among our sub-ontologies. LogicalFile is a concept defined in METON (Figure 52) [94]. The concept can be instantiated in several domain-specific ontologies. For example, an individual is added to the VERON ontology to represent a file committed by a contributor. A second individual is part of QUAMON when the measure that the user uses assesses logical files for style error violations. Given our reproducible URI generation scheme, individuals that are actually referring to the same LogicalFile can be identified as such in both ontologies. Having this reproducible URI generation also allows for the independent population of the sub-ontologies while supporting shared concept linking. SE-EQUAM also supports the generation of additional traceability links through SPARQL queries, inference rules, and DL axioms (e.g., property chain) to further enrich and link the sub-ontologies.

**Linking using SPARQL queries:** In the following example, a traceability link is established among quality assessments that take place at different time intervals. The snapshot concept is used to uniquely define these different time intervals. One potentially interesting traceability link is created by establishing a relationship between the Commit concept in the VERON ontology and the Snapshot concept in the QUAMON ontology by using the commit date property (Table 10).

**Linking through Inference rules:** This is another traceability link that can be established to connect the active Contributor concept in the METON ontology with a certain Snapshot by using an inference rules (Table 10).

TABLE 10 ESTABLISHING TRACEABILITY LINKS USING SPARQL AND INFERENCE RULES

Linking using SPARQL queries	Linking through Inference rules
<pre>SELECT DISTINCT ?snapshot ?res WHERE{ ?snapshot rdf:type quamon:Snapshot. ?snapshot quamon:hasSnapshotEndDate ?snd. ?snapshot quamon:hasPreviousSnapshot ?psn. ?psn quamon:hasSnapshotEndDate ?psnd. ?res rdf:type veron:Commit. ?res meton:hasDateTime ?d. Filter (?d &gt; ?psnd &amp;&amp; ?d &lt;= ?snd) }</pre>	<pre>[contPerSnapshot: (?cmt meton:hasContributor ?cont) (?cmt quamon:inSnapshot ?snapshot) --&gt; (?cont quamon:inSnapshot snapshot) ]</pre>

### 4.3. ONTEQAM ADDRESSES THE KNOWLEDGE POPULATION REQUIREMENT

Here, the knowledge population requirement refers to the A-Box ontological model population, i.e., the assertion of concrete individuals to the T-Box model.

The first step is to extract the knowledge from existing knowledge artifacts. In our current implementation, there are four major data sources supported by our assessment process: (1) version control repositories, e.g., CVS<sup>51</sup>, SVN<sup>52</sup>, and Git<sup>53</sup>; (2) issue tracking repositories, e.g., Jira<sup>54</sup>, and Bugzilla<sup>55</sup>; (3) source code, e.g. Java; and (4) third-party (external), static code analysis tools, e.g., PMD [163], CheckStyle [164], Simian [170], and JDepend [171], described in the previous section.

Figure 53 shows a *sample* of the populated ontologies (A-Box); it extends the T-Box modeling (Figure 52) with concrete individuals.

In order to obtain the populated knowledge in Figure 53, the following steps are followed:

<sup>51</sup> <http://cvs.nongnu.org/>

<sup>52</sup> <http://subversion.tigris.org/>

<sup>53</sup> <http://git-scm.com/>

<sup>54</sup> <https://www.atlassian.com/software/jira>

<sup>55</sup> <http://www.bugzilla.org/>

- Project setup: In this step, the user provides the project name (e.g., Creativecomputing). This will be used to populate the project name and then connect that project with the quality dimension level of the assessment model.
- Artifact setup:
  - Version Control: The user provides the version control URL from which the source code can be extracted (e.g., <http://creativecomputing.googlecode.com/svn/trunk/>). As part of the Ambient Software Evolution Group<sup>56</sup>, we support the extraction of subversion e.g., CVS<sup>57</sup>, SVN<sup>58</sup>, and Git<sup>59</sup>; versioning systems [120].
  - Issue Tracker: The user provides the issue tracker URL where the project's issues are populated. As part of the Ambient Software Evolution Group<sup>56</sup>, we support the extraction of issues from GoogleCode<sup>60</sup>, Jira<sup>61</sup>, and SourceForge<sup>62</sup>.
- Snapshot setup: The user defines the type of snapshot required to perform the evolvability assessment. A snapshot is defined by a period length and a type (e.g., 6 months, which indicates that the user would like to obtain an evolvability score for the specified project every 6 months). By default, the analysis will cover the date from the first available commit the most recent commit made (based on the assessment date). Our implementation supports snapshots of type day, week, month, and year.

---

<sup>56</sup> <https://sites.google.com/site/juergenrilling/>

<sup>57</sup> <http://cvs.nongnu.org/>

<sup>58</sup> <http://subversion.tigris.org/>

<sup>59</sup> <http://git-scm.com/>

<sup>60</sup> <http://code.google.com/>

<sup>61</sup> <http://www.atlassian.com/software/jira>

<sup>62</sup> <http://sourceforge.net/>

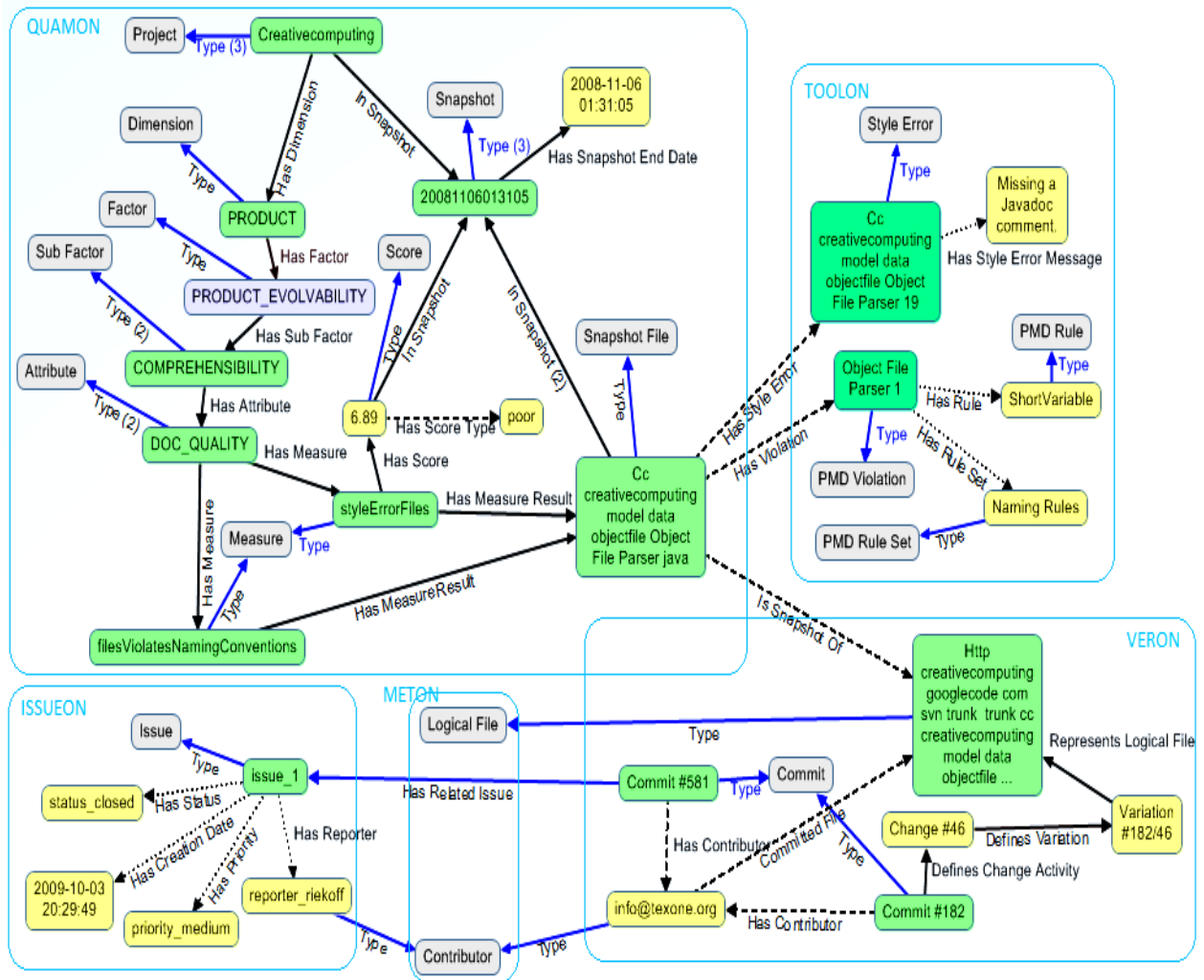


FIGURE 53 ONTOLOGY POPULATION (ONTOLOGICAL KNOWLEDGE MODEL)

- Source code setup: This step is sub-divided into three steps:
  - Location setup: The user needs to define a location on the file system where the application could checkout the source code. This step can be used only one time unless the user wants to change it.
  - Source code checkout: The source code is checked out to the previously defined location. Because our implementation considers space limitations, we only extract one snapshot at any point of time to perform the required analysis, after which we delete the checked out code from the file system before proceeding to the next snap-



shot. Note that each checkout is tagged with the snapshot end date (e.g., 20091004120322, where it represents 2009-Oct-04 at 12:03:22).

- Tools' report generation: We run external static analysis tools over the checked out source code such as PMD [163], Checkstyle[164], Jdepend[171], and Simian [170] and generate reports for source code violation, style errors, package dependency, and code duplication, respectively. These reports are in XML format. Note that each XML report is tagged with the project name, the tool that generated it, and the snapshot for which it was generated (e.g., Creativecomputing\_pmd\_20091004120322.xml).
- Populate the input artifact ontologies: In this step, the population process starts by creating the project ontology that defines the project name reusing DOAP <<http://usefulinc.com/ns/doap#name>>. Next, we populate the VERON ontology, which is used to define the start and end dates of the project lifetime. Based on these dates, the snapshots are calculated. The ISSUEON ontology is then populated with issues and their information. Then, we correlate each commit in VERON and each issue in ISSUEON with the snapshot based on the commit date and the issue creation date, respectively. For example, assume a snapshot 20091004120322, in which creative computing project has a commit number 33 and an issue ID 3 is reported. Here are the populated triples:
  - <<http://aseg.cs.concordia.ca/secold/resource/snapshot/creativecomputing/20091004120322>><rdfs:type> <quamon:Snapshot> to define a snapshot for Creative Computing project with end date 2009-10-04 at 12:03:22.
  - <<http://aseg.cs.concordia.ca/secold/resource/commit/creativecomputing/33>><rdfs:type> <veron:Commit> to define commit ID 33 for Creative Computing project.
  - <<http://aseg.cs.concordia.ca/secold/resource/commit/creativecomputing/33>><quamon:inSnapshot><<http://aseg.cs.concordia.ca/secold/resource/snapshot/creativecomputing/20091004120322>> to assign the snapshot in which the commit 33 is made.

- `<http://aseg.cs.concordia.ca/secold/resource/issue/creativecomputing/3><rdfs:type>  
<issueon:Issue>` to define a reported issue with ID 3 under Creative Computing project.
- `<http://aseg.cs.concordia.ca/secold/resource/issue/creativecomputing/3><quamon:i  
nSnapshot><http://aseg.cs.concordia.ca/secold/resource/snapshot/creativecomputing/200  
91004120322 >` to assign the snapshot during which the issue ID 3 is reported.

METON is populated with shared data where, for example, a contributor could be the commit author and the issue assignee. For example, the commit number 33 is submitted by John Snow, and John Snow has also been assigned issue ID 3 (the added value of this approach is the reusability of the URI generation schema described in the background chapter, as part of QUAMON will automatically link the contributor concept in the METON and ISSUEON ontologies):

- `<http://aseg.cs.concordia.ca/secold/resource/contributor/creativecomputing/johnsnow><rdfs:type><meton:Contributor>` to define a contributor named John Snow under creative computing project.
- `<http://aseg.cs.concordia.ca/secold/resource/commit/creativecomputing/33><meton:hasContributor><http://aseg.cs.concordia.ca/secold/resource/contributor/creativecomputing3/johnsnow>` to link commit ID 33 to the contributor John Snow.
- `http://aseg.cs.concordia.ca/secold/resource/issue/creativecomputing/3<issueon:hasAssignee><http://aseg.cs.concordia.ca/secold/resource/contributor/creativecomputing3/johnsnow>` to link issue ID 3 to the contributor John Snow.

Next, we populate the TOOLON ontology; the input for this ontology is the set of the XML-generated reports from the external analysis tools described in the previous step. For example, the PMD tool, as part of the reported generated for 20091004120322 snapshot, indicated that the file *f1* has violated the cyclomatic complexity rule. We parse this information from the XML to triple format:

- `<http://aseg.cs.concordia.ca/secold/resource/SnapshotFile/f1><toolon:hasViolation>  
<toolon:CyclomaticComplexityViolation>`.

The file *f1* here is a snapshot file (reported in snapshot 20091004120322). One important step needed here is the linking of this snapshot file with the logical file defined in the METON ontology. To do this, we run a SPARQL query that matches the address of the logical file with the snapshot file address; if they match then they are linked with `isSnapshotOf` relationship.

- `<http://aseg.cs.concordia.ca/secold/resource/SnapshotFile/f1><quamon:isSnapshotOf>  
<http://aseg.cs.concordia.ca/secold/resource/LogicalFile/f1>`.

The same process we described here for the PMD tool is applied for all the external tool reports. Each XML report (e.g., `Creativecomputing_pmd_20091004120322.xml`) has a corresponding sub-ontology under TOOLON called `Creativecomputing_pmd_20091004120322.ntriples`.

- Import QUAMON: As part of this step, we import the QUAMON ontology that defines a metamodel of quality concepts (dimensions, measures, etc.) and their relationships (e.g., `hasAttribute`, `hasMeasure`, etc.) into the populated knowledge base of version control, issue tracker, and quality model information populated in the previous steps.
- Populate OntEQAM: OntEQAM defines individuals of each of the SE-EQAM items, for example, `<http://aseg.cs.concordia.ca/secold/resource/attribute/creativecomputing/Analyzability><rdfs:type><quamon:Attribute>`. `<http://aseg.cs.concordia.ca/secold/resource/measure/creativecomputing/CommitsOverTime><rdfs:type><quamon:Measure>`. Meta information is also populated such as definition, score, and type. Details about how OntEQAM calculates the score are provided in the next chapter.
- Reason for inferred knowledge: This step involves importing a pre-defined ontology that we created with the added semantics used by QUAMON to infer indirect relationships between the different model levels using OWL2 property chain. (Details are provided in section 4.1.)

We use the Pellet reasoner [149] to infer new relationships, and we will show how it is used as part of the knowledge exploration to answer advanced queries in the next section.

#### 4.4. ONTEQAM ADDRESSES THE KNOWLEDGE EXPLORATION REQUIREMENT

---

In what follows we provide examples of how the resulting OntEQAM evolvability quality assessment model not only takes advantage of model reuse but also benefits from the semantic richer model during knowledge exploration. While some of the existing models do provide their assessment results on a web interface (e.g., SQO-OSS [42]) or as database dumps (e.g., QUALOSS via FLOSSMetrics [40]), they only provide support for simple queries. In contrast, our approach allows for the use of ontological reasoners to support semantically richer queries that include inferring implicit knowledge such as indirect relationships.

Using traditional querying techniques, a query for retrieving “assessment results” will only return the calculated measures values and their assessment scores. For the same problem, using a SPARQL query that takes advantage of our semantically rich representation will return an individual score breakdown across the complete model (e.g., dimension, factors, and subfactors).

The semantic modeling approach also forms the basis for user-defined structural queries to explore model structure. In fact, given our unified ontological representation, cross-ontological exploration can also be formulated.

In order to improve the readability, we omit the fully qualified names when referencing the ontologies. The following are examples of actual queries that can be supported natively by our semantic modeling approach.

**Structural queries:** (Q1) *Identify all measure(s) that are used to assess the score of the developer community size attribute?* This is a basic structure-related measure that uses the directly asserted relationship between the attributes and measures of the quality model (Q1 in Figure 54).

```
SELECT ?measure WHERE {?measure rdf:type quamon:Measure. ?attribute rdf:type quamon:Attribute.
?attribute quamon:hasName ?name. FILTER regex(?name, "Developer Community Size", "i").}
```

**Advanced structural queries:** (Q2) *Return the quality measures grouped by the domain model's dimension?* This query returns the breakdown of the dimension score at its lowest levels, with measures being the actual assessments of artifacts. The values of the measures are summed up in a certain process (assessment process details are provided in Chapter 5) in order to find the attributes scores, which, in turn, are used to find the sub-factors scores that contribute to finding the factors scores, and finally the factors are used to find the dimensions scores. In this query, we are skipping the whole model hierarchy and trying to find the indirect relationship between measures and dimensions. To answer this query, we need a relationship that connects measures to dimensions. This relationship is not explicitly asserted within the OntEQAM quality model but is inferred from the OWL property chain after it is integrating with SE-EQUAM (Q2 in Figure 54).

```
SELECT ?dimension ?measure WHERE {?dimension rdf:type quamon:Dimension. ?dimension quamon:hasMeasure ?measure. } order by ?Dimension
```

**Assessment queries:** (Q3) *What are the files that violated the source code cloning rules (applies to code that does not call Clonable interface, calls super.clone(), and throws the CloneNotSupportedException in this case)?* This is an interesting query for users assessing the quality of a software system based on the proper usage of clones. The following query returns a project quality in terms of clones detected. Files are extracted from METON, while violations are part of TOOLON.

```
SELECT ?file WHERE { ?file rdf:type meton:LogicalFile. ?file toolon:hasViolation ?violation. ?violation toolon:hasRuleSet ?ruleset. ?ruleset rdfs:label ?lbl. FILTER regex(?lbl, "Clone Rule Violation", "i"). }
```

(Q4) *What are the measures that scored excellent?* This helps the user define what aspects of the assessed project are strong and improve the overall quality score (marked as Q4 in Figure 54).

```
SELECT ?measure WHERE { ?measure rdf:type quamon:Measure. ?measure quamon:hasScore ?score. ?score quamon:hasScoreType "excellent". }
```

(Q5) *What are the commits that the contributor committed to the version control system with empty message?* This query is for users interested in the quality of the documentation/comments of the versioning system. This query uses VERON to get the commits where the commit message is a property of the commit.

```
SELECT ?commit WHERE { ?commit rdf:type VERON:Commit. OPTIONAL { ?commit meton:hasMessage ?msg}. FILTER (!bound(?msg))}
```

(Q6) *What is the assessment score of Stability quality attribute in snapshot Jan 24, 2010?* This is a specific query that provides the user with the information about the score of any quality model item at any point of time; this query could be used in many contexts where a certain time frame is of interest, for example, at the time that a core member has left or another competitive project made a new release. We can then check how that situation affected the evolvability quality of our assessed project. This query only considers two concepts from QUAMON: Attribute and Snapshot.

```
SELECT ?attribute WHERE { ?attribute rdf:type quamon:Attribute. ?attribute quamon:inSnapshot ?snapshot. ?snapshot quamon:hasSnapshotDate "01-24-2010". ?attribute rdfs:label ?lbl. FILTER regex(?lbl, "Stability", "i") }.
```

**Advanced assessment queries:** (Q7) *Identify all projects where “changeability” attributes scored “Average.”* This query is of particular interest to users who are assessing multiple projects for comparison or benchmarking purposes. In order to be able to answer this query, our approach can take advantage of the OWL2 property chain construct to infer implicit knowledge between the project and the changeability quality attribute (Q7 in Figure 54).

```
SELECT DISTINCT ?project WHERE { ?project rdf:type doap:Project. ?project quamon:hasDimension ?d. ?d quamon:hasAttribute ?attribute. ?attribute quamon:hasScore ?score. ?score quamon:hasScoreType "average". ?attribute rdfs:label ?lbl. FILTER regex (?lbl, "CHANGEABILITY", "i") }
```

(Q8) *Considering the “Excellent” score type as a positive impact on the assessment, what are the attributes that positively affect evolvability quality factor at any point of time?* This query is interesting in regards to defining the strong points of a certain item within the model (here it is evolvability). In cases when the evolvability score is in the upper scale (very good or excellent), the query returns all excellent quality score across the complete structure. This relationship is called hasAttribute and it connects attributes to factors; it is inferred using the property chain construct (Q8 in Figure 54).

```

SELECT ?factor ?attribute WHERE {
  ?factor rdf:type quamon:factor.
  ?factor quamon:hasAttribute ?attribute.
  ?attribute quamon:hasScore ?score.
  ?score quamon:hasScoreType "excellent".}

```

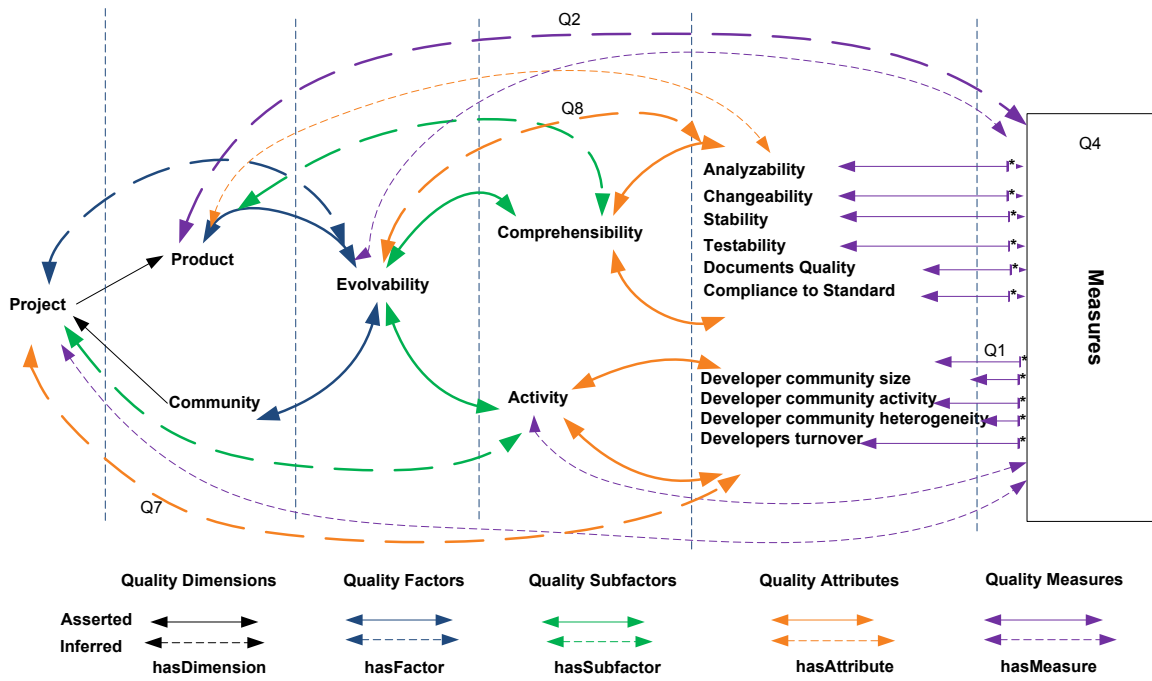


FIGURE 54 SEMANTICALLY RICHER KNOWLEDGE EXPLORATION

## CHAPTER 5: THE FUZZY LOGIC BASED ASSESSMENT PROCESS

A process is defined as “the set of interrelated or interacting activities which transforms input into output” [38]. The quality assessment process is based on a sequence of steps, with the input being a project with a set of knowledge artifacts (e.g., versioning system, issue tracker, source code) and its output typically being a single, overall quality score. A key challenge is how to derive the actual score and how to deal with various uncertainties that might affect the scoring. While some of the existing quality models make the calculation of the quality scores transparent to the end-user (e.g., QUALOSS [16, 40] and QSOS [55]), other models provide no details of how their assessment scores are derived (e.g., ISO/IEC 9126 [15]).

In this thesis, we introduce a fuzzified quality assessment process that is capable of dealing with uncertainties [78, 83, 85, 172]. There are two main sources during the assessment process where uncertainties can arise: (1) Supporting a quality assessment in such a global context requires the use of widely distributed, heterogeneous knowledge resources at different levels of representations and the modeling and integration of semantics. The required knowledge integration creates uncertainties about the availability, accuracy, and completeness of the available knowledge during the actual assessment; (2) The boundaries of the assessment scores.

For example, the SIG maintainability model (SMM) [64] crisp measures' scores based on benchmarking scales which are determined by expert opinion.

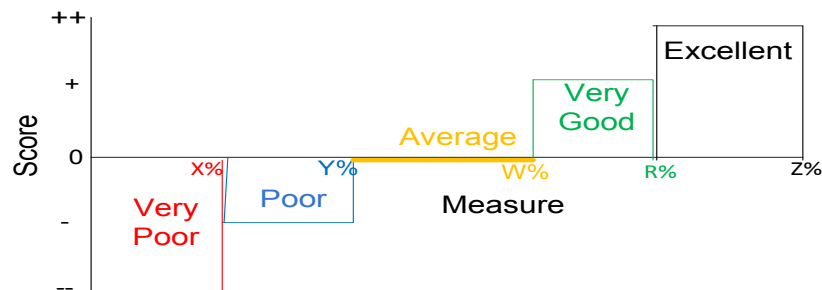

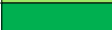





FIGURE 55 SIG MAINTAINABILITY MODEL BENCHMARKING APPROACH



Another example is Figure 56 which shows the SQALE model scale that defines crisp numerical boundaries around each of the colors (score).

<b>From</b>	<b>To</b>	<b>Rating</b>	<b>Color</b>
<	<b>0.9</b>	<b>A</b>	
<b>0.9</b>	<b>3</b>	<b>B</b>	
<b>3</b>	<b>9</b>	<b>C</b>	
<b>9</b>	<b>30</b>	<b>D</b>	
<b>30</b>	>	<b>E</b>	

**FIGURE 56 SQALE SCALE [70]**

Crisp boundaries indicate that the information in a knowledge source is precise and complete. However, when assessing software ecosystems, knowledge resources might no longer reflect precise or complete information and therefore the assessment values might no longer be crisp and actually be between two successive scores. As a result, crisp scores: (1) cannot accurately reflect uncertainties caused by scores around a measure's score boundaries (e.g., a measure value of x% or less is considered as very poor, whereas as a score of x% + 0.1 is ranked to be only poor); (2) lack support for considering the membership degree of a score. For example, using the crisp scale, two measures, 0% and x%, would be both rated as very poor. While in reality the x% is closer to poor (weak membership to the very poor scale) while the 0 strongly belongs to the very poor scale; (3) reflect a lack of automation and a reliance on experts to define the benchmarks and crisp values for quality score. This is both subjective and creates additional uncertainty.

