Fuzzy Logic Based Software Product Quality Model for Execution Tracing

MPhil Thesis

October 2013

Author: Tamas Galli

P10553741@email.dmu.ac.uk

Internal supervisors: Prof. Francisco Chiclana, Dr. Jenny Carter, Dr. Helge Janicke

External supervisor: Dr. Alexander Seidel

Centre for Computational Intelligence

Faculty of Technology

De Montfort University

The Gateway

Leicester

LE1 9BH

United Kingdom

A thesis submitted in partial fulfilment of the requirements of De Montfort University for the degree of Master of Philosophy.

Acknowledgement

This research has been carried out as the first step of a larger topic in the scope of the international MPhil/PhD program at the De Montfort University in Leicester, UK. I would like to acknowledge gratefully for the work, and support of my supervisors: Prof. Francisco Chiclana, Dr. Jenny Carter, Dr. Helge Janicke, and Dr. Alexander Seidel; moreover, also for their flexibility that they accepted my intention, deviating from the original plans, to change from the PhD route to MPhil. In addition, I would like to express my thanks also to Dr. Shashi Paul, Dr. Irma Agardi for the consultation on research methods, and Dr. Qin Xu for the consultation on quantitative methods.

Furthermore, I would also like to thank for the support of my colleagues, for talks, and comments, particularly to Zoltan Borcsiczky for his contribution to organising, and leading brainstorming, to Imre Miko for his contribution to rating the brainstorming outcomes. Moreover, I would like to express thanks also to the participants in the brainstorming group, and to my chiefs at Siemens CVC for making possible to take unpaid holidays regularly during the research, and thanks to those colleagues who made my substitution at these periods.

Finally, I would like to thank to my family, especially my mother, Maria, and step-father, Mihaly, for their constant encouragement, and Bandi Bacsi, grandmother's husband, and grandmother for their interest where I am in the research process. Thanks also to my friends, Uli for carefully proofreading the research proposal when I applied for admission and for her comments and encouragement, Teofil for his encouragement and showing some interesting steps in his own research and Mary and the sisters for their support. Thanks also to my deceased, old friend Dr. Janos Rausch for the great talks involved also in science years ago. Thanks also to Peter Nemeshegyi S.J. for his great books, the great talks with him and for the explicit expression that looking for the truth is one part of human dignity.

In summary, I would like to thank gratefully to God, Who gave the possibilities, the strength, the creativity to perform this research, and getting so far, and thanks also for the learning process, and for the great, and exciting adventure that all this means while looking for the truth also in the research.

Abstract

This report presents the research carried out in the area of software product quality modelling. Its main endeavour is to consider software product quality with regard to maintainability. Supporting this aim, execution tracing quality, which is a neglected property of the software product quality at present in the quality frameworks under investigation, needs to be described by a model that offers possibilities to link to the overall software product quality frameworks.

The report includes concise description of the research objectives: (1) the thorough investigation of software product quality frameworks from the point of view of the quality property analysability with regard to execution tracing; (2) moreover, extension possibilities of software product quality frameworks, and (3) a pilot quality model developed for execution tracing quality, which is capable to capture subjective uncertainty associated with the software quality measurement.

The report closes with concluding remarks: (1) the present software quality frameworks do not exhibit any property to describe execution tracing quality, (2) execution tracing has a significant impact on the analysability of software systems that increases with the complexity, and (3) the uncertainty associated with execution tracing quality can adequately be expressed by type-1 fuzzy logic. The section potential future work outlines directions into which the research could be continued.

Findings of the research were summarized in two research reports, which were also incorporated in the thesis, and submitted for publication:

1. Tamas Galli, Francisco Chiclana, Jenny Carter, Helge Janicke, “Towards Introducing Execution Tracing to Software Product Quality Frameworks,” Acta Polytechnica Hungarica, vol. 11, no. 3, pp. 5-24, 2014. doi: 10.12700/APH.11.03.2014.03.1

2. Tamas Galli, Francisco Chiclana, Jenny Carter, Helge Janicke “Modelling Execution Tracing Quality by Means of Type-1 Fuzzy Logic,” Acta Polytechnica Hungarica, vol. 10, no. 8, pp. 49-67, 2013. doi: 10.12700/APH.10.08.2013.8.3

Table of Content

List of Figures

List of Tables

Definition of Terms

Term	Definition
ANFIS	Adaptive Neuro-fuzzy Inference System
Execution tracing	A mechanism or tool to dump the data about the program state, and the path of execution for offline analysis, mainly for software developers; frequently used as synonym for logging in the literature
Logging	A mechanism or tool to dump the data about the program state, and the path of execution for offline analysis, mainly for system administrators; frequently used as synonym for execution tracing in the literature
Quality attribute	A low-level quality property in ISO/IEC 25000 standard family, in contrast to the ISO/IEC 14598 standard family where the term attribute is used to refer to the high-level quality properties of the ISO/IEC 9126 family
Quality characteristic	A high-level quality property that is located at the top of the hierarchy in the OSI/IEC 9126 standard family; in the terminology of ISO/IEC 14598 standard family it is called attribute
Quality measure	The association of quality measure elements, and a measurement function to calculate with; mainly used in the ISO/IEC 25000 standard family with similar meaning to the terminology metric in the ISO/IEC 9126 standard family
Quality measure element	A measurable property of quality defined in ISO/IEC 25021
Quality metric	The definition of the measurement method of quality properties including the definition of the measurement scale; quality metrics are assigned to sub-characteristics or characteristics; this term is used in the ISO/IEC 9126 standard family
Software product quality framework	A model that describes each aspect of software product quality; it is a