Fuzzy logic has been widely used as a potential approach to deal with these types of uncertainties. If we fuzzify the scale of colors used in Figure 56, the boundaries between any two colors will not change from one range to the other but rather will provide gradual shade of the next scale color. For example, if the user obtains a score of 30.5 then this would be in the overlapping areas (orange shade). The intensity of the color shade indicates the membership of the score (e.g., the darker the orange the closer it is to the red scale). This fuzzification is also fully automated using existing fuzzy inference systems such as JFuzzyLogic [173][88]. In addition to the fuzzification, uncertainties can

also be dealt with by adjusting scoring weights assigned to various quality concepts. Quality concepts with higher uncertainties (e.g., less reliability in the metrics) can be assigned lower weights.

The focus of our fuzzy-based assessment process is on how these uncertainties related to the assessment of the evolvability quality of software systems can be addressed. SE-EQUAM is optionally complemented with an underlying assessment process that addresses the fuzziness due to uncertainties in the extracted knowledge and provides evolvability scores at different abstraction levels.

At the end of the assessment, we populate our existing ontology knowledge model with the assessment results (evolvability score per each model item, e.g., measure score, attribute score, and dimension score) and score value per snapshot. The scores are available in both a fuzzy format (e.g., 40% average and 60% very good) and as a de-fuzzified (numerical) score (e.g., 5 out of 9). Populating the assessment results back to our ontology model provides the users with more information to further analyze and compare queries.

## 5.1. ASSESSMENT PROCESS

---

The input to the process is defined by specifying the project(s) for which the evolvability quality is to be assessed and the projects knowledge artifact(s) to be used during the assessment. The output is the quality assessment score. Figure 57 shows the set of assessment steps. Details of each of the steps are provided below:

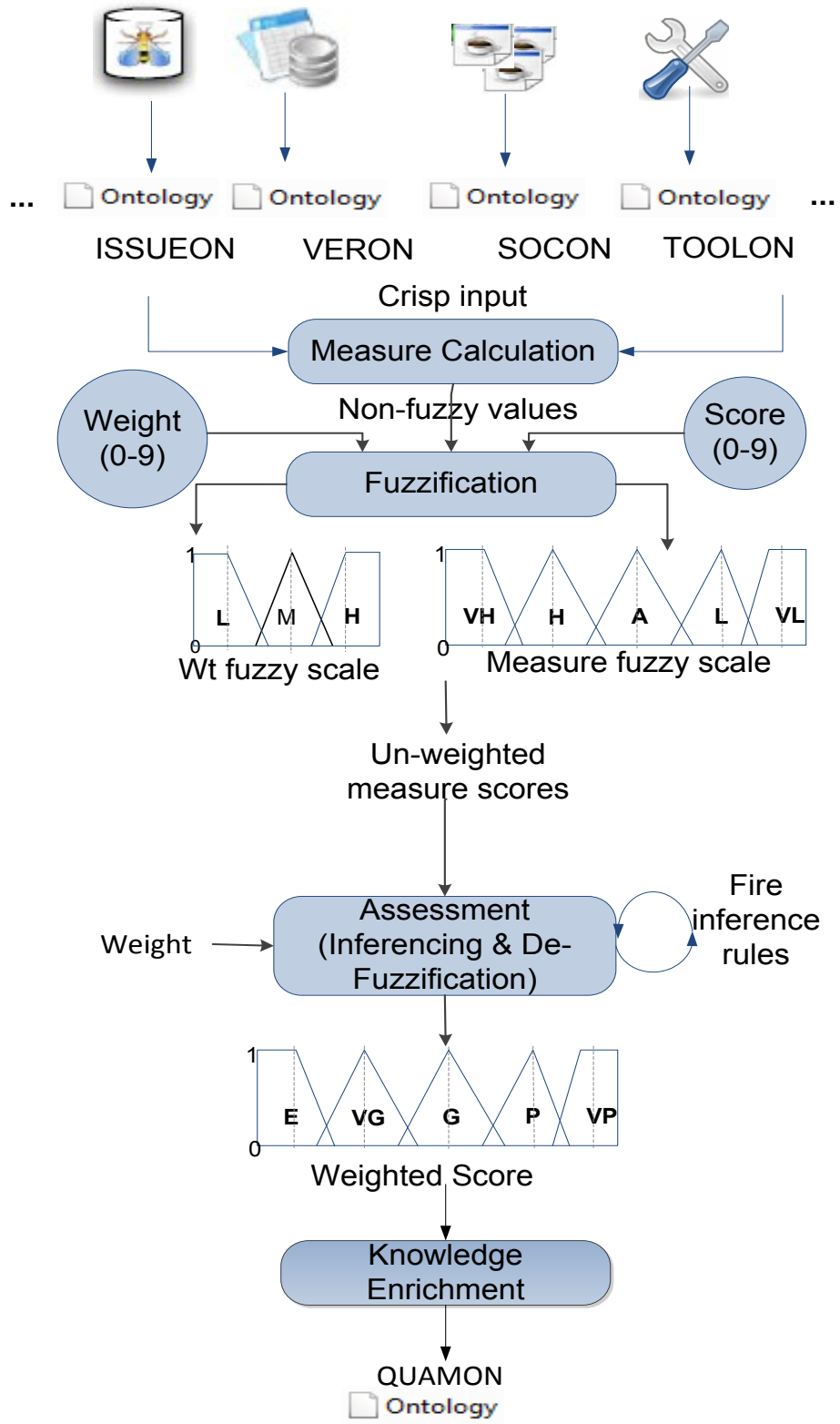


FIGURE 57 FUZZY ASSESSMENT PROCESS STEPS

### 5.1.1. MEASURE CALCULATION

As described in the OntEQAM knowledge population section in Chapter 4, which detailed knowledge extraction and population, OntEQAM assigns measures to attributes based on existing quality assessment models such as SMM [64], QUALOSS [40], and ISO/IEC 9126 [15]. Table 11 provides a more detailed view of the measures assignments.

**TABLE 11 ONTEQAM MEASURES ASSIGNMENTS (SUBSET)**

Subfactor	Attribute	Measure	Tool
Comprehensibility	analyzability	Evolution in commits size. The size represents the number of files committed in one activity on the version control system.	Ours
		Evolution in Cyclomatic Complexity, number of decision points in a method (1-4 denotes low, 8-10 denotes high and >=11 denotes very high complexity). Decision points could be if, while, for, and switch cases.	PMD
		Evolution in the number of commits made on the version control system.	Ours
		Evolution in activity on source code files. Activities represent modifications, additions and deletions made via commits on the version control system on a specific file.	Ours
		Evolution in code size (volume) violations in terms for npath complexity, number of methods/fields in a class, and length of methods or fields.	PMD
		Evolution in the number of non-commented source code method, variable, and/or constructor.	PMD
		Evolution in design rules violations <sup>63</sup> such as testing null with .equals, uncommented empty method, and god classes [174].	PMD
		Basic rules violations <sup>64</sup> such as override both equals and hash code, return from finally block, and unconditional if statement.	PMD
	Changeability	Evolution in the number of files that contains duplicated lines of code. The default threshold for code duplication is 6 lines of code. It is case insensitive for any string or character, ignores modifiers such as public and private.	Simain
		Evolution in the number of the overall duplicated lines.	Simian
		Evolution in code size (volume) violations in terms for npath complexity, number of methods/fields in a class, and length of methods or fields.	PMD
		Evolution in cyclomatic complexity, number of decision points in a method (1-4 denotes low, 8-10 denotes high and >=11 denotes very high complexity). Decision points could	PMD

<sup>63</sup> <http://pmd.sourceforge.net/pmd-5.0.4/rules/index.html#Design>

<sup>64</sup> <http://pmd.sourceforge.net/pmd-5.0.4/rules/index.html#Basic>

		be if, while, for, and switch cases.	
		Evolution in clone's violation. This include the proper implementation of clone() using super.clone(), throwing the CloneNotSupportedException and implementing clonable interface.	PMD
		Import statements violations. This includes duplicated imports, importing Java.lang, importing from the same package, un-used imports and un-necessary use of fully qualified names.	PMD
		Evolution in design rules violations <sup>65</sup> such as testing null with .equals, uncommented empty method, and god classes [174].	PMD
		Basic rules violations <sup>66</sup> such as override both equals and hash code, return from finally block, and unconditional if statement.	PMD
		Evolution in coupling between objects including attributes, local variables and return types. It also assesses the application of Law of Demeter [175, 176].	PMD
		Evolution in loose coupling; referencing objects with their interfaces rather than their concrete implementation.	PMD
	<b>Stability</b>	Evolution in the number of files that contains duplicated lines of code. The default threshold for code duplication is 6 lines of code. It is case insensitive for any string or character, ignores modifiers such as public and private.	Simain
		Evolution in the number of the overall duplicated lines.	Simian
		Evolution in cyclomatic complexity, number of decision points in a method (1-4 denotes low, 8-10 denotes high and >=11 denotes very high complexity). Decision points could be if, while, for, and switch cases.	PMD
		Evolution in code size (volume) violations in terms for npath complexity, number of methods/fields in a class, and length of methods or fields.	PMD
		Evolution in exceptions handling violations such as catching throwable, catching NullPointerExceptions, re-throwing exception, throwing exception in finally block, throwing instances of the same exception, and catching generic exceptions.	PMD
		Evolution in excessive imports, high number of import statements indicates high coupling, which should be avoided.	PMD
		Evolution in coupling between objects including attributes, local variables and return types. It also assesses the application of Law of Demeter [175, 176].	PMD
		Evolution in design rules violations <sup>67</sup> such as testing null with .equals, uncommented empty method, and god classes [174].	PMD
		Evolution in loose coupling; referencing objects with their	PMD

<sup>65</sup> <http://pmd.sourceforge.net/pmd-5.0.4/rules/index.html#Design>

<sup>66</sup> <http://pmd.sourceforge.net/pmd-5.0.4/rules/index.html#Basic>

<sup>67</sup> <http://pmd.sourceforge.net/pmd-5.0.4/rules/index.html#Design>

		interfaces rather than their concrete implementation.	
		Basic rules violations <sup>68</sup> such as override both equals and hash code, return from finally block, and unconditional if statement.	PMD
		Evolution in clone's violation. This include the proper implementation of clone() using super.clone(), throwing the CloneNotSupportedException and implementing clonable interface.	PMD
		Import statements violations. This includes duplicated imports, importing java.lang, importing from the same package, un-used imports and un-necessary use of fully qualified names.	PMD
	<b>Testability</b>	Evolution in junit tests quality violations. This includes measuring the signature of junit suite() method which should be public and static, inclusion of information message in assertions, each test case should include at least one assertion, the proper use of assertEquals versus assertTrue, assertNull and assertEquals.	PMD
		Evolution in cyclomatic complexity, number of decision points in a method (1-4 denotes low, 8-10 denotes high and >=11 denotes very high complexity). Decision points could be if, while, for, and switch cases.	PMD
		Evolution in exceptions handling violations such as catching throwable, catching NullPointerExceptions, re-throwing exception, throwing exception in finally block, throwing instances of the same exception, and catching generic exceptions.	PMD
		Basic rules violations <sup>69</sup> such as override both equals and hash code, return from finally block, and unconditional if statement.	PMD
	<b>Compliance to standard</b>	Evolution in Javadoc style errors. Checks include empty blocks, the use of Javadoc style comments, and the proper naming conventions <sup>70</sup> .	CheckStyle
		Evolution in number of files with Javadoc errors as defined in the previous measure.	CheckStyle
		Evolution in un-used code such as private fields, local variables, and private methods, which are defined but not used.	PMD
	<b>Doc quality</b>	Evolution in naming convention violations <sup>70</sup> .	PMD
		Evolution in Javadoc style errors. Checks include empty blocks, the use of Javadoc style comments, and the proper naming conventions <sup>70</sup> .	CheckStyle
		Evolution in the number of non-commented source code method, filed, and/or constructor.	PMD
		Evolution in number of files with Javadoc errors. Checks include empty blocks, the use of Javadoc style comments, and the proper naming conventions <sup>70</sup> .	CheckStyle
		Evolution in commits made on the version control system without a commit message to explain the change made.	Ours

<sup>68</sup> <http://pmd.sourceforge.net/pmd-5.0.4/rules/index.html#Basic>

<sup>69</sup> <http://pmd.sourceforge.net/pmd-5.0.4/rules/index.html#Basic>

<sup>70</sup> <http://docs.oracle.com/javase/specs/#40169>

Community	Activity	Development community size	Evolution in contributing community size. This is the number of contributors who made any type of change to the software based on the commit history in the version control system.	Ours
			Evolution in core contributors' size. A core member is a contributor who made more than the average number of commits to the version control system.	Ours
			Evolution in the number of commits made to the version control system.	Ours
			Evolution in number of files modified by a single contributor based on the version control system.	Ours
			Evolution in the number of commits made to the version control system.	Ours
	Development community activity	Evolution in number of files modified by a single contributor based on the version control system.	Ours	
		Evolution in the number of commits made by a single contributor.	Ours	
		Evolution in activities per file. Activities include change, add, and/or delete actions made to a file in the version control system.	Ours	
		Evolution in the number of files modified in a single commit. This represents the volume of a single change or activity made on the version control system.	Ours	

Measures' results are calculated by querying our ontological knowledge model using SPARQL (e.g., Table 12). The returned results are further processed to subsequently define the measure scale.

As we can see from Table 12, each query returns two values: the measure result and the snapshot with which it is associated. At the end, we count the number of returned results per snapshot to get a numerical representation of the measure that we would use to measure benchmarking (scale definition will be detailed in the next step).

TABLE 12 SAMPLE SPARQL QUERIES FOR MEASURES CALCULATION

Measure	SPARQL Query
Evolution in number of new contributors	SELECT distinct ?snapshot ?res WHERE { ?res rdf:type meton:Contributor. ?cmt meton:hasContributor ?res. ?cmt quamon:inSnapshot ?snapshot. ?snapshot quamon:hasSnapshotEndDate ?ss1. optional {?res quamon:inSnapshot ?s2. ?s2 quamon:hasSnapshotEndDate ?ss2. filter (?ss1 > ?ss2)}. filter (!bound(?s2))} order by ?snapshot
Evolution in files violate Naming Rule (PMD source)	SELECT distinct ?snapshot ?res WHERE { ?res rdf:type quamon:SnapshotFile. ?snapshot rdf:type quamon:Snapshot. ?res quamon:inSnapshot ?snapshot. ?res toolon:hasViolation ?v. ?v toolon:hasRule ?rule. ?rule rdfs:label ?lbl. FILTER regex(?lbl, "Naming Rule", "i") }
Evolution in	SELECT distinct ?snapshot ?res WHERE { ?res rdf:type

number of commits with empty message	veron:Commit. ?snapshot rdf:type quamon:Snapshot. ?res quamon:inSnapshot ?snapshot. OPTIONAL {?res meton:hasMessage ?msg}. FILTER (!bound(?msg))}
--------------------------------------	---

### 5.1.2. FUZZIFICATION

In this step, we explain how we create a fuzzy assessment scale for the assessment process input (measure and weight) and for the process output (score).

**Create measure fuzzy scale:** In most existing assessment models, a domain expert is required to evaluate and assess a measure value to establish a final score, e.g., [64]. In contrast, the objective of our approach is to reduce the dependency on domain experts and to provide a more uniform and objective assessment. In order to achieve this goal, one has to be able to deal with data uncertainties (e.g., data missing due to an incomplete project history). We apply a fuzzification approach that takes measured values as an input and distributes the values to form a fuzzy scale. The inputs for this step are (non-fuzzy) measure values obtained from earlier (e.g. Table 12) parts of our process and the output is measure fuzzy scales.

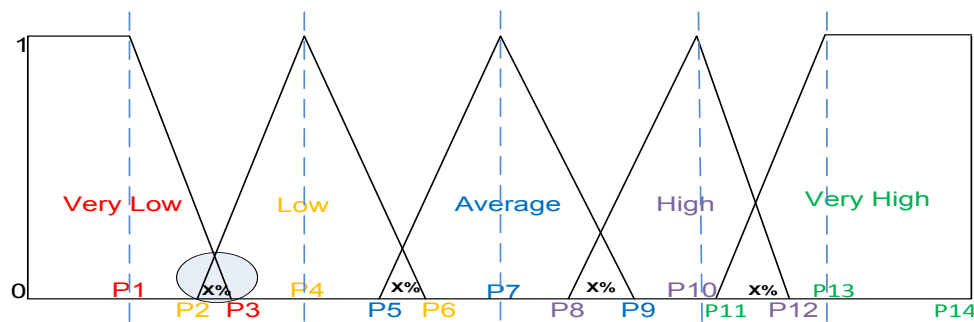


FIGURE 58 MEASURE FUZZY SCALE

Figure 58 shows such a fuzzy measure scale, where the x-axis represents the scores range and the y-axis represents the membership degree of the measure value to the fuzzy scale (range is 0-1). The higher the membership value, the stronger the value relations to its score scale. The x-axis points are distributed mathematically by using the  $P1$  as a minimum measure value (worst measure value over time),  $P13$  as a maximum value (best measure value over time),  $P7$  as an average value, and the mid-



points between each. The last point,  $P_{14}$ , is the maximum plus the standard deviation. Fuzzy terms used for measures are {*very low (VL)*, *low (L)*, *average (A)*, *high (H)*, and *very high (VH)*}.

We analyze the measure results over a user-defined time interval  $ti$ , with  $ti$  being, for example, a 6-month interval. As a result,  $ti$  is calculated as a function of project age divided by  $ti$ , with the project age starting at the availability of the first stable release of a project.

- Main points:  $P_1 = \text{Min}$ ,  $P_7 = \text{Average}$ , and  $P_{13} = \text{Max}$ .

-Mid points:  $P_4$  mid ( $P_1, P_7$ ),  $P_{10}$  mid ( $P_7, P_{13}$ ),  $P_{2.5}$  mid ( $P_1, P_4$ ),  $P_{5.5}$  mid ( $P_4, P_7$ ),  $P_{8.5}$  mid ( $P_7, P_{10}$ ), and  $P_{11.5}$  mid ( $P_{10}, P_{13}$ ).

- Max points:  $P_3, P_6, P_9$ , and  $P_{12}$ , with Max points being computed as:

$$P_{\text{Max}} = P_{(\text{Max}-0.5)} + ((0.5 * X\%) * P_{(\text{Max}-0.5)})$$

E.g., given an overlap  $X$  of 10%, the  $P_3$  term max point formula is:

$$P_3 = P_{2.5} + ((0.5 * 10\%) * P_{2.5})$$

In the next step we calculate the corresponding **overlapping** points:  $P_2, P_5, P_8$ , and  $P_{11}$ . These points represent the area of overlap between two boundaries on the fuzzy scale (e.g.,  $VP$  and  $P$ ). A larger  $X$  value indicates more uncertainty about the interpretation of the values, with overlapping points being computed as:

$$P_{\text{overlap}} = P_{(\text{next max})} - (X\% * P_{(\text{next max})})$$

Given an overlap of interest  $X$ , an overlapping point  $P_2$  is computed as:

$$P_2 = P_3 - (X\% * P_3)$$

In our approach,  $P_{14}$  (the last point) is located after the maximum measure value, since future improvements (new high scores) have to be supported.  $P_{14}$  is calculated as the highest actual measured value plus the standard deviation.

The overlapping values between fuzzy terms (indicated as  $x\%$  in Figure 58) are not fixed and are derived based on the actual measured values over time. Smaller overlaps indicate that a measure re-

mains stable or underwent only minor changes. In the case of measures that show a high fluctuation, the overlap will become larger. This overlap has been designed into our fuzzification approach, since fluctuation of values often indicates uncertainty about the measured result. Also, increasing overlaps tend to shift the assessments more towards improvement (a higher overall score). As a result, for example, values at the lower range of the *VH* scale have a higher chance of being scored as *H* because of their high degree of overlap.

For each measure, there are three types of scales that could be used:

(1) A **local scale**, based on project's specific (local) measures evolvability scores over time. The combination of individual scores and their trend analysis can be used to classify and prioritize process improvements at the product level; (2) A **global comparison scale** that compares different products within the same domain. In this case, the benchmarking (highest and lowest scores) are based on the scores obtained from the products to be compared. If a new benchmark score is obtained, the existing assessment scores in the comparison group will be adjusted accordingly. This scale is best suited for scenarios when evaluation and comparison among comparing different products in the same domain are required; (3) A **global market scale** compares the evolvability scores of different products against all assessed projects (independent from the domain). This form of assessment is best applied for comparing the evolvability of different software products created within an organization or in a global context. The global benchmarking represents the boundaries based on the results from other projects in our assessment database. Whenever a new product (assessment) is added to our assessment repository, all local benchmarks obtained from this assessment will be compared against the existing global maxima/minima boundaries for their *VH* and *VL* scales. In the case of a new maximum or minimum, the global scale for this measure will be recalculated and adjusted to reflect the availability of new (additional) data. For all scales, measures are calculated based on historical values and scales are not compared against pre-defined goals/targets.

**Create weight fuzzy scale:** Each element in our SE-EQAUM metamodel (including factors, attributes, measures, etc.) has an associated weight to reflect its importance towards the overall assessment. Default values for the weights are based on the ISO 9126 standard [15], with users being able to cus-

tomize these values to their specific needs. The input for this task is (0-9), with the output being the weight fuzzy scale {low, medium and high}, as in Figure 59. The weight fuzzy scale is calculated as is done with the measures. However, weight values are fixed and the fuzzy scale is calculated once.

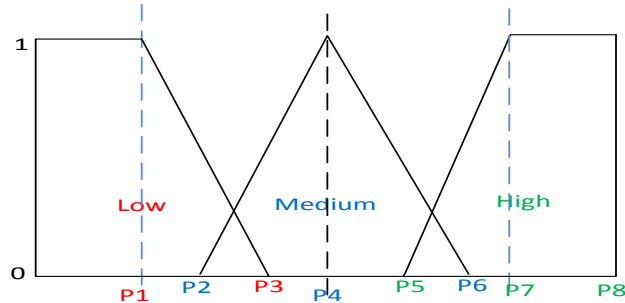


FIGURE 59 WEIGHT FUZZY SCALE

**Create score fuzzy scale:** This is the assessment weighted score. The input is fixed (0-9), and the output is a fuzzy scale {Very Poor (VP), Poor (P), Average (AVG), Very Good (VG) and Excellent (E)}; see Figure 60. Unlike measures fuzzy scale calculation, the score scales are fuzzified only once.

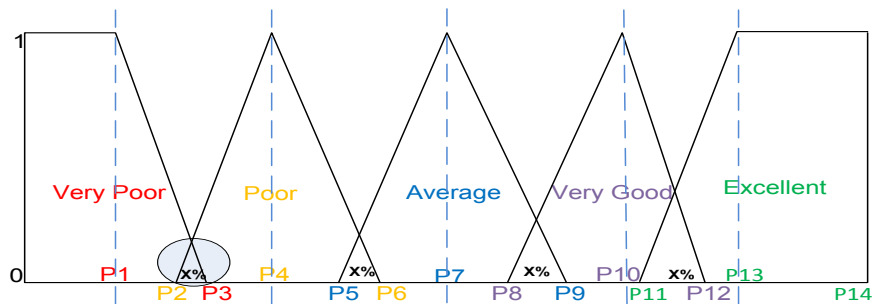


FIGURE 60 QUALITY SCORE FUZZY SCALE

### 5.1.3. ASSESSMENT (FINDING ASSESSMENT SCORES)

The input for this step is the fuzzy measure scores, and their weights and outputs are the weighted evolvability scores. The fuzzy logic-based assessment process has three main steps: fuzzification, inferencing, and de-fuzzification. The fuzzification step is covered in the previous step (creating the fuzzy scales). The inferencing and de-fuzzification steps are described here as part of the assessment.

The weighted evolvability scores are obtained by executing a set of fuzzy inference rules of the format: IF <input> THEN <output>. The input accepts AND operators for multiple inputs, where each inference rule has two inputs, a measure and weight, and one output, the overall score. As part of our assessment process, we automatically create a Fuzzy Control Language ((FCL) file per measure; refer to Table 13. FCL is published by the International Electrotechnical Commission (IEC-61131-7)<sup>71</sup> and used with the JFuzzyLogic tool [173]. The fuzzy inference engine provided with JFuzzyLogic fires the relevant fuzzy rule set. The following is an example of an auto-generated FCL file for the commits over the time measure:

**TABLE 13 SAMPLE FUZZY CONTROL LANGUAGE (FCL) FILE**

<pre> FUNCTION_BLOCK CommitsOverTime  VAR_INPUT CommitsOverTime: REAL;   weight : REAL; END_VAR  VAR_OUTPUT   score : REAL; END_VAR  FUZZIFY CommitsOverTime   TERM VERYLOW := (53.15,0.0) (64.0,1.0) (82.29,1.0) ;   TERM LOW := (38.51,0.0) (48.5,1.0) (59.06,0.0) ;   TERM AVERAGE := (23.86,0.0) (33.0,1.0) (42.79,0.0) ;   TERM HIGH := (9.22,0.0) (17.5,1.0) (26.51,0.0) ;   TERM VERYHIGH := (0.0,1.0) (2.0,1.0) (10.24,0.0) ; END_FUZZIFY  FUZZIFY weight   TERM LOW := (0.0,1.0) (2.5,1.0) (3.94,0.0) ;   TERM MEDIUM := (3.55,0.0) (5.0,1.0) (6.56,0.0) ;   TERM HIGH := (5.9,0.0) (7.5,1.0) (9.0,1.0) ; END_FUZZIFY  DEFUZZIFY score   TERM VERYPOOR := (6.5,0.0) (7.5,1.0) (9.0,1.0) ;   TERM POOR := (5.31,0.0) (6.25,1.0) (7.22,0.0) ;   TERM AVERAGE := (4.14,0.0) (5.0,1.0) (5.9,0.0) ;   TERM VERYGOOD := (2.95,0.0) (3.75,1.0) (4.6,0.0) ;   TERM EXCELLENT := (0.0,1.0) (2.5,1.0) (3.28,0.0) ;  METHOD : COG; //Center of Gravity de-fuzzification method END_DEFUZZIFY </pre>	<pre> RULE 0 : IF CommitsOverTime is VERYLOW and weight is LOW   THEN score is POOR ; RULE 1 : IF CommitsOverTime is VERYLOW and weight is MEDI-   UM THEN score is VERYPOOR ; RULE 2 : IF CommitsOverTime is VERYLOW and weight is HIGH   THEN score is VERYPOOR ; RULE 3 : IF CommitsOverTime is LOW and weight is LOW THEN   score is AVERAGE ; RULE 4 : IF CommitsOverTime is LOW and weight is MEDIUM   THEN score is POOR ; RULE 5 : IF CommitsOverTime is LOW and weight is HIGH THEN   score is VERYPOOR ; RULE 6 : IF CommitsOverTime is AVERAGE and weight is LOW   THEN score is VERYGOOD ; RULE 7 : IF CommitsOverTime is AVERAGE and weight is MEDI-   UM THEN score is AVERAGE ; RULE 8 : IF CommitsOverTime is AVERAGE and weight is HIGH   THEN score is POOR ; RULE 9 : IF CommitsOverTime is HIGH and weight is LOW THEN   score is EXCELLENT ; RULE 10 : IF CommitsOverTime is HIGH and weight is MEDIUM   THEN score is VERYGOOD ; RULE 11 : IF CommitsOverTime is HIGH and weight is HIGH THEN   score is AVERAGE ; RULE 12 : IF CommitsOverTime is VERYHIGH and weight is LOW   THEN score is EXCELLENT ; RULE 13 : IF CommitsOverTime is VERYHIGH and weight is ME-   DIUM THEN score is EXCELLENT ; RULE 14 : IF CommitsOverTime is VERYHIGH and weight is HIGH   THEN score is VERYGOOD ; END_RULEBLOCK  END_FUNCTION_BLOCK </pre>
---	---

<sup>71</sup> [http://jfuzzylogic.sourceforge.net/doc/iec\\_1131\\_7\\_cd1.pdf](http://jfuzzylogic.sourceforge.net/doc/iec_1131_7_cd1.pdf)

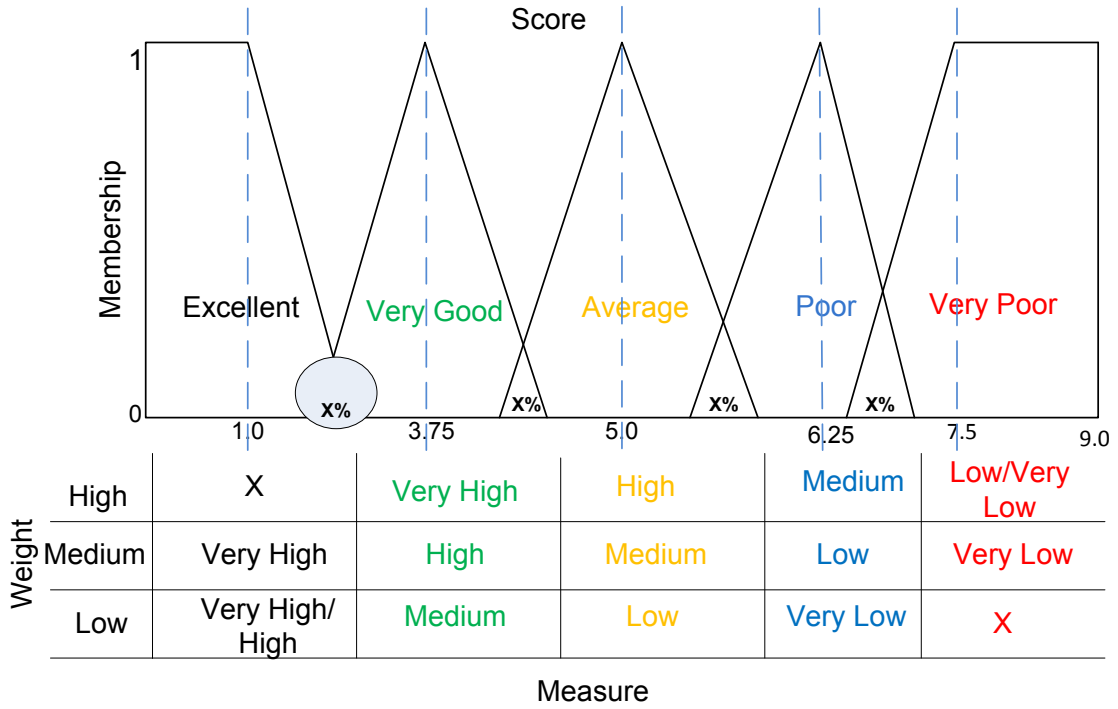


FIGURE 61 INFERRING SCORE RESULTS FOR A GIVEN MEASURE AND WEIGHT

As part of our assessment process, we made the following three *assumptions* to automate the fuzzy inference rules: (1) In cases when the weight is high then the scores are one level lower. VP scores will keep their value; (2) With low weight, scores are less relevant and the scores are one level higher. E scores keep their value; (3) With medium weight, scores keep their values. These assumptions reflect the fact that when a measure is of high importance to the assessment (high weight) then its score should be sensitive to the low measure value; therefore, the assumptions in this case are stricter for higher weights. Figure 61 summarizes the cross-reference of a measure scale (where the lower the measure value the better, which explains having Excellent in the lower scale range. This applies for measures such as code violations or style errors). The other input is the measure weight (low, medium, and high). The output is the score value in the table; for example, if the measure value is *medium* and the weight is *low* then the score will be very good (assumption 2).

Firing inference rules will calculate a final weighted measure score per snapshot and measure. Using a bottom-up approach, we calculate the attribute evolvability, where the model attributes score per snapshot (e.g., analyzability or testability) is based on the scores of its related measures. Similarly,

we calculate sub-factor scores, using their related attributes. The product evolvability score is computed based on the model sub-factors and will be a single evolvability score per snapshot.

As discussed previously, during the actual evolvability assessment, we can no longer be certain that the knowledge sources used for the assessment and benchmarking are complete and precise. Therefore, the use of crisp scoring boundaries is no longer applicable. We address this challenge by using a fuzzy logic approach, which provides: (1) the final scores that can be among two scales (e.g., VP and P) and (2) membership functions that provide the strength of a score with respect to its belonging to a certain scale (e.g., analyzability is 40% VG).

The last step in a fuzzy logic-based assessment process is **de-fuzzification**. In this step, we provide a numerical assessment score based on the calculated linguistic score (e.g., excellent) by using the center of gravity (COG) or the Centriod method used in, which is also considered as one of the most popular de-fuzzification methods [42,43].

$$COG = \frac{\sum_{i=1}^n xi MF(xi)}{\sum_{i=1}^n MF(xi)}$$
, where n represents the number of elements (e.g., score) and xi represents the

elements' values, while MF(xi) is the value of membership function of the element (e.g., 0.4 to the low and 0.6 to the medium scale). The JFuzzyLogic tool [173] automatically calculates the de-fuzzy value with the method of choice as specified in the FCL file.

#### 5.1.4. KNOWLEDGE ENRICHMENT

In this step of our process, we allow for further model enrichment with two types of knowledge: assessment result knowledge and inferred knowledge. For the assessment results, the calculated quality assessment scores are re-populated into the QUAMON ontology for further analysis, comparison, and exploration purposes. More advanced exploration is performed in the next chapter for score interpretation and prediction, while, for the inferred knowledge, we aim to add more semantic value to the knowledge base. For this purpose we sometimes use Description Logic axioms, e.g., OWL2 property chain are construct to infer new relationships among quality model levels. Another example is the population of the newly inferred concepts such as *SnapshotFile*, which is a copy of the *LogicalFile*

defined in the version control ontology, or the *CoreContributor*, who is a Contributor, as defined by the metadata ontology (METON), who made more than the average number of commits in a snapshot. Enrichment also includes adding more inference rules and more data sources or integrating scores obtained from another quality assessment model. This model enrichment will enhance the user experience while exploring our knowledge ontology model.

For example, the user can add the QUALOSS risk score (green, yellow, red, and black) as another score theme for our SE-EQUAM populated ontology. Assume that *m1* is a quality measure instance such as the evolution is number of commits, then

```
<:m1 rdf:type quamon:Measure> <:m1 quamon:hasScore "60% veryPoor"> <:m1 quamon:hasScore "black">
```

Being able to extend our knowledge base with semantically richer, new, inferred or calculated knowledge is a key to ensuring that the future assessment needs are met. It not only supports the reusability of the metamodel by extending it, it also addresses uncertainties with respect to the evolution of the assessment requirements.

## 5.2. CASE STUDY

---

The purpose of this case study is to show the steps and the results of applying our fuzzy-based quality assessment process on an existing project. For brevity purposes, the case study details the assessment process steps for one randomly selected measure (i.e. number of contributors over time), the same steps apply for the rest. The goal is also to show how the fuzzy assessment process can address uncertainties around the boundaries of two adjacent scores (e.g., poor and very poor or poor and average).

For this example, we assume that the interested stakeholder has already defined ONTEQAM as a domain model to assess the contributor's activity evolvability over time. The assessment is performed on:

**Project Name:** PMD

**Description:** Java source code analyzer.

**Artifact:** version control

**URL** <https://pmd.svn.sourceforge.net/svnroot/pmd>

**Snapshot:** We decided to study the measure evolvability over 6 months' time intervals. Given a 7 years lifespan; this result in 14 data points.

**Relevant measure:** the contributing community size for PMD project. *For the case study, we selected one measure to demonstrate the applicability of the assessment process. Similarly, the assessment can be performed for the other measures.*

We locally benchmark the measure by leveraging JFuzzyLogic [173] with its Fuzzy Control Language file (FCL) based on IEC 61131-7 ed1.0 standard, 2000.

**Measure calculation:** As described earlier in this chapter, software artifacts such as version control system in this example are extracted and populated as part of our ontological model (VERON, METON, TOOLON and QUAMON). SPARQL queries are used to obtain the non-fuzzy measure values. For example, the following SPARQL query is used to obtain the contributing community per snapshot:

```
SELECT DISTINCT ?snapshot ?res WHERE { ?cmt rdf:type veron:Commit. ?snapshot rdf:type quamon:Snapshot. cmt meton:hasContributor ?res. ?cmt quamon:inSnapshot ?snapshot.} ORDER BY ?snapshot.
```

**Fuzzification:** Here is a partial copy of the FCL file for the PMD contributing community size measure (denoted as DC). The input to be fuzzified is the measure value obtained from the previous step and its weight (in our example, we chose a medium weight value of 5. This is a customizable value), and the output to de-fuzzify is the measure score to be calculated in this fuzzification step. The points (X,Y) in each of the terms below are the three points that represent the fuzzy term in the scale of that



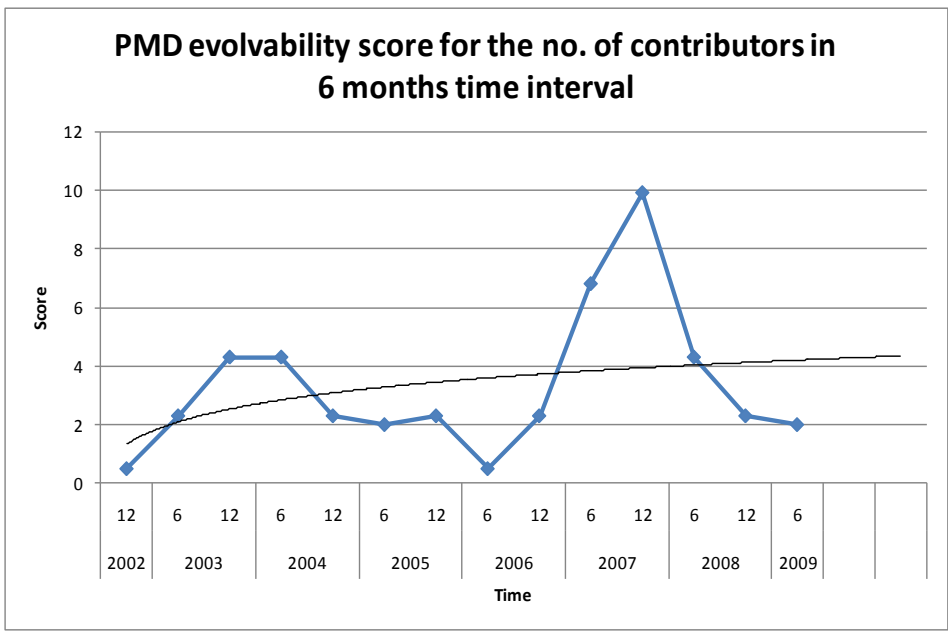
input/output value. The way to calculate each of these points is described in section 5.1.2 above (fuzzification).

```
FUZZIFY DC
  TERM VERYLOW := (0.0,1.0) (0.0,1.0) (4.13,0.0) ;
  TERM LOW := (3.72,0.0) (7.86,1.0) (12.38,0.0) ;
  TERM AVERAGE := (11.14,0.0) (15.72,1.0) (20.63,0.0) ;
  TERM HIGH := (18.57,0.0) (23.58,1.0) (28.88,0.0) ;
  TERM VERYHIGH := (25.99,0.0) (31.43,1.0) (44.68,1.0) ;
END_FUZZIFY
FUZZIFY weight //one time scale generated for values 0-9
  TERM LOW := (0.0,1.0) (2.5,1.0) (3.94,0.0) ;
  TERM MEDIUM := (3.55,0.0) (5.0,1.0) (6.56,0.0) ;
  TERM HIGH := (5.9,0.0) (7.5,1.0) (9.0,1.0) ;
END_FUZZIFY
DEFUZZIFY score //one time scale generated for values 0-9
  TERM VERYPOOR := (0.0,1.0) (2.5,1.0) (3.28,0.0) ;
  TERM POOR := (2.95,0.0) (3.75,1.0) (4.6,0.0) ;
  TERM AVERAGE := (4.14,0.0) (5.0,1.0) (5.9,0.0) ;
  TERM VERYGOOD := (5.31,0.0) (6.25,1.0) (7.22,0.0) ;
  TERM EXCELLENT := (6.5,0.0) (7.5,1.0) (9.0,1.0) ;
END_DEFUZZIFY
```

**Assessment:** The set of inference rules that are fired for each input depends on the value of that measure at that certain snapshot. Table 14 shows the corresponding evolvability score obtained for the 14 data points, including the membership of the score (the value in brackets next to each rule) and the fired rule(s). In order to have a more compact representation we use M: Measure, MED: Medium and W: Weight. For scale values: VL: Very Low, L: Low, A: Average, H: High, VH: Very High, P: Poor, VP: Very Poor, VG: very Good, and E: Excellent.

**TABLE 14 PMD MEASURE VALUES, SCORES, AND FIRED RULES**

Snapshot	Measure	Score	Fired rule
1	6.0	0.47	(0.42) if (M IS VL) AND (W IS MED) then score IS VP
2	8.0	2.29	(0.42) if (M IS L) AND (W IS MED) then score IS P
3	9.0	4.33	(0.038) if (M IS L) AND (W IS MED) then score IS P (0.42) if (M IS A) AND (W IS MED) then score IS A
4	9.0	4.33	(0.038) if (M IS L) AND (W IS MED) then score IS P (0.42) if (M IS A) AND (W IS MED) then score IS A
5	8.0	2.29	(0.42) if (M IS L) AND (W IS MED) then score IS P
6	7.0	1.95	(0.18) if (M IS VL) AND (W IS MED) then score IS VP
7	8.0	2.29	(0.42) if (M IS L) AND (W IS MED) then score IS P
8	6.0	0.47	(0.42) if (M IS VL) AND (W IS MED) then score IS VP
9	8.0	2.29	(0.42) if (M IS L) AND (W IS MED) then score IS P
10	11.0	6.78	(0.42) if (M IS H) AND (W IS MED) then score IS VG
11	13.0	9.81	(0.42) if (M IS VH) AND (W IS MED) then score IS E
12	9.0	4.33	(0.03) if (M IS L) AND (W IS MED) then score IS P (0.42) if (M IS A) AND (W IS MED) then score IS A
13	8.0	2.29	(0.42) if (M IS L) AND (W IS MED) then score IS P
14	7.0	1.95	(0.18) if (M IS VL) AND (W IS MED) then score IS VP



**FIGURE 62 PMD NO. OF CONTRIBUTOR'S EVolvABILITY SCORES**

The de-fuzzified evolvability scores are calculated using the COG (Center Of Gravity) method; the JFuzzyLogic tool [173] automatically calculated the de-fuzzified score for us. Figure 62 shows our results of the de-fuzzified evolvability scores representing the contributor working on PMD along with its trend line. We can observe that the number of people working on PMD increased initially, then stabilized for almost a year, before dropping gradually until mid 2006. Between mid 2006 and mid 2007 a significant increase in the number of people working on the project could be observed. Further analysis of the PMD data showed that in this time period PMD had two major releases 4.0 and 4.1. After the two major releases, the number of developers dropped again. This decrease in member activity is also reflected by the number of minor releases per year dropped from almost 16 per year in 2007 to 3 per year in 2009.

This example above shows how uncertainty can be handled using the fuzzy approach. Some values in the table are on an overlapping scale, like measure value 9.0, which reflects an uncertainty if its value should be scored as Low or Average. There are many reasons why uncertainty should be considered when assessing quality of open source projects such as the possibility of missing data from the artifacts (e.g., versioning history misses is used after two years) or incomplete data (e.g., there could be some bugs that are not reported on the issue tracker artifacts). To handle this uncertainty more than one inference rule has to be fired (Table 14). The fuzzy approach shows that the value 9.0 has a higher strength towards the Average scale (0.42) than the strength towards the Low scale (0.38). The rules will then be aggregated and de-fuzzified using the COG to provide the final non-fuzzy score.

In this example, we only considered one measure. Each measure is calculated the same way. In cases where more than one measures are involved, we first compute the non-fuzzy score for each measure and then aggregate these scores to calculate the attribute, subfactors, factors, and dimensions. The effect of the fuzzification will become mainly visible at the higher assessment abstractions, such as the quality dimension scores.

### 5.3. VALIDATION AGAINST QUALOSS

---

As part of our validation, we compare the results obtained from our assessment process with results from QUALOSS [40].

**Purpose:** The purpose of this validation is to check, given the same set of measure:

- Whether our evolvability assessment scores are comparable to QUALOSS. We assume that a VP/P score in ONTEQAM corresponds with a Black/Red (high risk), a A/VG score with Yellow (medium risk) and an E score should with Green (low risk).
- Whether our local assessment scores (project level scores specific to our approach) provides a finer grained assessment than the global scores (QUALOSS equivalent approach).

**Why QUALOSS?** QUALOSS [40] is the only quality model that also provides an evolvability specific assessment score. QUALOSS's benchmarking is based on their study for a large set of open source projects from the FLOSSMetrics online repository [177][40]. QUALOSS scores are mapped to a color scheme of Black, Red, Yellow, and Green, each of which corresponds to the evolvability quality from high to low risk, respectively.

For the validation, we applied our approach on the following projects: PMD<sup>72</sup>, Liquibase<sup>73</sup>, and JFreeChart<sup>74</sup>. Liquibase is an open source Java based SQL Client, JFreeChart is an open source Java based charting tool and PMD is a static Java code analyzer. For each of these tools QUALOSS provides data along with the assessment results online. The reason behind choosing these projects is the limited number of projects that has the QUALOSS quality indicators applied and provided online.

The data shown in Table 15 is based on the time intervals considered by QUALOSS assessment. The last year data was assessed in the QUALOSS project was 2009. For comparison reason we cover the same period in our analysis and benchmarking process.

---

<sup>72</sup> <http://melquiades.flossmetrics.org/projects/pmd/quality>

<sup>73</sup> <http://melquiades.flossmetrics.org/projects/liquibase/quality>

<sup>74</sup> <http://melquiades.flossmetrics.org/projects/jfreechart/quality>

TABLE 15 BASE MEASURES

Measure	PMD	Liquibase	JFreeChart
Total number of commits	6,970	864	2,205
Max. commits/month	441	82	260
Min. commits/month	0	0	0
Avg. commits/month	77	32	74
Total number of activities	39,272	17,030	10,149
Max. activities per month	1,837	7,359	1,976
Min. activities/month	0	0	0
Avg. activities/month	436	390	338
Total number of committers	31	9	3
Total number of unique files	12,468	14,469	2,639
Total SLOC	277,0243	106,083	484,298
Total LOC	728,6423	786,335	1,113,132

The following set of measures is used to assess the open source community liveness evolvability quality and is based on the QUALOSS project. Note that the names of the measures (e.g., Tb\_cm\_IWA1) correspond to the naming used and published online by QUALOSS:

**Tb\_cm\_IWA1:** Monthly analysis of the community activity like commits posts by each contributor. In the example below, for the versioning systems the commits per contributor is considered.

**Tb\_cm\_IWA2:** Monthly monitor for the increase and decrease in the community activity as number of events (regardless of the contributor).

**Tb\_cm\_SRA2:** The measure retrieves the date of the first commit for each member of the community, which will enable us to know if the number of new members committing code remains stable.

**Tb\_cm\_SRA3:** measures the first commit of each detected committer in the version control system whose commit is not a code commit (checked using the committed file extension).

**Tb\_cm\_SRA9:** Monthly analysis for the evolution in the number of contributors in the community which could help in measuring the size of a community.

We use a medium weight for all measures because QUALOSS does not consider a measure weight as an input in the assessment and a medium weight usually does not affect the actual calculated score (see our assumptions for fuzzy inference rules).

We used the global benchmarking technique for the validation against the QUALOSS results, since QUALOSS provides only global assessments.

From a score range perspective (e.g., poor/very poor corresponds to red score in QUALOSS), our ONTEQAM scores match the QUALOSS global scores (OntEQAM provides local scores as well). However, we also observed that the QUALOSS assessment result is *Red* for all three projects. Given the heterogeneity of the selected projects from domain, size, lifetime and activity dimensions, having the exact assessment score (Red) is unexpected. QUALOSS results motivated us to perform an additional analysis of the QUALOSS data.

The Data provided by QUALOSS via SQL dumps for the project quality (e.g. fm3\_jfreechart\_quality<sup>75</sup>) is incomplete and does not match the results obtained from running their own queries against CVSAnalys tool data that QUALOSS uses to calculate measures related to version control system. Using fm3\_jfreechart\_cvsanaly2\_svn\_scm<sup>76</sup> database we calculate the monthly number of commits per contributor (**tb\_cm\_iwa1**):

```
SELECT year(s.date), month(s.date), COUNT(s.rev)/COUNT(distinct s.committer_id) `cm-iwa1` FROM scmlong s GROUP BY year(s.date), month(s.date);
```

Then we compare the results against fm3\_jfreechart\_quality<sup>75</sup> database that QUALOSS also provides:

```
SELECT year, month, value FROM fm3_jfreechart_quality.tb_cm_iwa1_scm t where value >0 group by year, month;
```

---

<sup>75</sup> [http://melquiades.flossmetrics.org/data/projects/jfreechart/fm3\\_jfreechart\\_quality\\_20091123T11:45:01.sql.gz](http://melquiades.flossmetrics.org/data/projects/jfreechart/fm3_jfreechart_quality_20091123T11:45:01.sql.gz)

<sup>76</sup>

[http://melquiades.flossmetrics.org/data/projects/jfreechart/fm3\\_jfreechart\\_cvsanaly2\\_svn\\_scm\\_20090304T06:42:18.573465.sql.gz](http://melquiades.flossmetrics.org/data/projects/jfreechart/fm3_jfreechart_cvsanaly2_svn_scm_20090304T06:42:18.573465.sql.gz)

We found that JFreeChart **tb\_cm\_iwa1** misses results of 20 months. Our finding suggests that the justification of QUALOSS score results is related to (1) the incompleteness of data provided by QUALOSS (e.g. JFreeChart assessment actually considered 6 out of 26 data points) and (2) the focus of QUALOSS on global benchmarking as all projects are benchmarked on the same scale regardless of their size, domain and community activity. The validation against the incomplete dataset is marked as Global (I) in Table 16 where (I) means Incomplete.

Using our own implementation for data extraction, we assessed evolvability quality both locally and globally for the same projects and for the same set of measures as of QUALOSS. In order to compare to QUALOSS, we first used the same (incomplete) dataset by excluding the datapoints not considered in QUALOSS results, this is marked as incomplete (I) in Table 16. The incomplete scores are global only because QUALOSS approach corresponds to the global approach in our assessment. Yet, we are still interested in the assessment results for the complete dataset and how it compares to the incomplete one. Thus, we provided both local and global scores for the complete dataset. The assessment results taken from the complete set is different as we can see from Table 16.

**TABLE 16 ONTEQAM LOCAL & GLOBAL SCORES VERSUS QUALOSS (THREE OSS PROJECTS)**

Measures	PMD				Liquibase				JFreeChart			
	ONTEQAM			QUALOSS	ONTEQAM			QUALOSS	ONTEQAM			QUALOSS
	Global (I)	Local	Global		Global(I)	Local	Global		Global (I)	Local	Global	
iwa1	VP	P	P	Red	VP	A	A	Red	VP	P	P	Red
iwa2	VP	P	P	Red	VP	P	P	Red	VP	P	P	Red
sra2	VP	P	P	Red	P	P	VP	Yellow	VP	A	P	Red
sra3	VP	P	P	Red	VP	P	VP	Red	VP	VG	VP	Red
sra9	VP	A	A	Red	VP	P	VP	Red	VP	P	VP	Red

Below we further analyze the assessment results obtained for each of the projects for the selected set of measures in Table 16:

**Tb\_cm\_IWA1:** Considers the evolution of activities per committer based on information extracted from the versioning system. We could observe that for all three projects the local and global scores remained the same. This means for example, Liquibase had an average evolvability of the commits per committer over time on its own scale level considering different time intervals and it's also was considered average when compared with PMD and JFreeChart.

**Tb\_cm\_IWA2:** The recorded activity of the versioning system on a monthly basis, over the three projects. The activity for all three projects is considered as poor (P), which corresponds to the Red score in QUALOSS. For further validation, we compare this result against Ohloh [178] and we found that Liquibase has some high activity peaks over time compared to the other two projects but these high peaks are balanced out with the low activity at the beginning of the project lifetime so the overall evolvability is still poor.

**Tb\_cm\_SRA2:** Evaluating this measure we noticed that locally JFreeChart would obtain an Average (A) score but when compared with other projects its assessment drops to P. This outcome was expected since the total number of committers in JFreeChart is much lower compared to PMD (Table 16).

**Tb\_cm\_SRA3:** For this assessment, Liquibase's score drops by one score during the global benchmark. JFreechart on the other hand drops in this case by three scores. This more dramatic drop is due to the small number of committers in JFreeChart. The project has a local maximum of three committers compared to a benchmarked global maximum of 31. This result shows how local and global benchmarking results can vary significantly. Using existing assessment models, projects like JFreeChart will always score low because of its small contributor base, when compared with other projects. In particular, since the other projects could be in a different domain, size and activity. A local benchmarking is another, more specific product level assessment that is interesting for stakeholders looking for evaluating whether a certain product's evolvability is average or poor.

**Tb\_cm\_SRA9:** As expected, PMD having the largest number of contributors in this evaluation, it will score high in both the local and global benchmark.



Summarizing the above, scores might vary significantly when compared locally or globally, for example iwa1 measure for Liquibase in Table 16 scored Very Poor globally (Red in QUALOSS) while it was Average locally. The reason why the local and global scores vary is because of the different scales used to classify the score in a certain range (poor, very poor, or average range). As explained earlier in this chapter, the local scores obtain their scale from the scores of the assessed project (min and max scale ends are the min and max values calculated for that project over its lifetime) while the global scale is calculated based on the set of all the projects used in the assessment. We do believe that allowing for local and global scales of the assessment scores; will accommodate better the stakeholder requirements. The local score is useful for stakeholders who are interested in the project evolution over time by calculating its new scores relative to its own values rather than relative other projects. Local scores on the other hand are better indicators for assessing, how a project quality enhanced or declined compared to its historical growth. Assessing globally is useful when the projects are comparable from several perspectives such as size, lifetime, and domain. Global scores for two or more projects could be used for comparison to help adoption decisions.

## CHAPTER 6: EVOLVABILITY PREDICTION USING FINANCIAL TECHNICAL ANALYSIS

In this chapter, we first describe existing approaches used in the software domain to predict software quality. Then we discuss the financial technical analysis and the most popular indicators and patterns used in the stock market domain to interpret and predict the trends of price changes over time. The aim of this chapter is to provide the foundation needed to understand one of our research contributions i.e. applying the financial technical analysis used in predicting stocks price to predict software quality. The detail of our contribution is covered in the next chapter.

### 6.1 SOFTWARE QUALITY PREDICTION MODELS

---

Quality prediction claims a particular quality value in the future based on the analysis of its historical values [95]. Predicting software quality provides an early assessment of possible faults in software, which then reduces the maintenance effort, time, and costs [95, 96]. Prediction models are used to support process improvements and decision making.

Prediction models use historical data to predict future quality trends of software projects. To predict software quality, existing research analyzed the evolution of its change requests (CR) [97], defects [98], size [99], code changes [100], and social interactions [101].

There are several techniques that have been developed to predict software quality, each tested on a different datasets [96, 102]. Software size and complexity metrics are traditionally used to predict quality, mostly through predicting defects. In 1999, Fenton [103] criticized traditional software defect prediction models, including source code size and complexity, which are testing process prediction based on early defect inspection and design and development process prediction such as the Capability Maturity Model (CMM)<sup>77</sup>. In [103] the author defined two problems: first, the defect

---

<sup>77</sup> <http://cmminstitute.com/>

definition itself is unclear (i.e. post-release, total known defects, or after a pre-defined point of time). Second, the problems with each prediction model (e.g., size) are programming language dependent; the complexity measure formula ignores special control flow cases, such as crossing edges and type, and the models ignore design faults and defect severity.

Prediction techniques based on Artificial Intelligence (AI) such as fuzzy logic [80, 87] were popularized during the 2000s. A comparative study of these AI techniques was published in 2006 [95]. This study claims that the problem with regression models is that the prediction results are not reliable because of the simultaneous use of highly correlated measures [95]. Most thresholds are defined using sample data that do not necessarily reflect real life trends, which is why quality values need not be crisp; in this case, fuzzy logic is used to address the loss of accuracy at the boundaries of the predicted value. The study suggested using fuzzy logic with a genetic algorithm to automate the data generation and reduce the cost [95].

A study held in 2008 [96] of existing prediction techniques compared 30 different software quality prediction techniques on two public datasets published on PROMISE [104], e.g., case-based reasoning, Bayesian network, regression methods, voted perception, decision trees, etc. The authors divided the dataset into 66% training data and 34% testing data. Performance is evaluated using precision, recall, specificity, and accuracy of prediction; the error rate for each technique is evaluated using the difference between the actual and the predicted values. *The comparison classified regression techniques as one of the best techniques from performance and error rate perspectives.* Zimmermann [98] used the linear *regression* model to predict defects in three releases of Eclipse using the project's complexity metric. Precision, recall, and accuracy are calculated based on whether a file or package in Eclipse will have at least one predicted defect. A positive correlation is found between the complexity of the file/package and the number of defects. The same positive correlation is found between the pre- and post-release defect counts.

In 2010 [105] published a comparative study of five prediction models, including the linear regression models used in [106] and [107]. The authors applied the prediction models on the same dataset of 11 open source projects and 328 versions. The results show a quality trend (quality is

improving, constant, or deteriorating) for each project produced by each prediction model. The conclusion was that different models provide different conclusions for the same project. Another unexpected conclusion was the lack of correlation between the results obtained from the two linear regression models used in [106] and [107].

Another popular prediction technique is time series analysis. The time series technique used to analyze historical data since the 1980s by Yuen [108], whereby the author used time series analysis, more specifically the ARIMA (auto-regressive, integrated, moving average) model [109], as a prediction model. ARIMA is applied on five successive releases of an operating system dataset used by the author in previous work. The data is collected at the maintenance phase on a global/system level and on individual release level. The goal is to observe the dynamic evolution behavior using the auto-regression model formula. The results show that predicting the dynamic behavior of the system is more accurate on a detailed level. The global view of collective system components tends to hide evolutionary trend patterns, especially for complex systems.

In 1999, Kemerer and Slaughter [22] used the ARIMA model to predict evolvability by analyzing the history of the monthly code changes of 23 commercial projects that had a 20-year history. Both Kemerer [22] and Yuen [108] showed that the time series analysis approach did not provide any insight about the software evolution process, due to the randomness of the data [22].

More recently, in 2007 [99], the time series analysis ARIMA prediction model was applied on the version control system of three open source software to predict the evolution in size over the project's lifetime; the sample period of time used in this study was one day. The author [99] argues that the choice of ARIMA is because the linear regression model (which is a more popular prediction model for evolution trend) is not suitable for short-term evolution trend prediction. The validation is performed against the prediction of size for the last year.

The study in [110] shows the number of new change requests per KLOC in order to forecast and identify future trends. An increasing trend indicates either more interest of customers in new features or a decline in quality. A decreasing trend either indicates that the software is stabilized and

at a mature level or has become less popular. Time series analysis was used on the five-year history of Eclipse, Mozilla and JBoss; a two weeks' period was considered. The prediction accuracy provided differs from one project to another (20% in 80% of the cases in Mozilla, 25% in 70% of the cases in JBoss, and 25% in 50% of the cases for Eclipse) [110].

More recently, Mining Software Repositories (MSR) was used to both mine the history (past versions) of software artifacts (such as source code, documentation, bugs, etc.) and to predict future software-changes to the software artifacts. The current MSR research focuses on mining either individual or correlate analysis results from different repository types for change prediction, program comprehension, and process improvement and on planning evolutionary aspects. The most common repositories being mined are versioning systems and issue trackers [111-114]. Some of the evolution aspects they analyzed and visualized included developer's effort, detecting core developers, re-assessing development effort, and hotspot detection. In [115] an approach for mining email archives or public Free/Libre Open Source Software (FLOSS) mailing lists for open source projects was introduced. In [116] inline source code documentation was mined through the QUASOLEDO project and its quality assets, to investigate its effect on software evolution. The approach automatically measures completeness, quality, readability, and evolution of inline documentation.

On the other hand, change prediction is a well-established area that typically focuses on short-term prediction for a current change context by applying some form of Impact Analysis (IA) [119]. IA focuses on determining the effect of a modification (ripple and side-effect analysis) on software artifacts. Traditional IA methods, unlike MSR, focus on the most current versions of the software artifacts.

In our approach, MSR is used to extract both basic and value-added information from different software repositories [120].

## 6.2 FINANCIAL TECHNICAL ANALYSIS

---

Over the past fifty years stock market prices have been one of the most analyzed data [140]. The financial community assesses and analyzes fundamental qualities of a stock to predict future stock performance by considering various external factors (e.g., competition and global trends) and internal factors (e.g., earnings, product cycles, current, and historical stock price). Stock market analysts perform various types of analysis techniques to forecast/predict potential price trends for individual stocks or the stock market in general.

In the 1940s, Roberts Edwards and John Magee introduced the term *technical indicator* [141]. *Technical indicators* are metrics derived from the stock price value, such as Moving Average (MA), Moving Average Convergence Divergence (MACD), Relative Strength Indicator (RSI), and Bollinger Bands Indicator (BBI) [141]. Indicators are used to confirm price movement and to form a buy/sell signal. These indicators are used as parts of the technical analysis to assist traders make buy/sell decisions. These technical indicators are typically associated with some *technical patterns*, which are used to interpret price movement/directions (e.g., uptrend, downtrend, or side-to-side). These financial *technical patterns* are based on the assumption that history repeats itself and that knowledge is captured from the past in the form of reoccurring patterns. There are two types of patterns, which are distinguished in the technical analysis domain: reversal and continuation. A reversal pattern signals that a prior trend will reverse on completion of the pattern. Conversely, a continuation pattern indicates that the prior trend will continue onward upon the pattern's completion.

Success in trading stocks depends on timing the trades well. For years, stock traders have depended on two major tools: fundamental analysis (1), based on company performance and growth projection, and (2) technical analysis using technical indicators, which are based on the analysis of the trade history of a security through charts and mathematical formulas.

*Technical indicators* can be classified [142]: *oscillators* or *leading indicators*, and *lagging indicators*. Leading indicators represent a form of price momentum over a fixed look-back period, which is the

time lapse used to calculate the indicator. Leading indicators are usually used for prediction because they change before an event that leads to a price trend change. For example, a 20-day stochastic oscillator would use the past 20 days of price action in its calculation. All prior price action would be ignored. Some of the most popular financial leading indicators include Commodity Channel Index (CCI), Momentum, Moving Average Convergence-Divergence (MACD), and Relative Strength Index (RSI) [142]. In contrast, a lagging indicator follows a price change. These types of lagging indicators are commonly referred to as trend-following indicators and work best when markets develop strong trends. Lagging indicators are designed to get traders in and keep them in as long as the trend is intact. Some popular trend-following indicators include moving averages (exponential, simple, and weighted).

Technical analysis divergence is said to occur when an indicator movement does not agree with the price movement. Divergence can be *bullish* or *bearish*. If the indicator is making lower highs when the price is making a higher high, there is supposed to be a bearish divergence. In the same way, when the indicator makes higher lows when the price makes lower lows there is a bullish divergence. Divergence indicates a reversal in the current trend.

In what follows, we introduce some of these commonly used technical financial indicators used in the stock market analysis domain. In the stock market a combination of different patterns and indicators are used to make a trading decision. Some indicators are used as a confirmation to another indicator, while we use other indicators to show the price pattern (e.g., uptrend or downtrend); Table 17.

**TABLE 17 A SUMMARY OF TECHNICAL INDICATORS AND THEIR ASSOCIATED PATTERNS**

<b>Technical Indicator</b>	<b>Associated Pattern(s)</b>
Moving Average (MA)	Bullish/Bearish Crossover Pattern
MA Trend line	Uptrend/Downtrend Direction Pattern
Support and Resistance Trend lines	Support and Resistance Breakout Patterns
Bollinger Bands	M-Tops and W-Bottoms Patterns

### 6.2.1 MOVING AVERAGE (MA) INDICATOR

**Motivation:** MA is one of the most fundamental and widely used financial technical indicators in the stock market. Bigalow and Elliot [143] have pointed out that MAs are considered to be the most profitable technical trading rules. A simple moving average is an indicator that calculates the average stock price over a specified number of periods. If a security is exceptionally volatile, then a moving average will help data smoothness. MA filters are commonly used with time series data, as they smoothing out random noise and provide a cleaner perspective on the overall price activity. The length of the moving average varies from short-, to medium-, to long-term. In trading, there are some popular lengths, for example 20 data points for short-term predictions, 50 data points for traders interested in medium-term analysis, and a 200 data point MA is a popular length for a long-term investment analysis. More details about the choice for the MA length are provided in the application and example section below.

**Intent:** In the financial domain, MAs have been widely used to analyze if a stock (or financial markets in general) might be in a rising (bull) or falling (bear) trend. A buy signal is generated when the most current data value rises above the MA, and a sell signal is generated when the current value falls below the MA. If there is an actual clear trend occurring, this analysis approach works well. If, however, the market is moving sideways or if there were excessive volatility, many whipsaws (false signals in this case) patterns occur. Whipsaw patterns involve short-term upward movement in price, followed by a drastic, longer downward move or, alternately, when prices drop shortly followed by a suddenly longer upwards move.

**Application and Example:** The basic MA is computed as  $MA = \left( \sum_{j=1}^n Y_j \right) / n$ , with  $n$  representing the number of data points that are considered when calculating the MA (e.g., 20, 50, or 100 data points), and  $Y$  being the actual values for each individual data point.

The selection of an appropriate short- or long-term MA depends on the application context and objective. Short-term MAs are capable to react and follow stock price changes more closely and are



therefore more likely to be whipsaw-prone. In contrast, longer term MAs smooth out random noise and provide a cleaner perspective on the overall price activity by only highlighting major trend changes. A combination of short- and long-term MA types is widely used by traders to support their buy/sell decisions. MAs with different time intervals (e.g., 20-MA and 100-MA) are often applied to establish the existence (or absence) of trends and stock price directions. As part of their analysis, traders will obtain different MA trends during a period.

**Known-uses:** indicators that can trigger buy/sell signals in the financial markets.

## 6.2.2 MOVING AVERAGE CROSSOVER PATTERN

**Motivation:** These crossovers are by far the most commonly used of the MA methods and have been the subject of a lot of research. MA crossover happens when a short-term MA crosses through a long-term MA. This signal is used to identify that momentum is shifting in one direction and a strong move is likely approaching. The crossover MA technique attempts to reduce whipsaws while minimizing the lateness of signals received. There is, more or less, an agreement among technical analysis researchers on the ranges for identifying the short-, medium-, and long-term trend periods, which are less than 20 days, 20 to 100 days, and more than 100 days, respectively<sup>78</sup>. The general rule in this case is:

- Buy signal is generated when the short-term average crosses above the long-term average; and
- Sell signal is generated when the short-term average crosses below the long-term average.

**Intent:** To support trading sell/buy decision based on occurring patterns. This pattern defines two crossover types: bullish, which indicates a buy signal, and bearish crossover, which indicates a sell signal.

The premises for these behaviors are that a price that is moving up (or down) during period  $t$  is likely to continue to move up (or down) in period  $t+1$  unless there is evidence to the contrary. When the

---

<sup>78</sup> <http://www.investopedia.com/university/movingaverage/movingaverages1.asp>

short-term average crosses above the long-term average, this means that the average prices over the short period are relatively higher than those MA that cover longer periods and hence the prices have an upward momentum. This provides a lagged indicator that the price is moving upward relative to the historical prices. The opposite is true when the short-term average moves below the long-term average. Thus, the signals generated through the crossovers are very objective, which is why it is so popular.

**Application and example:** Crossover patterns occur when a short-term MA crosses a longer term MA. Figure 63 shows Google Inc. 20 and 50 days MA over a course of six months where the blue line is the stock price per day.

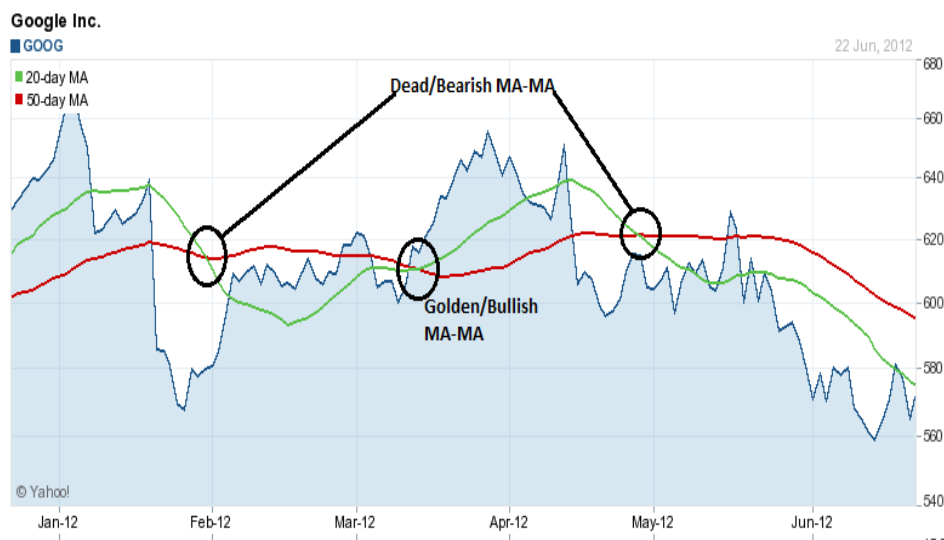


FIGURE 63 GOOGLE INC. 20/50 DAYS MA (6 MONTHS RANGE)<sup>79</sup>

If the longest MA (50 days shown in Figure 63) is at the bottom and the shortest MA is at the top then it is the correct order of an uptrend. A downtrend is the opposite. These uptrend/downtrends are usually supportive indicators, which are used in connection with other indicators to make final buy/sell decisions. Figure 64 zooms around the last crossover during the month of May for Google Inc., where one can observe how the stock entered in an uptrend after the three MAs (20 and 50

<sup>79</sup> <http://ca.finance.yahoo.com>

days) crossed over. Trend lines are simply straight lines drawn between at least two points (the point could be an actual price or a MA value), indicating an up or downtrend.



FIGURE 64 GOOGLE INC. 20 AND 50 DAYS MA CROSSOVER<sup>79</sup>

### 6.2.3 DYNAMIC SUPPORT/RESISTANCE TREND LINES AND RELATED BREAKOUT PATTERNS

**Motivation:** Moving averages have also been applied to determine dynamic support and resistance levels. They are dynamic since, in contrast to traditional horizontal support and resistance lines, these support/resistance lines are constantly changing depending on recent price action. Technical analysis attempts to gauge the strength and direction of a trend, based on the general assumption that once a trend in motion, its direction will continue for some time. Many traders use these moving averages based trend lines to determine key support or resistance levels for a stock, with support lines being price levels at which prices tend to bounce up again (unless the support line is actually broken) and resistance lines being price ceilings, which are typically not easily passed by a stock. MA helps better identify the support and resistance lines (Figure 65). To find the support and resistance lines, the following formulas are used based on Pivot Point (PP):

$$PP = (High\ Price + Low\ Price + Close\ Price) / 3$$

$$Resistance\ Line = (2 * pp) - Low\ Price$$

$$Support\ Line = (2 * pp) - High\ Price$$

**Intent:** To support traders by providing patterns based on sell/buy decisions. When shares prices approach the support lines, buyers become more likely to buy and sellers less likely to sell. At support price levels, buyers tend to enter the market, therefore preventing further price drops. After a share price drops below the support, this support line often becomes the new resistance level; this is because investors want to limit their losses and will sell later, when prices approach the former level.

Resistance occurs when the price hits an upside barrier, where an increase in the number of sellers and a decrease of new buyers can be observed. If the current resistant level holds (i.e., no breakout is observed), the supply of shares will remain larger than demand, often preventing the share price to rise above the resistance level. A resistance level is broken; this resistance level becomes often the new support level.

**Application and Example:** A pivot point (PP) analysis is often used in conjunction with calculating support and resistance levels, similar to a trend line analysis. In a pivot point analysis, the first support and resistance levels are calculated by using the width of the trading range between the pivot point and either the high or low prices of the previous day. The second support and resistance levels are calculated using the full width between the high and low prices of the previous day.

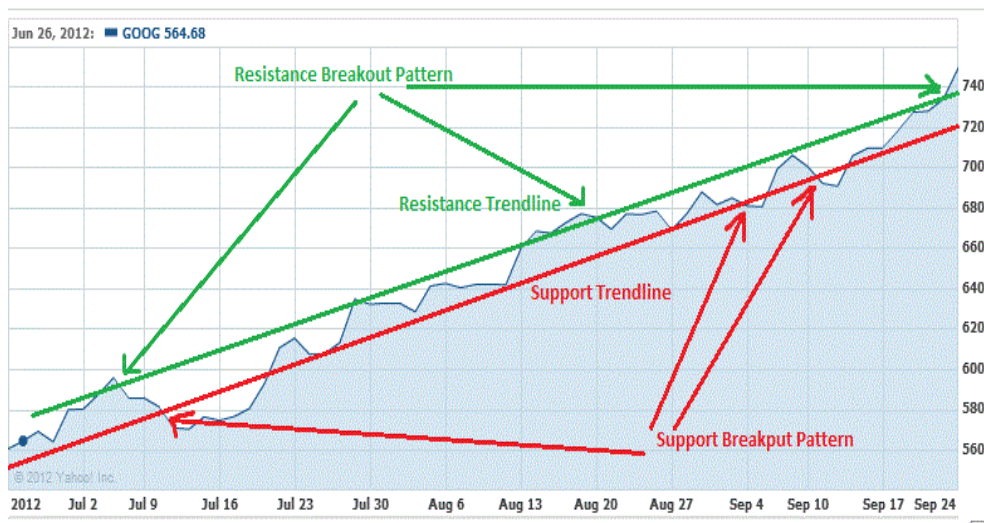


FIGURE 65 GOOGLE INC. 20 DAYS MA, INCLUDING SUPPORT AND RESISTANCE LINES<sup>79</sup>

## 6.2.4 MACD (MOVING AVERAGE CONVERGENCE/DIVERGENCE) INDICATOR

**Motivation:** Defined in the 1970s by Appel [144], this is one of the most popular analysis indicator derived from MA.

**Intent:** Investors should pay attention to crossovers of the two signal lines (MACD Short and MACD Long). A bullish pattern might occur if the MACD Short crosses the MACD Long from below and a bearish pattern can be observed if the MACD Short crosses the MACD Long from above. Also in some cases only a Zero Line crossover has been used as a signal. Buy when the MACD line crosses the zero line, and sell when the MACD line crosses below the zero line.

**Application and Example:**

$$MACD = 12MA - 26MA$$

$$Signal\ Line = 9MA$$

MACD represents the difference between 26-MA and 12-MA. It uses a signal line (9 periods MA) or zero line as the crossover baseline. Price *convergence* occurs when the 12-MA and 26-MA lines move *toward* each other and *divergence* happens when they move *away* from each other. Bullish divergence signal is generated when MACD rises *above* the signal or zero line (buy), while its bearish divergence signals when MACD drops *below* the signal or the zero line (sell). Figure 39 shows Google Inc MACD.

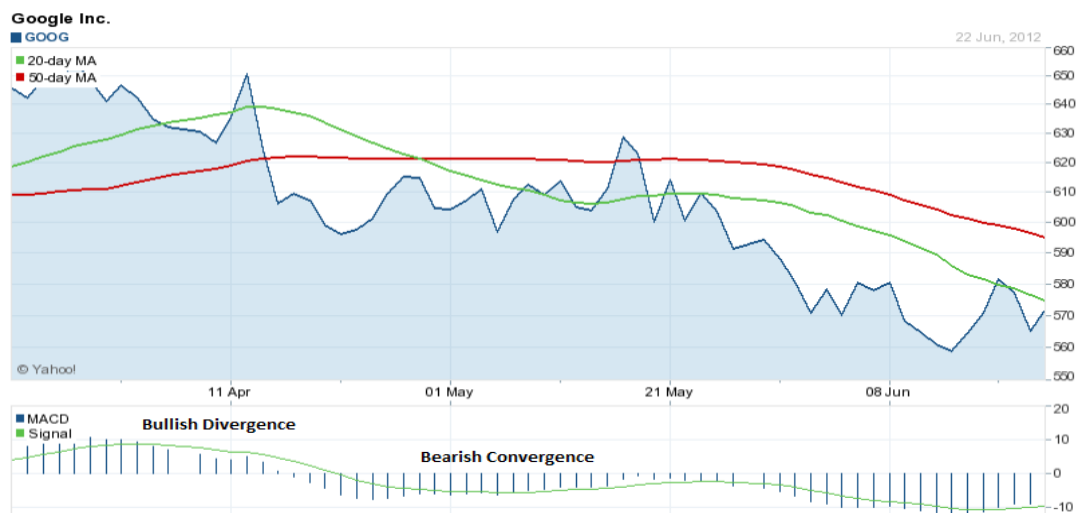


FIGURE 66 GOOGLE INC. 6 MONTHS MACD

## 6.2.5 BREAK-OUT PATTERNS: W-BOTTOM AND M-TOP PATTERNS

**Motivation:** W-Bottom and M-top patterns indicate the combined price fluctuation in a pattern that resembles “W” or “M” shapes that are widely seen and easily recognizable price movements.

**Intent:** (1) To show price evolution in its movement from a weaker to a stronger pattern, as detailed below; (2) Being reversal patterns, they define an upcoming uptrend/downtrend in price movement.

**Application and Example:** A W-Bottom pattern occurs when the price first drops close to the Lower Bollinger Band (LBB) (not necessarily crosses it); then, second, it bounces up toward the Middle Bollinger Band (MBB). Third, the price drops to usually lower low but if this low holds above the LBB then it indicates a stronger price signal that finally strongly moves upward, which confirms a bullish signal. There are different varieties of W-Bottom patterns; some are considered weaker than the others. Table 18 below shows the 16 different patterns arranged from the weakest (W1) to the strongest (W16).

The M-Top pattern is opposite to the W-Bottom; the M-Top pattern is detected when the price first rises upward toward the Upper Bollinger Band (UBB), second pulls back to the MBB, and third moves higher than the prior high (usually does not cross UBB), which is a sign that the rise in the price is not strong enough, which is confirmed by the fourth move of the price downward as a bearish signal. Below, we show the 16 different M-Tops (M1 to M16).

There exist many variations of these pure W-Bottom (Table 18) and M-top (Table 19) patterns, with some of these variations being weaker in their prediction accuracy than others. Also these patterns evolve over time through different levels of maturity.

The **Root** level represents a first indicator for the existence of a pattern and therefore is also considered the least reliable form of that particular pattern. The remaining **evolution** levels of a root pattern confirm the existence of a particular (root) pattern. The tables below show the evolution step as a dotted red line.

TABLE 18 W-BOTTOM ROOT PATTERNS EVOLUTION

W-Bottom							
Root		Level1		Level2		Level3	
W1		W2		W3		W8	
W4		W5		W10			
W6		W7		W11			
W9		W13		W14			
W15		W16					

TABLE 19 M-TOP ROOT PATTERNS EVOLUTION

M-Tops							
Root		Level1		Level2		Level3	
M2		M1					
M8		M4					
M11		M10		M3		M5	
M13		M12		M6			
M16		M15		M7			
				M14		M9	

## 6.3 SUMMARY

---

As part of this chapter, we discussed software models currently used in analyzing and predicting different software qualities such as defects, change request, and maintainability.

We introduced financial technical analysis which has been widely applied within the financial domain to interpret and predict market movements. Financial market analysis is a mature, widely applied domain and shares some common goals with assessing and predicting evolution qualities in software systems. Both domains deal with the mining of existing data and have to deal with effects of external/internal factors. In addition indicators and associated trend patterns were introduced which are commonly used in the financial sector to identify and predict reoccurring patterns and trends. The next chapter will discuss how some of these financial indicators and their associated patterns can be reapplied to the software assessment domain, to identify and predict software quality trends and patterns.



## CHAPTER 7: EVOLVABILITY SCORE INTERPRETATION AND PREDICTION

In Chapter 6, we described the financial technical analysis and the most popular indicators and patterns used in the stock market to analyze the trends of the price changes over time and to predict future prices. The contribution of our research is to investigate the applicability of existing stock market indicators and associated trend patterns for the analysis and interpretation of quality assessment scores obtained from assessing software systems. More specifically, we selected Moving Average (MA) based indicators and patterns due to their wide use in the financial domain for assessing, predicting, and interpreting share values and reapplied them in a software assessment context to predict and interpret results from quality assessments (values) of software systems. Moving Averages are commonly used with data to smooth out short-term fluctuations and highlight longer-term trends in the stock market. We map different financial indicators and patterns, e.g., M-Tops, W-Bottoms, and MA crossovers to the assessment scores obtained from open source projects, and perform a qualitative and quantitative analysis to determine the applicability of financial patterns for the software domain.

**Motivation:** Like the financial markets, many social, technical, and environmental factors affect the evolution of software systems. In [9, 10] eight laws of software evolution were introduced describing software evolvability as continuing change, resulting in increasing complexity and declining quality of a software product. In [9, 10] software evolvability is defined as the ability to maintain and enhance a software product continuously over its lifetime without a significant decline in its overall quality. Common to both financial markets and software products is the need to analyze and predict trends from large historical data repositories. The financial community assesses and analyzes fundamental qualities of a stock to predict future stock performance by considering various external factors (e.g., competition and global trends) and internal factors (e.g., earnings, product cycles, current and historical stock price). Similarly, the software domain also considers internal factors (e.g., product) and external factors (e.g., community) when assessing the quality of a software system.

**Goal:** In this research, our main objective is to introduce a cross-disciplinary approach that investigates whether well-established financial market analysis techniques can also be applied in the software domain. Similar to software design patterns, these financial patterns provide a pattern description, a context of use and a method of capturing expert domain knowledge, as well as an interpretation of the analysis results. As part of our research, we are particularly interested in the following research question: *Can the interpretation and prediction of evolvability trends be supported by the indicators and patterns traditionally used in financial markets?*

## 7.1 FINANCIAL TECHNICAL ANALYSIS PROCESS

The financial technical analysis is the process of interpreting and predicting software quality scores using financial technical indicators and patterns. In order to analyze the software quality trends, we adopt the financial technical analysis process. The input for this analysis process is the quality assessment scores obtained by any quality assessment model (e.g., OntEQAM for evolvability quality [17]). The output is the set of financial technical indicators and associated patterns in Table 20:

**TABLE 20 SUMMARY OF TECHNICAL INDICATORS AND ASSOCIATED PATTERNS**

<b>Technical Indicator</b>	<b>Associated Pattern(s)</b>
Moving Average (MA)	Bullish/Bearish Crossover Pattern
MA Trend Line <sup>80</sup>	Uptrend/Downtrend Direction Pattern
Support/Resistance Trend Lines	Support/Resistance Breakout Patterns
Bollinger Bands Indicators (BBI)	M-Tops and W-Bottoms Patterns

Similar to the financial markets, a dataset for the actual analysis will be required. The dataset should contain sufficient data points to allow for the mining of different indicators (e.g., MA 20/50); in our research, the input dataset consists of the quality assessment scores. Depending on the actual analysis and prediction context, these quality scores can be obtained at different granularity levels, ranging from the system level to the artifact level (e.g., evolution of commits over time as measure for

---

<sup>80</sup> Trend line is a line drawn between two prices over time, used to interpret price movement/trend as rising, falling, or stable.

the evolution of the version control artifact) with snapshots being taken at regular time intervals (similar to the daily stock market price). In the next step, we calculate the financial indicators (e.g., MAs and Bollinger Bands) to use them in the score trend analysis; these same indicators are also used as support indicators for the analysis patterns identified in the pattern detection step. The pattern detection highlights the reoccurrences of popular financial patterns, such as W-Bottoms and M-Tops. In the last step, we combine the results of the previous two steps (indicators and patterns) for further interpretation of the applicability of these patterns and the prediction results in the software domain. The steps we follow in order to achieve this goal are summarized in Figure 67.

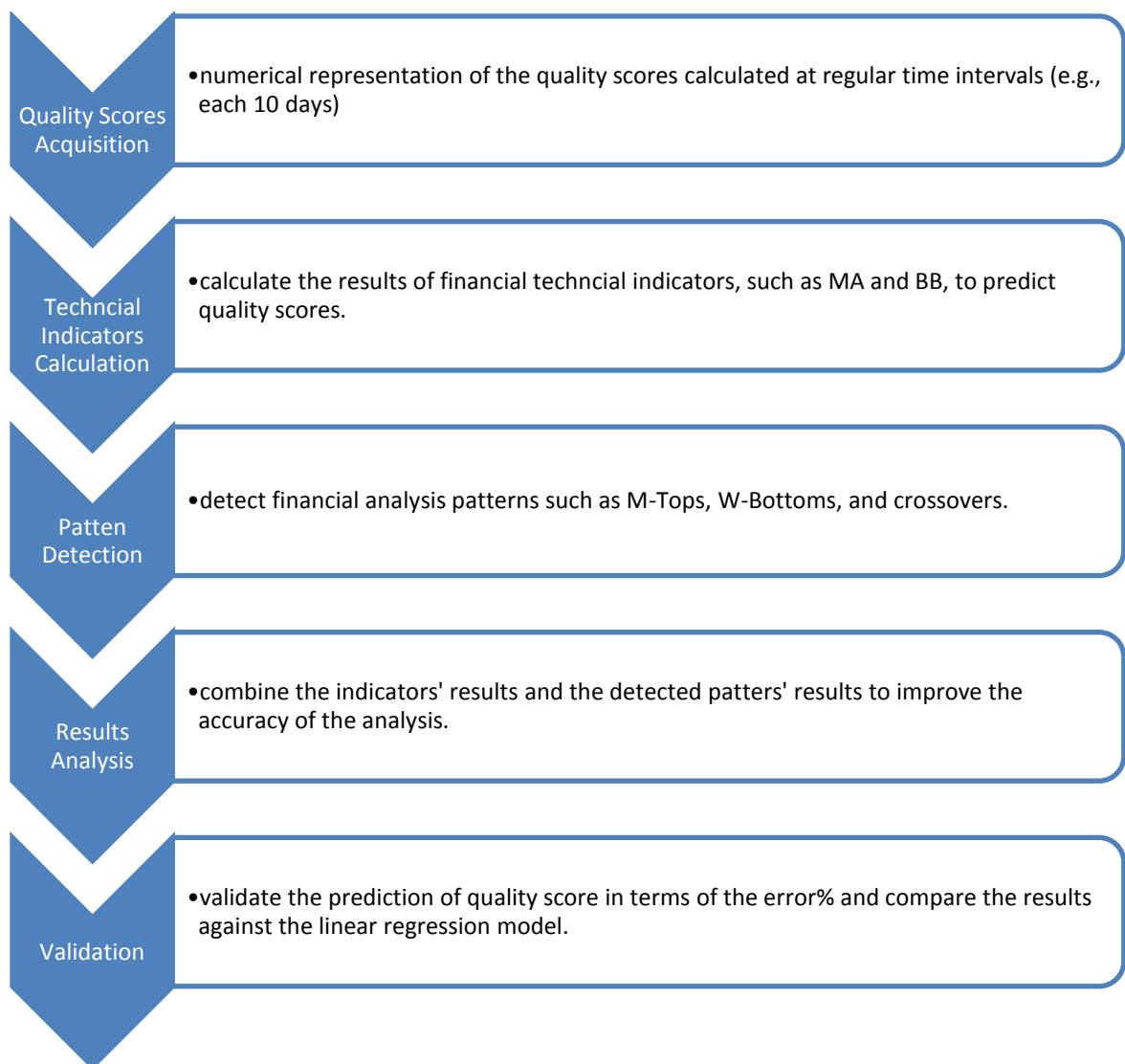


FIGURE 67 FINANCIAL TECHNICAL ANALYSIS PROCESS

## 7.2 CASE STUDY- PREDICTING SOFTWARE EVOLVABILITY QUALITIES

---

The case study applies the financial technical analysis process steps on four open source projects in order to interpret and predict their evolvability quality scores.

### 7.2.1 QUALITY SCORES ACQUISITION

We evaluate the applicability of the proposed financial indicators by applying them to the prediction of quality aspects related to the evolvability of open source projects. For the study, we select four mature, midsize open source projects (Table 21) with software repositories, containing several years of development data history. Data points that correspond to a 10-day time interval are extracted and the evolvability scores are calculated for each data point. The choice for 10 days for our data points is based on existing approaches, e.g., time series analysis for predicting new change requests per KLOC where the data points of choice were bi-weekly, the study covered five years history of three projects (Mozilla, Firefox and JBoss)[110].

Another reason why we selected the data points every 10 days is the need to obtain enough data points to calculate the MA indicator; for example, you need a minimum of 50 snapshots to compute the first data point for a 50 MA. Three of the projects chosen as part of this case study (i.e., Appfuse<sup>82</sup>, Pulse<sup>84</sup>, and Beehive<sup>81</sup>) are from the same domain (open source, Java-based, web programming tools). They all date back to 2005-2006 and provide us therefore with more than 100 data points to analyze. We also chose ArgoUML<sup>85</sup>, an open source, Java-based project that addresses a different domain (i.e., modeling) and provides more data points for interpretation. Below is a more detailed description of each project:

**Apache Beehive**<sup>81</sup> builds an object model on J2EE and struts for easier web programming. The project was introduced in 2005 as part of the Apache software foundation. In Jan 2010, the project retirement was announced due to community inactivity.

---

<sup>81</sup> <http://beehive.apache.org/>

**Appfuse**<sup>82</sup> is a Java based web programming tool. We selected Appfuse since it has a mature, well-established source code. Despite its small community size (18 contributors with a single active<sup>83</sup> member), Appfuse is actively evolving. Our goal is to evaluate how this property would affect its evolution pattern.

**Pulse**<sup>84</sup> project provides an extensible portal for building J2EE web application. The project has a total of 14 members but less than five have actively contributed.

**ArgoUML**<sup>85</sup> is a CASE tool for drawing UML diagrams. The project has been actively developed as part of the tigris.org open source community since 1998. To date, 51 members have contributed into ArgoUML code but with an average of 6 active committers at a time.

**TABLE 21 M-TOP PATTERN APPLICABILITY (ACROSS ALL PROJECTS)**

	<b>Beehive</b> <sup>86</sup>	<b>Appfuse</b> <sup>87</sup>	<b>Pulse</b> <sup>88</sup>	<b>ArgoUML</b> <sup>89</sup>
<b>Data ranges considered</b>	2005 -2010	2006 -2012	2005 - 2012	1998 - 2012
<b>Versions analyzed</b>	1.0.1-1.0	1.5-2.1	0.4-1.0	0.10.1-0.34
<b>Project status</b>	Retired	Active	Active	Active
<b>Domain</b>	J2EE	J2EE	J2EE	Modeling
<b># of snapshots (data points)</b>	113	288	137	557
<b># of revisions</b>	1154	3336	2722	17712
<b># of developers</b>	11	18	14	51
<b># of files</b>	12101	5012	7532	15899

For the assessment scores, we consider evolvability quality scores for open source projects (e.g., evolution of commit activity or code style errors). These scores were obtained for every 10 days

---

<sup>82</sup> <http://appfuse.org/display/APF>

<sup>83</sup> active here refers to a contributor who committed more than the average number of total commits based on the version control system

<sup>84</sup> <http://sourceforge.net/projects/pulse-java/>

<sup>85</sup> <http://argouml.tigris.org/>

<sup>86</sup> <http://beehive.apache.org/>

<sup>87</sup> <http://appfuse.org/display/APF>

<sup>88</sup> <http://sourceforge.net/projects/Pulse-java/>

<sup>89</sup> <http://argouml.tigris.org/>

through OntEQAM [17], our evolvability assessment quality model; and our fuzzified, quality assessment process described in the previous chapter.

## 7.2.2 FINANCIAL TECHNICAL INDICATORS CALCULATION

In this step, for each project we compute the **technical indicators** 20-MA, 50-MA, UBB, LBB, as well as the support and resistance lines based for the input quality scores acquired in the previous step.

1. MA calculation  $MA = \left( \sum_{j=1}^n Y_j \right) / n$  where n is the number of data points; in our case study we use 20 for shorter- and 50 for longer-term MA. Y is the value of each data point; in our case study Y represents the evolvability quality score value obtained using OntEQAM (Chapter 4) and using our fuzzified quality assessment process (Chapter 5).
2. Upper/Middle/Lower Bolinger Bands (UBB/MBB/LBB): are calculated using the following formulas where DP MA is the 20-MA values and STDEV is the standard deviation.

$$MBB = 20 - DP MA$$

$$UBB = 20 - DP MA + [STDDEV(20 - DP score) * 2]$$

$$LBB = 20 - DP MA - [STDDEV(20 - DP score) * 2]$$

3. Support and Resistance lines: are computed based on the pivot point value (PP) equation where the scores are calculated relative to the snapshot of time at which the score is considered:

$$PP = (Max Score + Min Score + Current Score) / 3$$

$$Resistance Line = (2 * pp) - Min Score$$

$$Support Line = (2 * pp) - Max Score$$

Details about these indicators are described in section 6.2.

## 7.2.3 PATTERN DETECTION

As defined in the previous chapter, we detect M-Tops/W-Bottom, bullish/bearish crossover, and support/resistance lines breakout patterns. Details about the patterns definition and calculation are discussed in section 6.2. The pattern detection process is a semi-manual process, and obtained quality scores for the project being analyzed are exported to an Excel sheet. With some of the

equations are provided directly within Excel (e.g., MA and linear regression) or manually entered and copied over to other projects.

1. Bullish/Bearish MA crossover patterns detection: are detected whenever the short/long-term MA trend lines cross each other. A bearish crossover is detected when the longer-term MA crosses over the shorter MA indicators. In the stock market, the bearish crossover is also called a dead crossover and indicates a sell signal because whenever a bearish crossover happens, the future stock price is expected to go down (downtrend). The opposite is the bullish crossover, which is detected when the shorter-term MA crosses over the longer-term MA indicator. Bullish crossover pattern is also called a golden pattern because it indicates a buy signal and an expected uptrend for the future prices. The mapping of the pattern interpretation in software quality is described below in the Table 22.
2. Support/Resistance lines breakout patterns detection: The breakout patterns are based on the support/resistance lines calculations. In the stock market, these lines indicate the strength of the price trend (in our case, the evolvability quality trend). In the stock market, when the stock price crosses above the resistance line indicator, a resistance breakout pattern is detected and indicates a sell signal because the price should subsequently fall. The support line breakout pattern happens when the stock price crosses below the support line indicator. It indicates a buy signal and the price is expected to subsequently rise because of the increase in demand. The mapping of the pattern interpretation in software quality is described below in the Table 22.
3. W-Bottom/M-top pattern detection: To detect these two patterns, we derived several pattern detection rules. These rules address issues related to pattern granularity, outliers, and detection of different pattern evolution levels. Figure 68 describes the M-Top and W-Bottom pattern points that are observed when the detection process takes place.

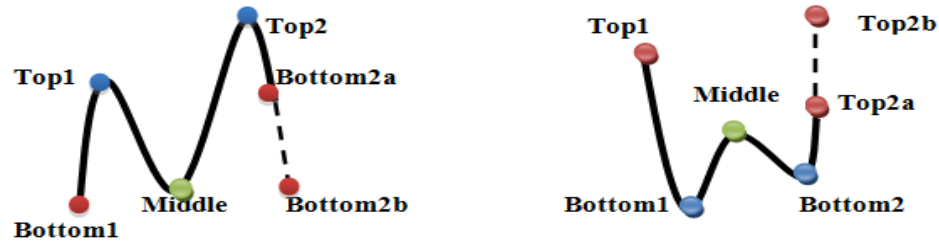


FIGURE 68 M-TOP & W-BOTTOM PATTERNS

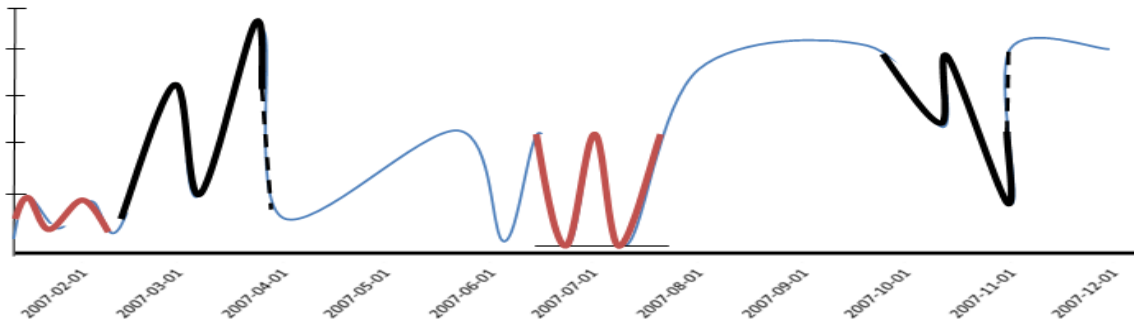



FIGURE 69 SAMPLE PATTERN DETECTION (BLACK IS A PATTERN AND RED IS NOT A PATTERN)

- X-Axis rule:** For the detection of the initial core patterns, we consider a chart scale that covers 3 years of a project's lifetime and data points for every 10 days. We expect that the initial core patterns occur over 4 to 6 data points, based on [141] work which states that the closer the tops/bottoms the less valid the pattern (the duration between the two tops/bottoms should be at least a month, which indicates that the root pattern would span for more than a month<sup>90</sup>. In our case study, we considered a 40- to 60-day range). We chose 4-6 data points based on the stock market with the evolution of these core patterns adding at least one data point for each evolution level. For example, in Figure 69, the first red M-Top-like pattern is not detected because it does not span more than one data point (a month in our example here).
- Y-Axis rule:** To reduce noise in the data and the potential false positives that may result, we only consider that scores are part of the pattern if they reflect at least a 15% change in score value [141].

<sup>90</sup> [http://www.trending123.com/patterns/double\\_bottom.html](http://www.trending123.com/patterns/double_bottom.html)



- **Pattern height rule:** The two tops of an M-Top and the two bottoms of a W-Bottom pattern cannot be at the same level in order to be an actual instance of these patterns. According to [141], there should be at least a 15% incline between the two tops/bottoms. The second W-Bottom-like pattern in Figure 69 violates this rule, which is why it is not detected.
- **Noise rule:** If as part of a pattern evolution, outlier data points occur (e.g., ) we ignore these if (a) they are only a single data point and (b) the outlier has no lasting affected on the general pattern trend.

We performed a two-step analysis process. First we detect the occurrence of root patterns (e.g., W1, M2) and then determined if the pattern continued evolving (e.g., W1 evolves to W2, W3, and then W8); refer to Figure 70 for W-Bottom and M-Top root patterns and their associated evolution patterns.

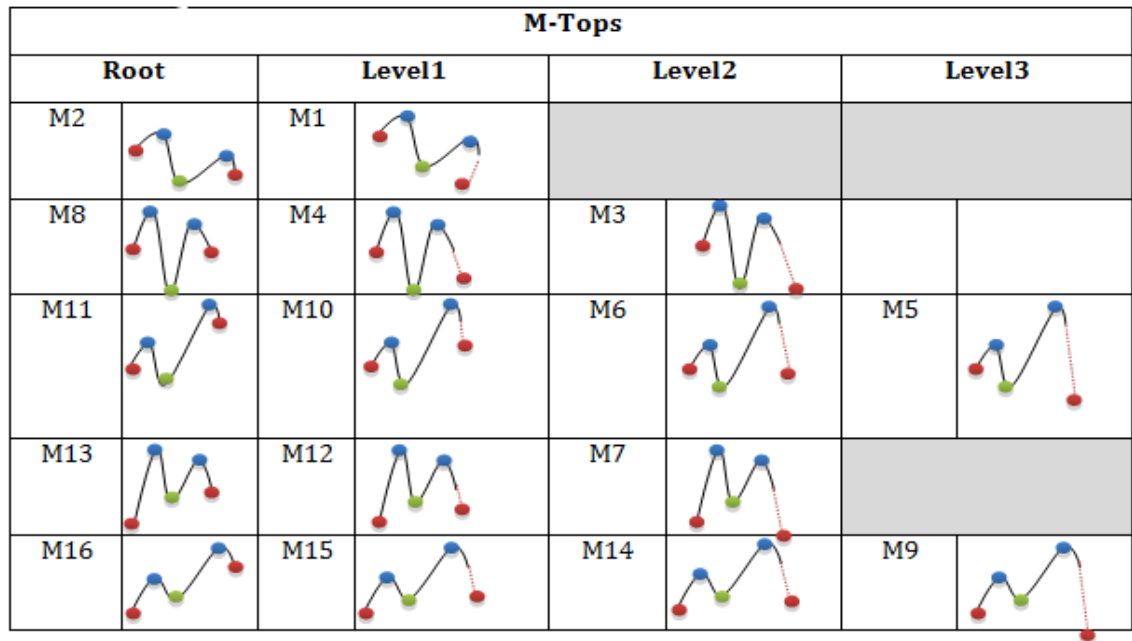
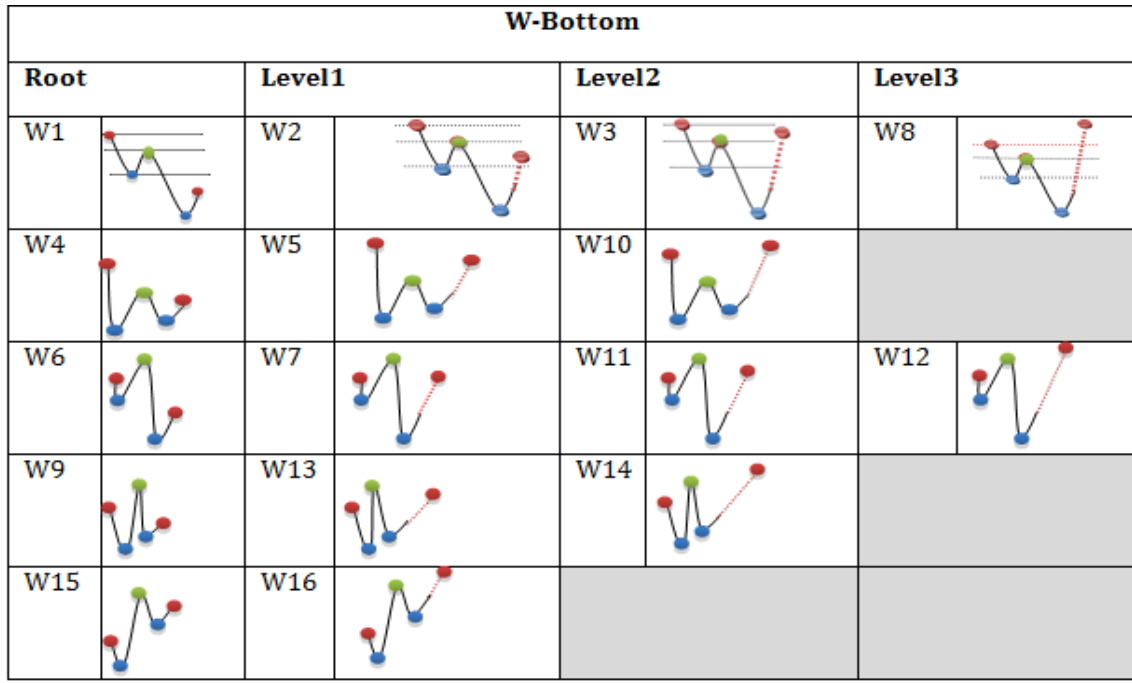


FIGURE 70 W-BOTTOM AND M-TOP ROOT AND EVOLUTION PATTERNS

The M-Top and W-Bottom pattern detection approach is semi-manual, whereby the UBB/MBB/LBB and all data points are automatically calculated and exported in excel format; in addition the manual process includes creating the charts and detecting the root and evolution patterns Figure 70.

Secondly, once a pattern is confirmed through the evolution level, we evaluate the short-term trend in evolvability scores to validate if the pattern matches the actual evolution trend. The mapping between the stock market patterns analysis and the software evolvability score analysis used to perform this validation step is summarized in Table 22:

**TABLE 22 THE MAPPING BETWEEN STOCK PRICE AND EVOLVABILITY SCORE PATTERN INTERPRETATION**

<b>Pattern</b>	<b>Detected when</b>	<b>Stock market interpretation</b>	<b>Evolvability quality<sup>91</sup> score interpretation</b>
<b>Bearish cross-over</b>	Longer-term MA crosses over/above the shorter-term MA.	Declining price trend => sell signal	Declining quality -> Avoid or improve
<b>Bullish crossover</b>	Shorter-term MA crosses over/above the longer-term MA.	Buy signal & price up-trend	Positive signal -> Quality will improve
<b>Resistance breakout</b>	Quality score crosses above the resistance line	Potentially long term buy signal (till new resistance to sell)	Positive signal -> consider reuse -> quality continue improving
<b>Support breakout</b>	Quality score crosses below the support line	Potentially long term sell signal (till new support to buy)	Quality is declining => Avoid adoption
<b>M-Tops</b>	Figure 70	Bearish signal & price will drop	Warning -> quality will decline
<b>W-Bottoms</b>	Figure 70	Bullish signal & price will rise	Positive signal -> quality will improve

#### 7.2.4 RESULT ANALYSIS

Here we combine the detected pattern to interpret and predict the quality score trend. We start with W-Bottom and M-Top then use the MA crossovers and support/resistance breakout patterns to confirm our findings. The approach outlined below is used to improve the accuracy of the interpretation:

---

<sup>91</sup> Quality here does not necessarily mean the overall project quality. It is the evolution quality of the concept used in the analysis and prediction such as measure, attribute, or dimension.

### **Analysing W-Bottom and M-Top Patterns**

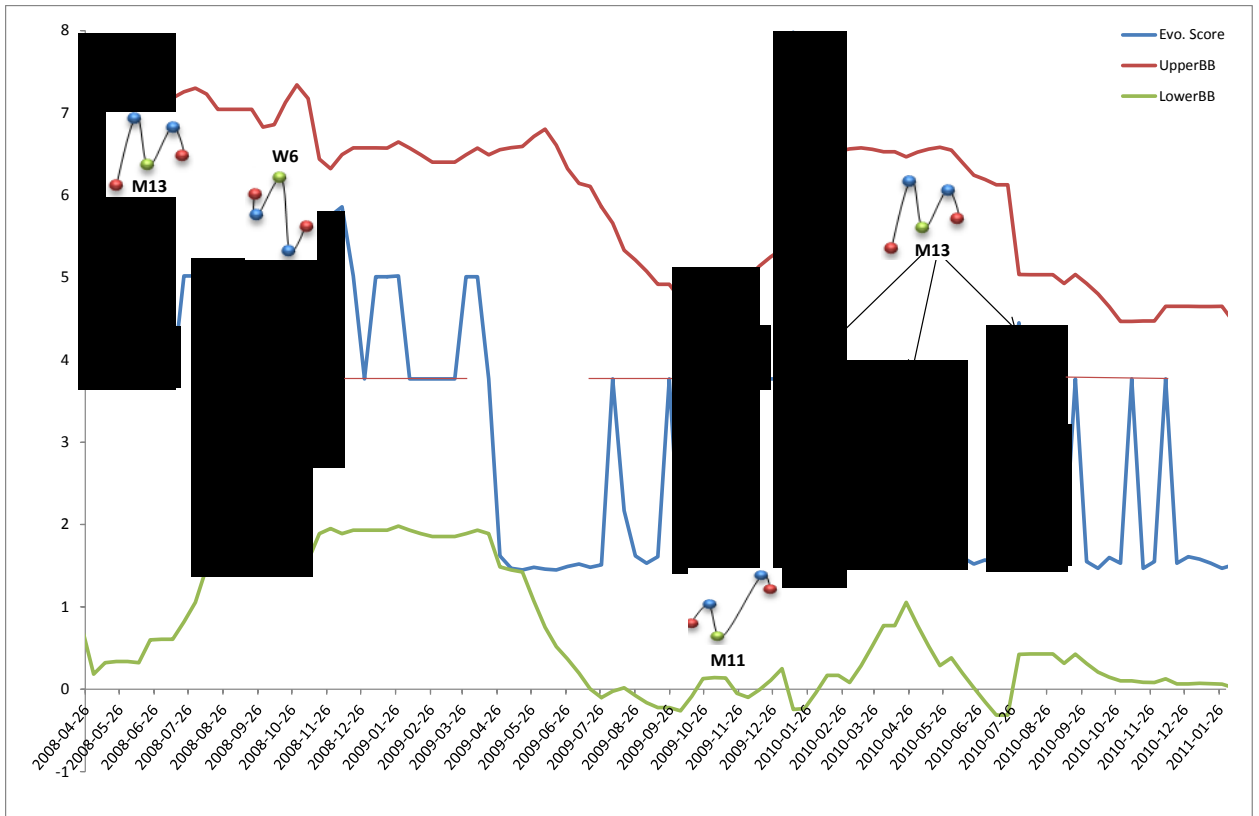
We considered a pattern occurrence to be a true positive (TP) if a pattern evolves (based on pattern detection rules) to a stronger version of its root W-Bottom/M-Top pattern; otherwise the pattern is considered to be a false positive (FP).

Precision ( $\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$ ) captures the likelihood that we observed a pattern evolve from the root level to a particular evolution level.

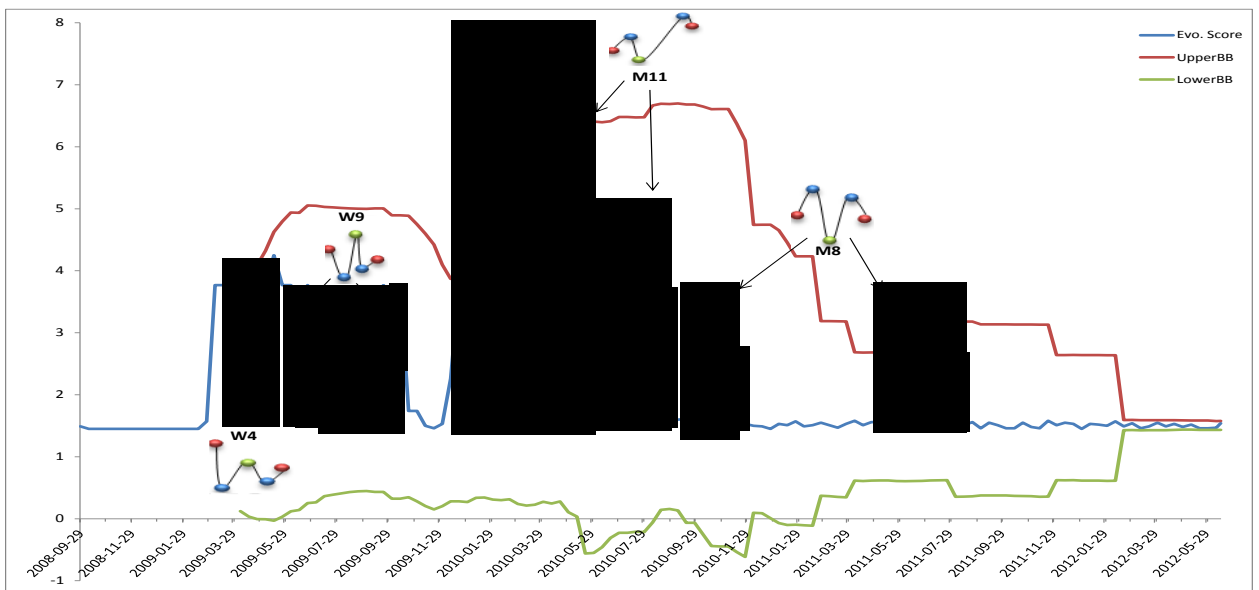
In our analysis, we first determined the frequency of the root pattern and if the patterns actually evolve into stronger evolution levels. The longer a pattern follows its evolvability path, the stronger the evidence that the root pattern can actually predict a short-term trend.

For the four FLOSS projects, we manually annotated the W-Bottom and M-Top patterns. Figure 71 and Figure 72 show M-Tops and W-Bottoms for part of ArgoUML history (2009-2011) and Pulse (2008-2012) lifetime. The X-Axis represents the data points and the Y-Axis represents the evolvability score values.

The evolvability score is obtained in 10-day snapshots using the OntEQAM quality assessment model and our fuzzy assessment process; Figure 71 and Figure 72 show the evolution in number of commits made on the version control system for each of the projects. We highlighted the patterns and mapped them to their equivalent root pattern (the solid black line); then we observed their evolution (black dotted line). Some patterns evolve to their strongest form and some do not (e.g., the last M13 did not evolve while W6 evolved to its strongest form W12).



**FIGURE 71 ARGOUML EVOLUTION PATTERNS 2008-2011**



**FIGURE 72 PULSE- PATTERNS ANNOTATION**

The results of this analysis applied to the four open source projects considered are shown for both M-Top and W-Bottom pattern types in Table 23 and Table 24.

**TABLE 23 W-BOTTOM PATTERN APPLICABILITY (ACROSS ALL PROJECTS)**

Root				Level1				Level2				Level3
Pattern	TP	FP	Precision	Pattern	TP	FP	Precision	Pattern	TP	FP	Precision	Pattern
W1	0	0	NA	W2	0	0	NA	W3	0	0	NA	W8
W4	2	2	50%	W5	0	2	0%	W10				
W6	7	0	100%	W7	7	0	100%	W11	2	5	28%	W12
W9	3	1	75%	W13	2	1	66%	W14				
W15	0	0	NA	W16	0	0						

**TABLE 24 M-TOP PATTERN APPLICABILITY (ACROSS ALL PROJECTS)**

Root				Level1				Level2				Level3
Pattern	TP	FP	Precision	Pattern	TP	FP	Precision	Pattern	TP	FP	Precision	Pattern
M2	2	0	100%	M1	0	0						
M8	15	0	100%	M4	4	11	26%	M3				
M11	14	1	93%	M10	13	1	92%	M6	10	3	76%	M5
M13	10	1	90%	M12	3	7	30%	M7				
M16	3	0	100%	M15	1	2	33%	M14	1	0	100%	M9

**Observations:**

1. We observed that the root patterns follow their evolution patterns for up to 100% of the time in Level 2 and Level 3 and therefore can predict short/midterm trends. This means that the M-Top/W-Bottom patterns and price evolution trends as used in the financial technical analysis domain are also applicable to the software evolvability quality domain. Once a pattern has been detected, the behavior that has been observed and associated in the financial domain with these patterns can also be re-applied to the software domain, as described in Table 22 (e.g., short-term price fall/rise).
2. Similar to the financial markets, W and M-patterns seem to perform better for short-term trend prediction. For example, in Table 24 when M11 root pattern is detected in 14 TP occurrences, the short-term prediction is that the quality score value will drop more to form M10, which was true

in our case study with 92% precision. M10 pattern is predicted to drop even further to form M6, and that prediction was 76% precise. This observation indicates that if the user aims for a high precision rate (> 90%) then the applicability of the M-Top/W-Bottom patterns is better limited to the short term evolution trends (mostly up to level 1).

3. The precision rates are usually lower for stronger evolution patterns. We believe that this is related to the fact that stronger evolution patterns occurrences are less frequent. One more factor to consider when calculating the precision rate is the human error because of the semi-manual pattern detection/annotation part of the case study.
4. Some root patterns (e.g., M11 and W6) play a more important role on a pattern's ability to predict short-term assessment scores than others. For example, the M11 root pattern outperformed the M8 pattern and W6 outperformed W4 and W1 in our studies. This indicates that the trustworthiness of the M11 and W6 is higher (within the scope of this case study).

### **Analysing MA Crossover and Support/Resistance Lines Breakout Patterns**

We also evaluated the following patterns on our dataset of four open source projects: Bullish crossover, Bearish crossover, Resistance breakout, and Support breakout. For our study, we used 20 and 50 data points for Moving Averages. The evolvability scores are based on the evolvability scores for the number of commits made by each project, extracted from their version control system. These assessment scores are calculated every 10 days using the OntEQAM quality assessment model and our fuzzy assessment process (the results used for analyzing M-Top and W-Bottom above). Table 25 provides a summary of our analysis for the MA-crossover patterns and resistance/support lines breakouts for the four open source projects in our dataset. We considered a pattern occurrence to be a true positive (TP) if:

- A **bullish crossover** is followed by an uptrend that lasted more than 4 weeks. If it lasted less than 4 weeks, it is a FP (not significant enough of a post-pattern effect to make a prediction).
- A **bearish crossover** is followed by a downtrend that lasted more than 4 weeks, and FP if less.

- A **resistance line breakout** continues upward for more than 2 weeks, and FP if less.
- A **support line breakout** continues downward for more than 2 weeks, and FP is less.

Otherwise, the pattern is a false positive (FP). The Precision is again =  $TP / (TP+FP)$ .

We chose a 4-week duration for the crossover patterns and a 2-week duration for the breakout patterns because crossover patterns are calculated using MA lines and the MA is based on the past 20-50 historical data points. This historical duration is long enough to preserve the post-pattern effect for a longer duration (4 weeks in our example). While the breakout patterns are based on the evolvability score value breaking the support/resistance lines. The evolvability score is highly volatile, which could be due to two factors: first, the selected measure itself (e.g., the number of commits over time in this example) and second performing the assessment in such a low granularity (10-day snapshots in this example). Hence, the effect of the post-breakout pattern does not last long (if the breakout lasts for up to 4 weeks then it is a sign of pattern reversal, i.e., if the score breaks the resistance line and lasts longer than 4 weeks then we expect the resistance line to become the new support line). Figure 73 and Figure 74 show the crossover and breakout patterns for Pulse and ArgoUML. More details about our observations for these figures are provided below.

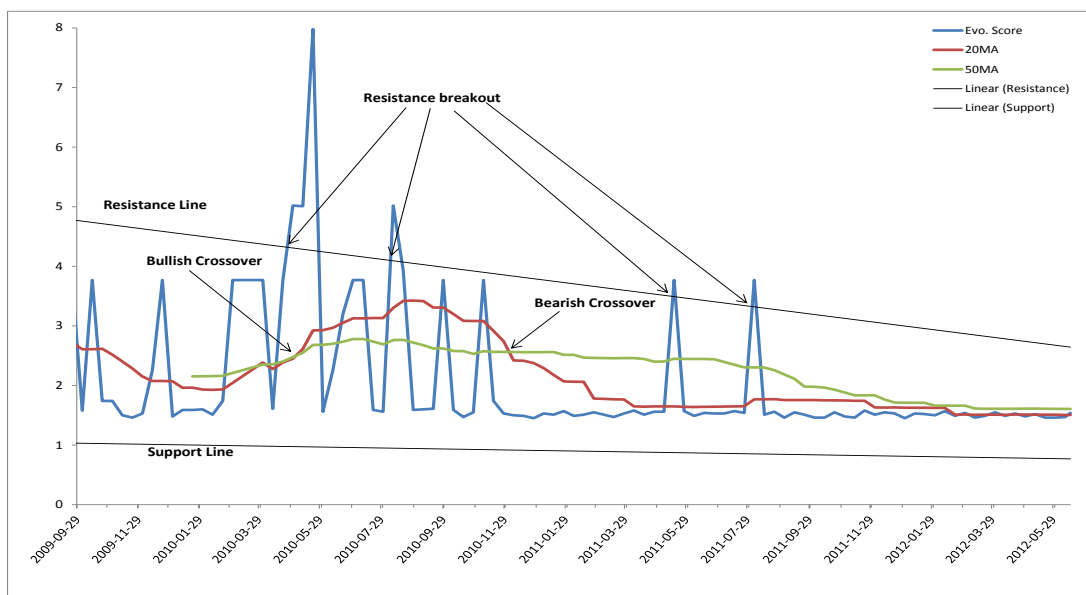
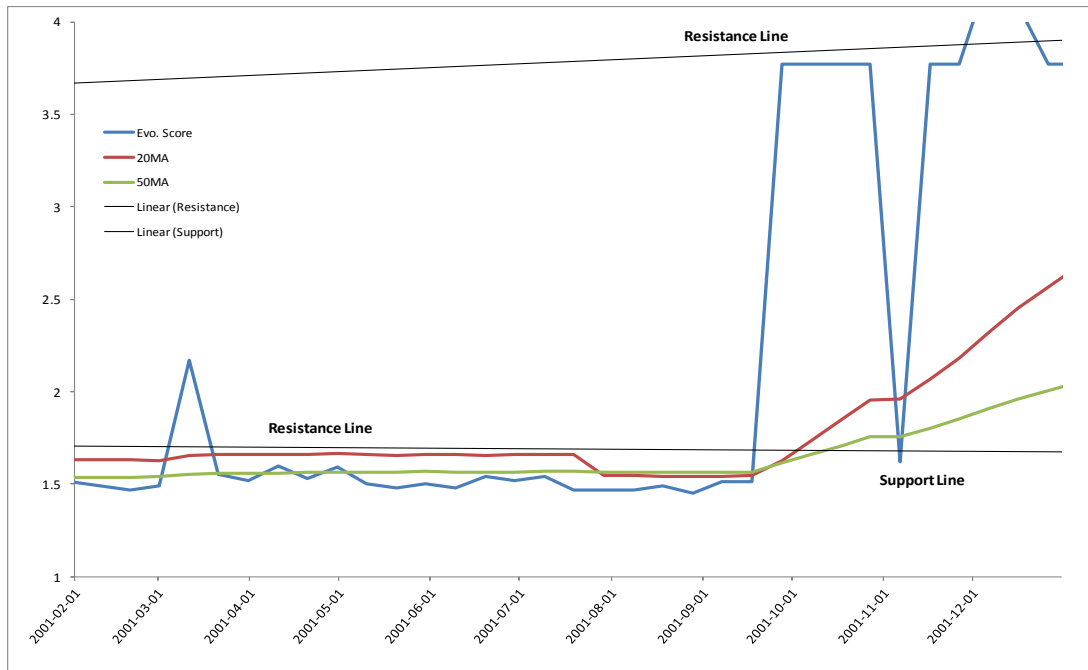


FIGURE 73 PULSE CROSSOVER AND BREAKOUT PATTERNS





**FIGURE 74 ARGOUML- RESISTANCE BREAKOUT PATTERN /BULLISH CROSSOVER CAUSES A PATTERN REVERSAL**

**Observations:**

1. The detected **bullish crossover** is followed by an uptrend (reflecting improvement in the quality score), similar to the occurrences of the pattern in the stock market financial analysis (Figure 73). In this case the uptrend is strong enough to cause the score to breakout the resistance line and lasted > 2 weeks (yet not long enough to cause a pattern reversal).
2. The detected **bearish crossover** is followed by a downtrend (a decline in the evolvability quality score), which again confirms with the stock market financial analysis (Figure 73).
3. The **resistance line breakout** is confirmed if the uptrend breakout duration lasts more than 2 weeks, which means there is a possibility of trend reversal (the evolution will continue upward above the resistance line then the resistance line will become the new support). Figure 74 shows this case in ArgoUML around the end of 2001. In Figure 74 we see that the evolvability score is below the first resistance line until the end of Sept 2001, at which point the score starts improving upward to breakout the resistance line and continues for around two months, at

which point the score trend builds a new resistance line and the old resistance becomes the new support.

4. The **support line breakout** is confirmed in the same way as the resistance line breakout but in a downtrend fashion rather than an uptrend. In Pulse (Figure 73), the evolvability score did not breakout the support line at any point of its lifetime. This indicates that Pulse preserved a stable score range (not necessarily a good score range) over time and the score did not drop strongly enough to break the support line level. Figure 73, also shows a support breakout for ArgoUML t around mid-November 2001. However, this breakout did not last more than 2 weeks and therefore is considered a False Positive(FP); and its post-pattern effect (downtrend for the support line breakout) will have no lasting effect. This decision is confirmed in this example, since the post-pattern downtrend effect did not occur and the score continued instead improving.
5. For our dataset, both crossover patterns outperformed the resistance and support lines breakout patterns (Table 25). The observation is expected because, as explained earlier, the crossover patterns are stronger as they consider an average of long historical data points (20-50 data points in our case study), while the breakout patterns are based on the volatile evolvability score value.

Table 25 shows the TP, FP, and precision rate of the pattern detection for all the projects considered in our case study.

**TABLE 25 QUANTITATIVE ANALYSIS OF MA-CROSSOVER AND SUPPORT/RESISTANCE LINES**

<b>Pattern</b>	<b>TP</b>	<b>FP</b>	<b>Precision</b>
<b>Bullish crossover</b>	10	6	62%
<b>Bearish crossover</b>	14	4	77%
<b>Resistance breakout</b>	29	27	51%
<b>Support breakout</b>	0	11	0%

A more detailed analysis on a per project basis (Table 26) shows a significant variation among projects not only in terms of pattern frequency but also in terms of their ability to predict.

**TABLE 26 QUANTITATIVE ANALYSIS OF MA-CROSSOVERS AND SUPPORT/RESISTANCE PER INDIVIDUAL PROJECTS**

<b>Project</b>	<b>Overall precision</b>	<b>Bullish crossover precision</b>	<b>Bearish crossover precision</b>	<b>Resistance breakout precision</b>	<b>Support breakout precision</b>
<b>ArgoUML</b>	59%	77%	80%	68%	0%
<b>Beehive</b>	75%	NA	100%	33%	NA
<b>Appfuse</b>	58%	33%	66%	35%	NA
<b>Pulse</b>	83%	100%	100%	25%	NA

For higher prediction accuracy, we correlate the MA crossover pattern with the M-Top/W-Bottom patterns to see if they confirm the same trend.

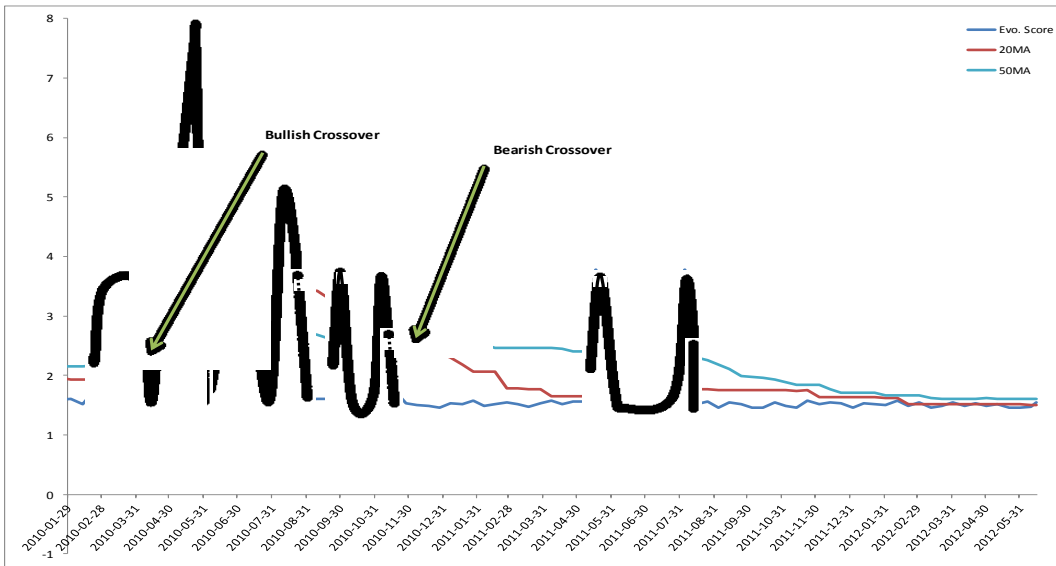
Figure 75 shows the Pulse project evolvability score over time with annotated patterns. Around March 2010, we detected the first M-Top pattern (M11). We observe that the second top of M11 goes upward from score value 2 to 8, which is the highest score that Pulse ever achieved. That score uptrend affected the short-term MA (20-MA), whereby it moved in an uptrend fashion until the end of August 2010. This is a significant, long-lasting post-pattern effect because the change in score should be strong enough to cause the last 20 historical data points to move upward.

Around the same time (March 2010), there are two crossover patterns; the first is a bullish crossover whereby the short-term MA crossed over the longer-term MA. The interpretation of this crossover indicates an expected rise in the short-term future scores (expected based on Table 22). Both M-Top and MA-Crossover patterns confirmed the uptrend prediction of the evolvability score.

The opposite case happens around mid-November 2010, at which point the MA shows a bearish signal accompanied by an M-Top (M8) that reaches score 4 (half the first M-Top).

We conclude that the stronger the M-Top pattern, the more likely a crossover is to happen; M11 caused a bullish crossover (March 2010) while the two weaker M8 patterns that happened after (between May and November 2010) were needed to initiate a bearish crossover. Strong and weak

here refer to the M-Top height (the higher the stronger, as the height here corresponds to the quality score value and the strongest value achieved is 8 out of 10 for Pulse). Based on the financial technical analysis, a bullish crossover is followed by an uptrend (higher score compared to the overall score range) and a bearish crossover is followed by a downtrend or lower scores; we can consider these as early signals to a rise or drop in the upcoming score.



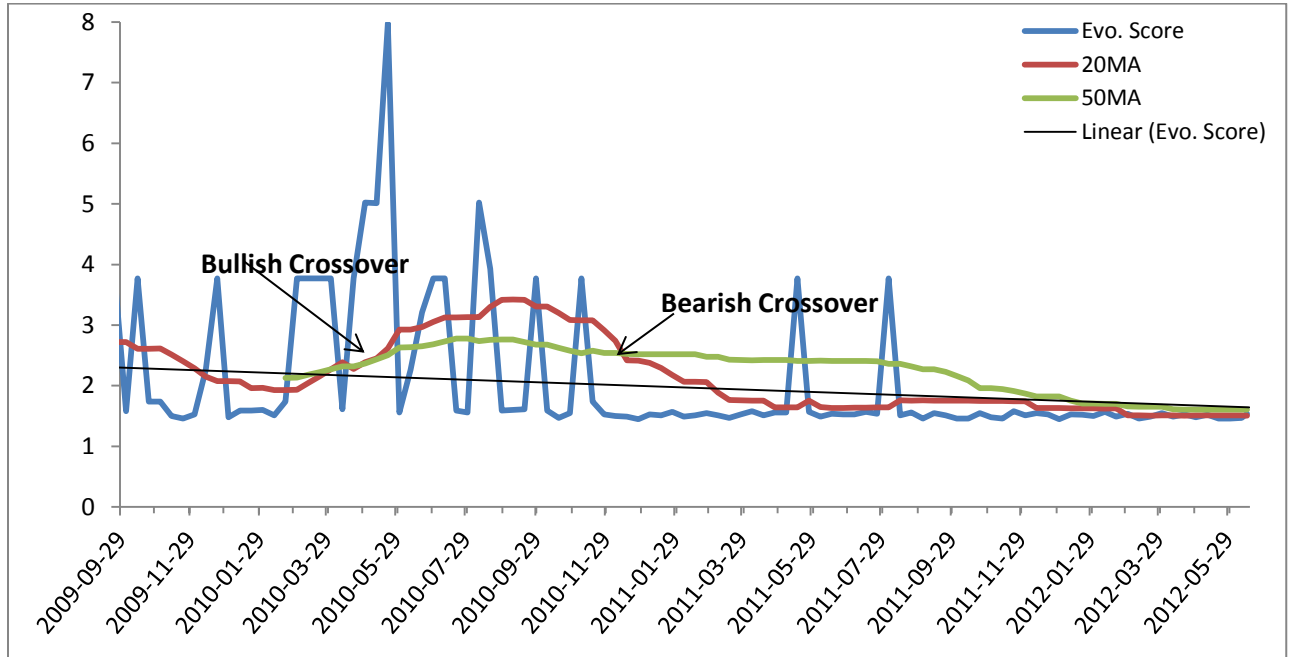
**FIGURE 75 PULSE- MA CROSSOVER PATTERN AND M-TOPS PATTERN CORRELATION**

### 7.2.5 VALIDATION

In this section, we validate the MA trend line and crossover patterns against linear regression as one of the most popular and most used indicator for trend analysis and prediction in software domain [105]. Figure 76 below shows the linear regression applied to the same dataset on which we performed our experimental analysis. Figure 76 also shows the short-term (20-MA) and longer-term (50-MA) trend lines and crossover patterns against the evolvability score.

The MA trend lines are more sensitive to the fluctuations in evolvability score over time; for example, throughout the year 2010 there has been a significant improvement in the evolvability score, which is obvious from both 20-MA uptrends that started around March 2010. This uptrend was confirmed by the bullish crossover pattern. This evolution trend gives the user an early sign that the evolvability quality is improving significantly compared to the historical evolvability scores.

However, by only considering the linear regression trend line for the year 2010, the assumption would be that the evolvability score is slowly dropping over time in a linear fashion. However, the real evolution in 2010 is improving.



**FIGURE 76 20-MA, 50-MA AND MA CROSSOVER PATTERNS VERSUS LINEAR REGRESSION**

To assess the performance of our prediction, we reuse the Percent Prediction Error (PPE) formula as defined in [110] where an agreed on error of 25% is considered acceptable for 75% of the dataset [110].

$$PPE = \left( \frac{ABS(measured\ score - predicted\ score)}{measured\ score} \right) * 100$$

In our case study, the actual value represents the evolvability score value and the prediction is based on the 20-MA.

Table 27 summarizes the actual evolvability score for the last three data points per project. The table shows side-to-side comparison between the predictions applied using linear regression (LR) versus the 20-MA used in our approach.

Linear regression is calculated using  $LR = TREND(actual\ y's, actual\ x's, const)$

With the  $y$  value representing the actual quality scores for all data points before the next data point to be predicted for the formula  $y = mx + b$ . *Actual x's* represent an optional set of the snapshots (DPs), for which the score is calculated using the same formula  $y = mx + b$ . *Const* is an optional value that is set to zero if  $b$  is zero. The slope ( $m$ ) is calculated using the difference between two scores over time (e.g.,  $(x_1, y_1)$  and  $(x_2, y_2)$  where  $x$  is the score and  $y$  is the time then  $m = (y_2 - y_1) / (x_2 - x_1)$ ).

Note that it is by coincidence in our evaluation that the scores for the data points in the four projects shown below (Table 27) were close.

**TABLE 27 ACTUAL VERSUS PREDICTED SCORE FOR THE LAST THREE DATA POINTS USING 20-MA AND LR**

Project	1 <sup>st</sup> actual	1 <sup>st</sup> predicted		2 <sup>nd</sup> actual	2 <sup>nd</sup> predicted		3 <sup>rd</sup> actual	3 <sup>rd</sup> predicted	
		20-MA	LR		20-MA	LR		20-MA	LR
<b>ArgoUML</b>	1.45	1.75	2.30	1.48	1.76	2.31	1.45	1.63	2.31
<b>Beehive</b>	1.45	1.45	3.79	1.45	1.45	3.78	1.45	1.45	3.77
<b>Appfuse</b>	1.45	1.46	2.98	1.45	1.46	2.98	1.45	1.46	2.98
<b>Pulse</b>	1.46	1.51	1.58	1.47	1.51	1.58	1.54	1.50	1.57

Table 28 shows the total number of data points for each of the four open source projects we used in this chapter as part of our case study. For each project, we calculate the 20-MA and associated the number of data points that has a prediction error percentage (PPE%)  $\leq 25$  (PPE%  $\leq 25\%$  is shown to be an acceptable error rate for 70%-75% of the dataset [110]). This is calculated as:

$$PPE\% \text{ relative to total DPs} = (\#PPE \leq 25\%) * 100 / \#of \text{ DPs used in 20-MA}$$

Table 28 then shows the percentage of the predicted data points that met the 25% PPE out of the total number of that data points used per project. Then, we predict the last three data points in our dataset (e.g., in ArgoUML, we use 554 DP for training and the last three for testing). The last three columns show the PPE% of the three predicted DP.

**TABLE 28 PERCENT PREDICTION ERROR RELATIVE TO THE TOTAL DATASET PER PROJECT (USING 20-MA)**

<b>Project</b>	<b>#Snapshots (data points)</b>	<b>#PPE &lt;=25%</b>	<b>≈PPE% relative to total DPs</b>	<b>1<sup>st</sup>DP prediction PPE%</b>	<b>2<sup>nd</sup>DP prediction PPE%</b>	<b>3<sup>rd</sup>DP prediction PPE%</b>
<b>ArgoUML</b>	557	334	70%	21%	18%	12%
<b>Beehive</b>	113	86	73%	3%	3%	3%
<b>Appfuse</b>	288	125	50%	0.7%	0.6%	0.6%
<b>Pulse</b>	137	57	50%	3.4%	2.7%	2.2%

In Table 29, we apply the linear regression prediction to the same dataset and calculate the PPE% to validate our prediction against linear regression.

**TABLE 29 PERCENT PREDICTION ERROR RELATIVE TO THE TOTAL DATASET PER PROJECT (USING LR)**

<b>Project</b>	<b>#Snapshots (data points)</b>	<b>#PPE &lt;=25%</b>	<b>≈PPE% relative to total DPs</b>	<b>1<sup>st</sup>DP prediction PPE%</b>	<b>2<sup>nd</sup>DP prediction PPE%</b>	<b>3<sup>rd</sup>DP prediction PPE%</b>
<b>ArgoUML</b>	557	151	30%	58%	56%	59%
<b>Beehive</b>	113	0	0	162%	161%	160%
<b>Appfuse</b>	288	21	8%	105%	105%	105%
<b>Pulse</b>	137	29	25%	73%	72%	65%

Table 28 and Table 29 illustrate that our prediction approach using the 20-MA provides a higher accuracy (lower error rate per dataset) compared to standard linear regression models. We observed that the linear regression approach, the number of data points that did not exceed the target acceptable error rate was less than half that of the MA approach.

Further analyzing the PPE%, the goal is to have PPE% as low as possible (PPE% of zero means that our predicted value equals the actual value). The average PPE% for the three predicted data points using 20MA is 5.7% while the Linear Regression approach provided an average error rate (PPE%) of 98% for the same dataset. In linear regression, PPE% exceeds 100% (the predicted value is more than double the actual) in some cases such as Beehive and Appfuse. This means that the predicted

score value is more than one time higher than the actual value; for example, in Beehive the 3<sup>rd</sup> predicted DP with an error rate of 160% had a predicted score 3.77 while the actual is 1.45. As a result, the predicted value is twice the actual value in some cases where linear regression approach is used.

Based on our interpretation, the following are the main factors that could impact the accuracy of the quality prediction:

- *Volume*: The number of data points (snapshots). The larger (as in ArgoUML) the number of data points considered, the higher the error rate. Unless the project is relatively stable and the scores do not fluctuate over time.
- *Stability*: The stability of the score value over time is dependent on the measure itself, such as the number of bug reports or code size or turnover in community members. The score varies by the project's activity on the selected measure being predicted. In our example, we chose to predict the number of commits to be made in the next 10, 20, and 30 days. The more active the project is, the higher the fluctuation rate of the score (less stability) and hence the higher the prediction error.
- *Prediction model*: Different models, such as moving averages and linear regression, provide different prediction rates. In our case study, we show that moving averages can provide for short-term predictions (20 data points in this example) a lower error rate compared to linear regression.

Our approach provides two major advantages over existing approaches (e.g., linear regression). Firstly, our evaluation shows that our approach can achieve a higher precision (lower error rate) for predicting the short-term trend (next three data points) based on our validation presented above. This finding proves that the financial technical indicators (such as MA) and patterns (such as crossovers, M-Tops, and W-Bottoms) are applicable when interpreting and predicting software evolvability quality.

The second advantage includes the supporting patterns such as the crossovers, M-Tops and W-Bottoms. These patterns (despite being calculated semi-manually) provide a finer grained prediction



of the next quality score. For example, if we detect a root *W*-Bottom pattern such as *W6*, then we can predict that the next score will be higher (based on the next evolution pattern on *W6*) while in other models such as LR, the scores are smoothed in a linear line where rises and drops are unpredictable.

Based on our evaluation, we do recommend using the MA for score prediction when the user is interested in short term prediction and the user is concerned about the precision of the prediction score where the support of other indicators/patterns comes in handy. As part of our study, we predicted the same data set using 50MA to see how a longer-term interval would affect the prediction accuracy. The results for the 50MA are still better than the prediction provided by the LR model (e.g., for ArgouML where PPE% relative to total DPs is approximately 30% using LR against 63% using 50MA). The linear regression model is preferred for longer-term predictions (beyond the 50MA) as it smooths down the evolution trend over time by considering all the historical data points rather than the last 20 or 50 for example.

### 7.3 SUMMARY

---

In this chapter, we studied the applicability of the financial technical analysis used for the stock price evolution forecast to interpret and predict the software evolvability score. Our objective was to address the question: *Can the interpretation and prediction of evolvability trends be supported by the indicators and patterns traditionally used in financial markets?*

To answer this question we applied five different steps 1) define the dataset of quality scores; 2) compute the financial technical indicators (e.g., MA and BB and support/resistance lines); 3) detect the associated patterns (e.g., *M*-Top, *W*-Bottom, MA crossovers and support/resistance line breakouts); 4) map the analysis used in stock market to our evolvability scores; and, finally, 5) validate our approach against existing, popular approach such as the linear regression model.

Our evaluation shows that the use of moving average and the supporting patterns provide lower prediction errors (higher precision) and finer grained prediction compared to a linear prediction line. The answer to our question is yes, since financial indicators such as MA provided an error

prediction rate as low as 5.7% and can therefore, as discussed in [110], be considered a reliable predictor. Patterns such as M-Top and W-Bottoms to predict rises/drops future score followed the expected behaviour as in the stock price analysis by following the trend of the evolution patterns.

## CHAPTER 8: CONCLUSION AND FUTURE WORK

This chapter concludes the presented work in this thesis by revisiting our initial research questions and explaining how they were addressed in the thesis. This chapter also provides an overview of the potential threats to validity. Finally, we discuss future work.

### 8.1 REVISITING OUR RESEARCH QUESTIONS

---

- RQ1: Can a formal, reusable quality assessment metamodel be developed that is readable by both humans and machines?
  - We address this research question by introducing a novel, evolvable, quality assessment metamodel (SE-EQUAM) [120, 12] in Chapter 3. This model is based on an ontological representation to provide a metamodel that is both machine- and human-readable. The SE-EQUAM metamodel can be reused and extended by domain specific quality assessment models regardless of the quality or set of qualities the model is assessing. In Chapter 4, we demonstrated a concrete instance (reuse) of the SE-EQUAM model by introducing OntEQAM [17], an ontological evolvability assessment model. The SE-EQUAM metamodel takes advantage of the integration of semantic reasoning technology, OWL DL, and SPARQL.
- RQ2: How to address uncertainties around the assessment score boundaries?
  - Assessing software qualities involves a diverse set of artifacts. However, the knowledge captured by these artifacts might be either incomplete or missing during extraction or population time. This incomplete or missing knowledge can cause uncertainty about the assessment result/score. In Chapter 5, we show that our quality assessment process is based on a fuzzy logic approach in which uncertainty is addressed by avoiding crisp score boundaries (e.g., a measure value of  $x\%$  or less is considered to be very poor, whereas as a score of  $x\% + 0.1$  is ranked as only poor). We apply the fuzzy logic assessment approach as part of our evolvability assessment

where the assessment score has relationship strength with the range (e.g., 40% poor and 60% average).

- RQ3: Can the interpretation and prediction of evolvability trends be supported by indicators and patterns used traditionally in financial markets?
  - In Chapter 7, we introduced a novel, interdisciplinary research that re-applies the stock market analysis patterns used to interpret and predict stock price evolution on the software evolution [13, 14]. We also compared our prediction against the linear regression prediction model, one of the most popular prediction models. Our validation showed that the use of moving average and the supporting patterns provides lower prediction errors (reached approximately 94% accuracy) and finer grained prediction supported by evolvability patterns, such as M-Top and W-Bottom (compared to a linear prediction line).
- RQ4: How can the formal reusability of the metamodel be validated and the integration of the assessment results be supported?
  - To validate the reusability of our SE-EQUAM [12] semantic, ontological metamodel in Chapter 3, we instantiated/resued SE-EQUAM ontological representation by populating OntEQAM [17], our evolvability quality model in Chapter 4. OntEQAM [17] is extracted from existing models (e.g., ISO/IEC 9126 [15] and QUALOSS [16]). OntEQAM formally reuses SE-EQUAM ontological representation and semantics and assesses evolvability using the fuzzy logic assessment process described in Chapter 5. The ontological model we have for SE-EQUAM and its instance OntEQAM is enriched with knowledge extracted from the assessment results. The knowledge base is used for further analysis and custom queries.

## 8.2 THREAT TO VALIDITY

---

### 8.2.1 DESIGN AND IMPLEMENTATION THREATS

One of the major benefits of our unified quality metamodel SE-EQUAM [12] is the ease of formal (machine-readable) reuse. Reusability in this context requires an ontology expert in order to extend our ontology metamodel to a domain ontology model. In particular, modeling new constraints and relations or inferring knowledge that is not modeled in the metamodel requires expertise in ontology modeling and reasoning. We believe that this threat is not unique to our domain and can be observed in other modeling domains (e.g., software design, database design), where the quality of the final model/design similarly depends mostly on the expert performing the design/modeling step.

Another threat related to SE-EQUAM is the possibility of having design defects. Assessing the ontology design quality is an inherently difficult problem, since what constitutes quality depends on different non-functional requirements (e.g., reuse, usability, extendibility). We partly address this threat by using an ontological reasoner (e.g., Pellet [149]), which is capable of checking the ontology design for syntactical and consistency problems. However, the verification and optimization with respect to other functional and non-functional requirements remain the responsibility of the domain expert.

The possibility exists of a threat caused by implementation defects when instantiating (reusing) our metamodel ontology. We have addressed this potential threat by instantiating a partial evolvability domain ontology model and comparing the results obtained from this model against results published by QUALOSS [91]. For this validation, both models were using the same input and quality model factors, subfactors, and attributes.

Validating the correctness of the newly inserted knowledge, such as adding a new attribute that is not actually an attribute, is another potential threat. This threat can only be partially mitigated by adding rules and constraints against the populated concepts, since much of the interpretation of what constitutes an attribute in an assessment model is subjective to human interpretation and the specific assessment context.

Adopting most of the measures results through using external tools could be a threat for two reasons: first, there is a risk associated with maintaining the tool interface with our implementation if the tool API changes and second there is uncertainty regarding the accuracy of the results calculated. For the first threat, unless the tool modifies the XML generated report structure, there should be no inconsistency with our current implementation. If the XML report structure changes, then an expert should manually update the parser implementation we have for that tool. In addition, we partially addressed the second threat by selecting tools that are widely used by other quality models and researchers in the field, such as SonarSystem<sup>92</sup>.

The case studies described in this thesis are limited in their scope to open source Java project and the results obtained from our case studies might not be applicable to other programming languages or system types.

### 8.2.2 COMPLETENESS THREATS

A potential threat to our approach is whether the set of quality measures we considered in our assessment as part of OntEQAM evaluation (refer to Chapter 4) are sufficient to accurately capture evolvability as a quality factor. We addressed this threat by selecting our evolvability measures from a well-established subset of existing quality models, such as ISO/IEC 9126 [15], QUALOSS [16], and SIG MM<sup>93</sup>. While we only selected a subset of these evolvability attributes, we believe this subset is sufficient to illustrate the applicability of our assessment model. Given the ontological reusability of our design, extending OntEQAM to support other requirements including new measures, attributes or subfactors is a straightforward task

Another threat to validity could be that we did not have sufficient data available to perform an accurate trend analysis (especially for longer interval MAs). We tried to mitigate this threat by limiting our analysis to a maximum of 100 data points, rather than the 200 data point MA typically

---

<sup>92</sup> <http://www.sonarqube.org/>

<sup>93</sup> [http://www.sig.eu/en/Research/690/Maintainability\\_Model.html](http://www.sig.eu/en/Research/690/Maintainability_Model.html)

used in the stock market analysis. Using shorter MAs provides us with sufficient data points to analyze the prediction quality of these indicators.

### 8.2.3 ACCURACY THREATS

Another threat to validity can be the trustworthiness of the OntEQAM evolvability scores used during the analysis. We addressed this threat, by *partially* validating our evolvability scores against existing models that addressed evolvability using a different assessment process (e.g., QUALOSS [16]). This validation is performed over a pre-selected set of common measures. The lack of having a universal definition of what measures to consider when assessing a certain software quality, such as evolvability, makes it difficult, if not impossible, to compare our results to those of other existing quality models.

Another threat is the accuracy of our prediction. We agree that the reliability of MA and its capabilities to predict both long- and short-term evolvability might be limited. Further analysis with respect to the reliability and predictive power are needed. Furthermore, similar to the stock market, software projects have to deal with a magnitude of internal and external factors (e.g., competition and global trends), which are not predictable. We therefore believe that MAs should be used in conjunction with other prediction methods in order to improve the overall reliability and accuracy of the prediction.

For the case study in Chapter 7, we illustrate the applicability of the presented approach using a limited subset of the overall quality assessment model.

## 8.3 FUTURE WORK

---

The work presented in this thesis could be extended in several ways:

- There are several online repositories, such as FlossMetrics [177], Ohloh [178], and SECOLD.org that host thousands of FLOSS projects. Enriching these repositories with evolvability scores using existing models, such as OntEQAM [17], would provide the quality

research community with a reference to compare against in future work and a better validation of the assessment results.

- This thesis provides preliminary evidence that demonstrates that financial technical indicators and their associated patterns are applicable to the interpretation and prediction of evolvability quality scores [13, 14]. This work can be further extended by first automating the pattern detection process (which in turn makes it easier to have a larger dataset of projects) and second looking for new patterns that could be applicable only in the software domain or only for certain software qualities. Furthermore, the prediction approach should be applied on a larger set of quality measures and different assessment levels, such as attributes, factors, or dimensions. Also, as part of the future our work we plan to compare the scores against other prediction more models, such as ARIMA [108,109].
- In this thesis, SE-EQUAM [12] has been instantiated by OntEQAM [12, 17]; more domain models that assess other software qualities, such as security, should be evaluated as part of the future work.
- The measure set that OntEQAM [12, 17] uses can be further extended by including additional evolvability measures related to new knowledge artifacts, such as e-mail or project online forums discussions.
- As part of the thesis several usage scenarios were briefly introduced (i.e., process improvement, advisory systems). As part of the future work actual user studies should be conducted to validate the applicability of the proposed usage scenarios in actual project contexts. .



## REFERENCES

- [1] D. V. Edelstein, "Report on the IEEE STD 1219-1993 standard for software maintenance," *ACM SIGSOFT Software Engineering Notes*, vol. 18, pp. 95, Oct. 1993.
- [2] H. A. Muller, S. R. Tilley, and K. Wong, "Understanding software systems using reverse engineering technology perspectives from the Rigi project," in *Proc. 1993 Conf. Centre for Advanced Studies on Collaborative Research: Software Engineering*, Toronto, Ont., 1993, vol. 1, pp. 217-226.
- [3] D. Takahashi, "Why Vista might be the last of its kind," *The Seattle Times*, December 2006, [Online]. Available: [http://seattletimes.nwsourc.com/html/business/technology/2003460386\\_btview04.html](http://seattletimes.nwsourc.com/html/business/technology/2003460386_btview04.html) [Accessed: July 2013].
- [4] R. Seacord, D. Plakosh, and G. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*. Boston, MA: Addison-Wesley, 2003.
- [5] I. Sommerville, *Software Engineering*. New York, NY: Addison-Wesley, 2007.
- [6] M. V. Zelkowitz, A. C. Shaw, and J. D. Gannon, *Principles of Software Engineering and Design*. Englewood Cliffs, NJ: Prentice-Hall, 1979.
- [7] L. Erlikh, "Leveraging legacy system dollars for E-business," *IT Professional*, vol. 2, pp. 17-23, May/June 2000.
- [8] M. V. Mäntylä, "Empirical software evolvability - code smells and human evaluations," in *2010 IEEE Int. Conf. on Software Maintenance (ICSM)*, Timișoara, Romania, 2010, pp. 1-6.
- [9] M. M. Lehman, "On understanding laws, evolution and conservation in the large program life cycle," *J Syst Software*, vol. 1, pp. 213-221, 1980.
- [10] M. M. Lehman, "Laws of software evolution revisited," *Lect Notes Comput Sc*, vol. 1149, pp. 108-124, 1996.
- [11] M. M. Lehman, "Laws of Program Evolution - Rules and Tools for Programming Management," in *Proc. Infotech State of the Art Conference, Why Software Projects Fail?*, pp. 11/1-11/25, 1978.
- [12] A. Hmood, I. Keivanloo, and J. Rilling, "SE-EQUAM - an evolvable quality metamodel," in *36<sup>th</sup> Computer Software and Applications Conference and Workshops (COMPSAC)*, 2012, pp. 334-339.
- [13] A. Hmood and J. Rilling, "Analyzing and predicting software quality trends using financial patterns," in *7<sup>th</sup> IEEE Int. Workshop on Quality Oriented Reuse of Software (QUORS) - Co-Located with COMPSAC 2013*, 2013, p. 6.
- [14] A. Hmood, M. Erfani, I. Keivanloo, and J. Rilling, "Applying technical stock market indicators to analyze and predict the evolvability of open source projects," in *28<sup>th</sup> Int. Conf. in Software Maintenance (ICSM)*, 2012, pp. 613-616.
- [15] ISO, "ISO/IEC 9126: Software engineering-product quality-part 1: Quality model-part 4: Quality in use metrics." 2001-2004.
- [16] J. Deprez, F. F. Monfils, F. Steels, J. Flamand, and N. Devos, QUALOSS, November 2009 [Online]. Available: <http://www.cetic.be/QualOSS.500> [Accessed: July, 2013].

- [17] A. Hmood, P. Schugerl, J. Rilling, and P. Charland, "OntEQAM: A methodology for assessing evolvability as a quality factor in software ecosystems," in *IEEE Industrial Software Evolution and Maintenance Processes (WISEMP)*, 2010.
- [18] M. Mäntylä, "Empirically discovered evolvability issues and human evaluations," Ph.D. dissertation, Dept. Comp. Sci. and Eng., Helsinki Univ. of Technology, Helsinki, 2009.
- [19] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Boston, MA: Addison-Wesley, 1995.
- [20] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution-the nineties view," in *Proc. 4th Int. Software Metrics Symposium*, 1997, pp. 20-32.
- [21] M. M. Lehman, J. F. Ramil, and G. Kahen, "Evolution as a Noun and Evolution as a Verb," in *Workshop on Software and Organisation Co-evolution (SOCE)*, 2000.
- [22] C. F. F. Kemerer and S. Slaughter, "An empirical approach to studying software evolution," *IEEE Trans. Software Eng.*, vol. 25, pp. 493-509, July/Aug. 1999.
- [23] C. L. Nehaniv "Introduction to software evolvability," in *Proc. 3rd Int. IEEE Workshop on Software Evolvability*, 2007, pp. vi – vii.
- [24] ISO/IEC and IEEE, "Software Engineering - software lifecycle process - maintenance," *Int'l Standard*, vol. 2006, pp. 4, 2006.
- [25] Khanh Hoa Dam, "Supporting software evolution in agent systems," Ph.D. dissertation. School Comp. Sci. and I.T. RMIT Univ., Melbourne, Australia, 2008.
- [26] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev, "Ontological approach for the semantic recovery of traceability links between software artefacts," *IET Software*, vol. 2, pp. 185-203, 2008.
- [27] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target [special issues section]," *Software, IEEE*, vol. 13, pp. 12-21, Jan. 1996.
- [28] A. C. Gillies, *Software Quality: Theory and Management*, London: Chapman and Hall, 1992.
- [29] D. Hoyle, *ISO 9000 Quality Systems Handbook (5thed.)*. Jordan Hill, Oxford: Butterworth-Heinemann, 2005.
- [30] ISO/IEC, "Quality management and quality assurance-- vocabulary," vol. ISO 8402, pp. v, 1994.
- [31] *IEEE Standard for a Software Quality Metrics Methodology*, IEEE Standard 1061-1998, 1998.
- [32] S. Ciraci and van Den Broek PM, "Evolvability as a quality attribute of software architectures," in *Int. ERCIM Workshop on Software Evolution*, France, 2006, pp. 6-7.
- [33] ISO, "ISO/IEC 14598-1:1999. Information technology - Software product evaluation -- Part 1: General overview," 1999.
- [34] J. Deprez, F. Monfils F. M. Ciolkowski, and M. Soto, "Defining software evolvability from a free/open-source software," in *3rd Int. IEEE Workshop on Software Evolvability*, Paris, France, 2007, pp. 29-35.
- [35] J. F. Monfils and J. Deprez, "FLOSS managed data sources maturity level: A first attempt," in *3rd Int. IEEE Workshop in Software Evolvability*, Paris, France, 2007, pp. 54-59.
- [36] F. Ruiz et al., "A proposal of a software measurement ontology," in *Proc. Argentine Symposium on Software Engineering*, Buenos Aires, Argentina, 2003.

- [37] D. E. Perry, "Dimensions of software evolution," in *Proc. Int. Conf. on Software Maintenance ICSM*, pp. 296-303, 1994.
- [38] ISO, "TC 176/SC (2005). ISO 9000:2005, Quality management systems- Fundamentals and vocabulary." 2005.
- [39] N. F. Schneidewind, "Report on the IEEE standard for a software quality metrics methodology," in *IEEE Std 1061-1998*, 1998, pp. i.
- [40] D. Izquierdo-Cortazar J. M. Gonzalez-Barahona, S. Duenas, and G. Robles, "Towards automated quality models for software development communities: the QualOSS and FLOSSMetrics case," in *Quality of Info. and Comm. Tech. (QUATIC), 7th Int. Conf.*, 2010, pp. 364-369.
- [41] A. Bergel S. Denier, S. Ducasse, J. Laval, F. Bellingard, P. Vaillergues, F. Balmas, and K. Mordal-Manet, "SQUALE – software QUALity enhancement," in *13th Euro. Conference Software Maintenance and Reengineering (CSMR)*. 2009, pp. 285-288.
- [42] Samoladas, I., G. Gousios, D. Spinellis, and I. Stamelos, "The SQO-OSS quality model: measurement based open source software evaluation," in *OSS'08: Open Source Development, Communities and Quality (IFIP 2.13)*, 2008, pp. 237.
- [43] N. E. Fenton, *Software Metrics: A Rigorous and Practical Approach, Revised (2nd ed.)*. Course Technology, 1998.
- [44] J. A. McCall, P. K. Richards, and G. F. Walters,, "Factors in Software Quality. Vol. I. Concepts and Definitions of Software Quality, General Electric, 1977.
- [45] P. Berander, L. Damm, J. Eriksson, T. Gorschek, K. Henningsson, P. Jonsson, S. Kaa gstrom, D. Milicic, F. Maartensson, K. Ronkko, and P. Tomaszewski, *Software Quality Attributes and Trade-Offs*. Blekinge Institute of Technology, 2005.
- [46] A. Rawashdeh and B. Matakah, "A new software quality model for evaluating COTS components," *Journal of Computer Science*, vol. 2, pp. 373-381, 2006.
- [47] M. Côté, W. Suryn. and E. Georgiadou., "Software quality model requirements for software quality engineering," *Quality Engineering*, 2003.
- [48] B. Boehm, J. Brown, and M. Lipow, "Quantitative evaluation of software quality," in *Proc. 2nd Int. Conf. on SE*, 1976, pp. 592-605.
- [49] N. Bawane and C. V. Srikrishna, "A novel method for quantitative assessment of software quality," *International Journal of Computer Science and Security (IJCSS)*, vol. 3, pp. 508, 2009.
- [50] M. Ortega, M. A. Perez, and T. Rojas, "Construction of a systemic quality model for evaluating a software product," *Software Qual J*, vol. 11, pp. 219-242, 2003.
- [51] ISO, "ISO/IEC 25010:2011 Systems and software engineering - systems and software quality requirements and evaluation (SQUARE)- Systems and software quality models," 2011.
- [52] R. G. G. Dromey, "A model for software product quality," *IEEE Trans. Software Eng.*, vol. 21, pp. 146-162, 1995.
- [53] R. G. Dromey, "Concerning the chimera," *IEEE Software*, vol. 13, pp. 33-43, 1996.

- [54] L. Hyatt and L. Rosenberg, "A software quality model and metrics for risk assessment," in *Proc. of the Product Assurance Symposium and Software Product*, Noordwijk, the Netherlands, 1996.
- [55] A. Origin, QSOS. [Online]. Available: <http://www.qsos.org/> [Accessed: July, 2013].
- [56] J. C. Deprez and S. Alexandre, "Comparing assessment methodologies for free/open source software: OpenBRR and QSOS," *Lect Notes in Comput Sc*, vol. 5089, pp. 189-203, 2008.
- [57] F. W. Duijnhouwer and C. Widdows, "Capgemini expert letter open source maturity model," *CapGemini*, 2003.
- [58] E. Petrinja, R. Nambakam, and A. Sillitti, "Introducing the OpenSource Maturity Model," in *2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, pp. 37-41, 2009.
- [59] OpenBRR [Online]. Available: <http://www.openbrr.org/> [Accessed: July, 2013].
- [60] I. Samoladas, I. Stamelos, and L. Angelis, and A. Oikonomou, "Open source software development should strive for even greater code maintainability," *Commun ACM*, vol. 47, pp. 83-87, 2004.
- [61] G. Gousios, V. Karakoidas, K. Stroggylos, P. Louridas, V. Vlachos, and D. Spinellis, "Software quality assessment of open source software," in *Proc. 11th Panhellenic Conf. on Informatics*, Athens, 2007, pp. 303-315.
- [62] J. Deprez, F. F. Monfilsc, M. Ciolkowski, and M. Soto, "Defining software evolvability from a Free/Open-source software perspective," in *3rd Int. IEEE Workshop on Software Evolvability*, Paris, 2007, pp. 29-35.
- [63] D. Izquierdo-Cortazar J. Gonzalez-Barahona, G. Robles, J. C. Deprez, and V. Auvray, "FLOSS communities: Analyzing evolvability and robustness from an industrial perspective," *Open Source Software: New Horizons*, pp. 336-341, 2010.
- [64] I. Heitlager, T. Kuipers, and J. Visser, "A practical model for measuring maintainability," in *Quality of Information and Communications Technology (QUATIC). 6th Int. Conf.*, 2007, pp. 30-39.
- [65] B. Luijten and J. Visser, "Faster defect resolution with higher technical quality software," in *Proc. 4th Int. Workshop on System Quality and Maintainability*, 2010, pp. 11-20.
- [66] R. Baggen, J. Correia, K. Schill, and J. Visser, "Standardized code quality benchmarking for improving software maintainability," *Software Qual J*, vol. 20, pp. 287-307, 2012.
- [67] SQUALE, 2008 [Online]. Available: <http://www.squale.org/index.html> [Accessed: July, 2013].
- [68] K. Mordal-manet, S. Denier, H. Wertz, J. Laval, F. Bellingard, and P. Vaillergues, "The squale model - A practice-based industrial quality model," in *25th IEEE Int. Conf. on Software Maintenance ICSM*, Edmonton, AB, 2009, pp. 531-534.
- [69] J. Letouzey and T. Coq, "The SQALE Analysis Model: An analysis model compliant with the representation condition for assessing the quality of software source code," in *2nd Int. Conf. on Advances in System Testing and Validation Lifecycle*, Nice, 2010, pp. 43-48.
- [70] J. Letouzey and M. Ilkiewicz, "Managing technical debt with the SQALE Method," *IEEE Software*, vol. 29, pp. 44-51, 2012.
- [71] R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Englewood Cliffs, N.J.: Prentice Hall, 1992.

- [72] D. Taibi et al., *OpenBQR: A Framework for the Assessment of OSS*. Boston, MA: Springer, 2007.
- [73] OpenBRR, 2005 [Online]. Available: <http://www.openbrr.org> [Accessed: July 2013].
- [74] J. Letouzey, "The SQALE method for evaluating technical debt," in *Managing Technical Debt (MTD), 3rd Int. Workshop*, Zurich, 2012, pp. 31-36.
- [75] P. Adams, D. Nutter, S. Rank, and C. Boldyreff, "Using open source tools to support collaboration within CALIBRE," in *Proc. 1st Int. Conf. on Open Source System*, 2005, pp. 61.
- [76] D. Nutter, S. Rank, and C. Boldyreff, "An open source collaboration infrastructure for Calibre," in *Proc. 3rd Workshop in Cooperative Support for Distributed Software Engineering Processes (CSSE)*, Linz, Austria, 2004.
- [77] ISO, "ISO/IEC 15939:2007 Systems and Software Engineering- Measurement process," 2007 .
- [78] H. Mittal and P. Bhatia, "Software maintainability assessment based on fuzzy logic technique," *ACM SIGSOFT Software Engineering Notes*, vol. 34, pp. 1, 2009.
- [79] J. B. Mittal, P. Bhatia, and H. Mittal, "Software maintenance productivity assessment using fuzzy logic," *ACM SIGSOFT Software Engineering Notes*, vol. 34, pp. 1, 2009.
- [80] H. A. Sahraoui, M. Boukadoum, H. M. Chawiche, Gang Mai, and M. Serhani, "A fuzzy logic framework to improve the performance and interpretation of rule-based quality prediction models for OO software," in *Proc. 26th Annual Int. Computer Software and Applications Conf. (COMPSAC)*, 2002, pp. 131-138.
- [81] R. Park, *Goal-Driven Software Measurement: A Guidebook*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
- [82] T. Coq and J. Rosen, "The SQALE quality and analysis models for assessing the quality of ada source code," in *Proc. 16th Ada-Europe Int. Conf. on Reliable Software Technologies*, Edinburgh, UK, 2011, pp. 61-74.
- [83] Y. Singh, P. K. Bhatia, and O. Sangwan, "Predicting software maintenance using fuzzy model," *ACM SIGSOFT Software Engineering Notes*, vol. 34, pp. 1, 2009.
- [84] K. K. Aggarwal et al., "Measurement of software maintainability using a fuzzy model," *Journal of Computer Science*, vol. 1, pp. 538-542, 2005.
- [85] J. Senior, I. Allison, and J. A. Tepper, "Automated software quality visualisation using fuzzy logic techniques," *Techniques*, vol. 7, pp. 25-40, 2007.
- [86] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, pp. 338-353, 1965.
- [87] H. A. Sahraoui, M. Boukadoum, and H. Lounis, "Building quality estimation models with fuzzy threshold values," in *L'OBJET*, 2001.
- [88] P. Cingolani, *jFuzzyLogic Tool* [Online]. Available: <http://jfuzzylogic.sourceforge.net/> [Accessed: July, 2013].
- [89] C. Mathworks, Fuzzy Logic Toolbox [Online]. Available: <http://www.mathworks.com/products/fuzzy-logic/index.html> [Accessed: July 2013].
- [90] D. Izquierdo-cortazar, G. Robles and J. M. Gonz'alez-barahona, "Assessing FLOSS communities: an experience report from the QualOSS project," in *OSS*, pp. 364, 2009.

- [91] M. Soto and M. Ciolkowski, "The QualOSS open source assessment model measuring the performance of open source communities," in *3rd Int. Symposium on Empirical Software Engineering and Measurement (ESEM)*, Lake Buena Vista, FL, 2009, pp. 498-501.
- [92] D. Spinellis, G. Gousios, V. Karakoidas, P. Louridas, P. J. Adams, I. Samoladas, and I. Stamelos, "Evaluating the quality of open source software," *Electronic Notes in Theoretical Computer Science*, vol. 233, pp. 5-28, 2009.
- [93] M. Conklin, J. Howison, and K. Crowston, "Collaboration using OSSmole: a repository of FLOSS data and analyses," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, 2005.
- [94] I. Keivanloo, C. Forbes, J. Rilling, and P. Charland, "Towards sharing source code facts using linked data," in *Proc. 3rd Int. Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation*, Waikiki, HI, 2011, pp. 25-28.
- [95] M. J. Khan, S. Shamail, M. M. Awais, and T. Hussain, "Comparative study of various artificial intelligence techniques to predict software quality," in *IEEE Multitopic Conf. (INMIC)*. Islamabad, 2006, pp. 173-177.
- [96] S. Shafi, S. M. Hassan, A. Arshaq, M. J. Khan, and S. Shamail, "Software quality prediction techniques: A comparative analysis," in *4th Int. Conf. on Emerging Technologies (ICET)*. Rawalpindi, 2008, pp. 242-246.
- [97] A. E. Hassan and R. C. Holt, "Predicting change propagation in software systems," in *Proc. 20th IEEE Int. Conf. on Software Maintenance*, 2004, pp. 284-293.
- [98] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proc. 3rd Int. Workshop on Predictor Models in Software Engineering*, Minneapolis, MN, 2007.
- [99] I. Herraiz, J. M. Gonzalez-Barahona, G. Robles, and D. M. German, "On the prediction of the evolution of libre software projects," in *IEEE Int. Conf. Software Maintenance (ICSM)*, 2007, pp. 405-414.
- [100] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE T Software Eng*, vol. 30, pp. 574-586, 2004.
- [101] N. Bettenburg and A. E. Hassan, "Studying the impact of social interactions on software quality," *Empir Softw Eng*, vol. 18, pp. 375-431, 2013.
- [102] M. Goulao, N. Fonte, M. Wermelinger, and F. B. Abreu, "Software evolution prediction using seasonal time analysis: a comparative study," in *European Conf. in Software Maintenance and Reengineering (CSMR)*, Szeged, 2012, pp. 213-222.
- [103] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE T Software Eng*, vol. 25, pp. 675-689, 1999.
- [104] T. Menzies, B. Caglayan, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan, "*The PROMISE Repository of empirical software engineering data* [online]. Available: <http://promisedata.googlecode.com>, 2012 .
- [105] R. Lincke, T. Gutzmann, and W. Löwe, "Software quality prediction models compared," in *10th Int. Conf. Quality Software (QSIC)*, 2010, pp. 82-91.
- [106] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects," *IEEE T Softw Eng*, vol. 29, pp. 297-310, 2003.

- [107] Ping Yu, T. Systa, and H. Muller, "Predicting fault-proneness using OO metrics: an industrial case study," in *Proc. 6th European Conf. Software Maintenance and Reengineering (CSMR)*, 2002, pp. 99-107.
- [108] C. K. S. Chong Hok Yuen, "On analytic maintenance process data at the global and the detailed levels: A case study," in *Proc. Conf. Software Maintenance*, 1988, pp. 248-255.
- [109] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*. Hoboken: NJ, Holden-Day, 1976.
- [110] B. Kenmei, G. Antoniol, and M. Di Penta, "Trend analysis and issue prediction in large-scale open source systems," in *12th European Conf. Software Maintenance and Reengineering*, 2008, pp. 73-82.
- [111] M. D'Ambros, H. Gall, M. Lanza and M. Pinzger, "Analyzing software repositories to understand software evolution," in *Software Evolution*, T. Mens and S. Demeyer, Eds. Berlin: Springer. 2008. pp. 39-70.
- [112] C. Kiefer, A. Bernstein, and J. Tappolet, "Mining Software Repositories with iSPAROL and a Software Evolution Ontology," in *4th Int. Workshop Mining Software Repositories (MSR)*, May 2007. pp. 10.
- [113] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse,, "How Developers Drive Software Evolution," in *8th Int. Workshop Principles of Software Evolution (IWPSE)*, 2005, pp. 113-122.
- [114] Y. Liu, E. Stroulia, and H. Erdogmus, "Understanding the open-source software development process: a case study with cvschecker," in *Proc. Int. Conf. Open Source Systems*, 2005, pp. 154-161.
- [115] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan, "Mining email social networks," *Proc. Int. Workshop on Mining Software Repositories (MSR)*, Shanghai, China, 2006, pp. 137.
- [116] D. Schreck, V. Dallmeier, and T. Zimmermann, "How documentation evolves over time," in *9th Int. Workshop on Principles of Software Evolution in Conjunction with the 6th ESEC/FSE Joint Meeting (IWPSE)*, 2007, pp. 4.
- [117] A. Zaidman, B. V. Rompaey, S. Demeyer, and A. V. Deursen, "Mining software repositories to study co-evolution of production & test code," in *Int. Conference on Software Testing, Verification, and Validation*, 2008, pp. 220-229.
- [118] A. E. Hassan, "Mining software repositories to guide software development," in *Int. Workshop on Mining Software Repositories (MSR)*, Missouri, USA, 2005.
- [119] R. Arnold and S. Bohner, *Software Change Impact Analysis*, Los Vaqueros, LA: IEEE Computers Society Press, 1996.
- [120] I. Keivanloo, C. Forbes, A. Hmood, M. Erfani, C. Neal, G. Peristerakis, and J. Rilling, "A linked data platform for mining software repositories," in *9th IEEE Working Conference on Mining Software Repositories (MSR)*, Zurich, 2012, pp. 32-35.
- [121] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.
- [122] D. Vldan, "Understanding ontological engineering," *Commun. ACM*, vol. 45, pp. 136-144, 2002.
- [123] M. Saeki and H. Kaiya, "On relationships among models, meta models, and ontologies," in *Proc. 6th OOPSLA Workshop on Domain-Specific Modeling (DSM)*, 2006.
- [124] C. Calero, F. Ruiz, and M. Piattini, *Ontologies for Software Engineering and Software Technology*. Berlin: Springer, 2006.

- [125] T. Menzies, "Cost benefits of ontologies," *Intelligence*, vol. 10, pp. 26-32, 1999.
- [126] M. P. Barcellos, R. A. Falbo, and R. Dal Moro, "A well-founded software measurement ontology," in *Proc. 6th Int. Conf. on Formal Ontology in I.S. (FOIS)*, Toronto, ON, 2010, pp. 213-226.
- [127] J. Rilling, W. Meng, R. Witte, and P. Charland, "Story-driven approach to software evolution," *IET Software*, vol. 2, pp. 304-320, 2008.
- [128] P. Wongthongtham, E. Chang, T. Dillon, and I. Sommerville, "Development of a software engineering ontology for multisite software development," *IEEE T Knowl Data En.*, vol. 21, pp. 1205-1217, 2009.
- [129] N. Guarino, "Formal ontology and information systems," in *Proc. Formal Ontology in I.S (FOIS)*, 1998, pp. 3-15.
- [130] S. Ahmed, "Mining software repositories to support software evolution," M.A. thesis, Faculty Comp. Sci. and Software Eng., Concordia Univ., Montreal, QC., 2009.
- [131] J. Tappolet, "Semantics-aware software project repositories," in *Proc. ESWC Ph.D. Symposium*, Spain, 2008.
- [132] J. Tappolet, C. Kiefer, and A. Bernstein, "Semantic web enabled software analysis," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 8, pp. 225-240, 2010.
- [133] B. Decker, E. Ras, J. Rech, B. Klein, and C. Hoecht, "Self-organized reuse of software engineering knowledge supported by semantic wikis," in *Proc. Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2005.
- [134] M. Wursch, G. Ghezzi, M. Hert, G. Reif, and H. C. Gall, "SEON: a pyramid of ontologies for software evolution and its applications," *Computing*, vol. 94, pp. 857-885, 2012.
- [135] R. C. de Boer and H. van Vliet, "QuOnt: An ontology for the reuse of quality criteria," in *ICSE Workshop Sharing and Reusing Architectural Knowledge (SHARK)*, Vancouver, BC, 2009, pp. 57-64.
- [136] M. Bertoa and A. Vallecillo, *Software Engineering and Software*, Berlin: Springer, 2006.
- [137] M. P. Barcellos, R. D. A. Falbo, A. R. Rocha, and V. Brazil, "Establishing a well-founded conceptualization about software measurement in high maturity levels," in *7th Int. Conf. of Quality of Information and Communications Technology (QUATIC)*, Porto, pp. 467-472, 2010.
- [138] P. Schuegerl, J. Rilling, and P. Charland, "Enriching SE ontologies with bug report quality," in *4th Int. Workshop on Semantic Web Enabled Software Engineering (SWESE) at the 7th Int. Semantic Web Conference (ISWC)*, 2008, pp. 1-15.
- [139] I. Keivanloo, L. Roostapour, P. Schuegerl, and J. Rilling, "SE-CodeSearch: A Scalable Semantic Web-Based Source Code Search Infrastructure," in *IEEE Int Conf. on Software Maintenance (ICSM)*, 2010, pp. 1-5.
- [140] C. W. J. Granger, "Forecasting stock market prices: lessons for forecasters," *Int J Forecasting*, vol. 8, pp. 3, 1992.
- [141] R. D. Edwards and J. Magee, *Technical Analysis of Stock Trends*. Stock Trend Service, 1948.
- [142] J. Chen, *Essentials of Technical Analysis for Financial Markets*. Hoboken, NJ: Wiley, 2010.
- [143] S. W. Bigalow and D. Elliott, "Day-trading with candlesticks and moving averages," *Futures: News, Analysis and Strategies for Futures*, vol. 33, pp. 40-42, 2004.



- [144] G. Appel, *Technical Analysis: Power Tools for Active Investors*. Upper Saddle River, NJ: Financial Times Prentice Hall, 2005.
- [145] SQO-OSS [Online]. Available: <http://www.sqo-oss.org/> [Accessed: July 2013].
- [146] T. Gilb, "Towards the engineering of requirements." *Requir Eng*, vol. 2, pp. 165-169, 1997.
- [147] E. S. Raymond, *The Cathedral and the Bazaar*. 1<sup>st</sup> edition, Sebastopol, CA, USA, O'Reilly & Associates, Inc., 1999.
- [148] H. J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. 2nd Int. Workshop on Semantic Web Enabled Software Engineering (SWESE)*, 2006, pp. 5-9.
- [149] L. Clark & Parsia, Pellet Reasoner [Online]. Available: <http://clarkparsia.com/pellet/> [Accessed: July, 2013].
- [150] L. Obrst, "Ontologies for semantically interoperable systems," in *Proc. 12th Int. Conf. on Information and Knowledge Management*, 2003, pp. 366-369.
- [151] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Comput Surv*, vol. 28, pp. 415-435, 1996.
- [152] J. L. Cybulski, "Introduction to software reuse," The Univ. of Melbourne, Department of IS, Melbourne, Tech. Rep. Research Report 96/4, 1996.
- [153] G. Cardino, F. Baruchelli, and A. Valerio, "The evaluation of framework reusability," *SIGAPP Appl. Comput. Rev.*, vol. 5, pp. 21-27, 1997.
- [154] G. Sindre, R. Conradi, and E. Karlsson, "The REBOOT approach to software reuse," *J Syst Software.*, vol. 30, pp. 201-212, 1995.
- [155] Fernández-López M., A. Gómez-Pérez, and N. Juristo, "Methontology: From ontological art towards ontological engineering," in *Proc. Symposium on Ontological Engineering*, 1997.
- [156] Stanford Center for Biomedical Informatics Research, Protégé [Online]. Available: <http://protege.stanford.edu/> [Accessed: July, 2013].
- [157] E. P. Bontas, M. Mochol, and R. Tolksdorf, "Case studies on ontology reuse," in *Proc. 5th Int. Conf. on Knowledge Management (IKNOW)*, 2005.
- [158] W. Meng, J. Rilling, Y. Zhang, R. Witte, and P. Charland, "An ontological software comprehension process model," in *Proc. 3rd Int. Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering*, 2006, pp. 1-15.
- [159] F. Garcia, F. Ruiz, C. Calero, M. F. Bertoa, A. Vallecillo, B. Mora, and M. Piattini, "Effective use of ontologies in software measurement," *Knowl Eng Rev*, vol. 24, pp. 23-40, December, 2009.
- [160] F. Garcia, M. Bertoa, C. Calero, A. Vallecillo, F. Ruiz, M. Piattini, and M. Genero, "Towards a consistent terminology for software measurement," *Information and Software Technology*, vol. 48, pp. 631-644, 2006.
- [161] Y. Zhang, R. Witte, J. Rilling and V. Haarslev, "An ontology-based approach for traceability recovery," in *3rd Int. Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering*, 2006, pp. 36-43.
- [162] L. Baresi, E. Di Nitto, and C. Ghezzi, "Toward open-world software: issue and challenges," *Computer*, vol. 39, pp. 36-43, 2006.

- [163] InfoEther, PMD Tool [Online]. Available: <http://pmd.sourceforge.net/> [Accessed: July, 2013].
- [164] Oliver Burn, Checkstyle Tool [Online]. <http://checkstyle.sourceforge.net/> [Accessed: July, 2013].
- [165] R. Garcia-Castro, M. Esteban-Gutiérrez, M. Kerrigan, and S. Grimm, "An ontology model to support the automated evaluation of software," in *Int. Conf. in Knowledge Systems (SEKE)*, Redwood City, 2010.
- [166] M. De Los Angeles Martin and L. Olsina, "Towards an ontology for software metrics and indicators as the foundation for a cataloging Web system," in *Proc. of First Latin American Web Congress*, pp.103,113, Nov. 2003
- [167] L. Olsina and M. De Los Angeles Martin, "Ontology for software metrics and indicators," *J of Web Eng*, vol. 2, pp. 262-281, 2003.
- [168] L. Olsina and G. Rossi, "Measuring web application quality with WebQEM," *IEEE Multimedia*, vol. 9, pp. 20-29, 2002.
- [169] J. Deprez, K. Haaland, and F. Kamseu, QualOSS methodology & QualOSS assessment method (2008)[Online].Available [http://www.academia.edu/817534/QualOSS Methodology and QualOSS Assessment Method](http://www.academia.edu/817534/QualOSS_Methodology_and_QualOSS_Assessment_Method),
- [170] S. Harris, Simian Tool [Online]. <http://www.harukizaemon.com/simian/> [Accessed:July, 2013].
- [171]I. Clarkware Consulting, Jdepend Tool [Online].Available: <http://clarkware.com/software/JDepend.html> [Accessed: July, 2013].
- [172] H. Mittal, "Software quality assessment based on fuzzy logic technique," *Int. Journal of Software Computing Applications*, pp. 105-112, 2008.
- [173] P. Cingolani and J. Alcalá-Fdez, "jFuzzyLogic: A robust and flexible fuzzy-logic inference system language implementation," in *IEEE Int. Conference on Fuzzy Systems (FUZZ-IEEE)*, 2012, pp. 1-8.
- [174] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. Berlin: Springer, 2006.
- [175] D. Thomas and A. Hunt, *The Pragmatic Programmer: From Journeyman to Master*. Longman, Amsterdam: Addison-Wesley Professional, 1999.
- [176] K. J. Lieberherr and I. M. Holland, "Assuring good style for object-oriented programs," *IEEE Software*, vol. 6, pp. 38-48, 1989.
- [177] GSyC/LibreSoft, Melquiades (2009) [Online]. Available: <http://melquiades.flossmetrics.org/> [Accessed: July, 2013].
- [178] Black Duck Software, Inc, OHLOH (2010) [Online]. Available: <http://www.ohloh.net/> [Accessed: July, 2013].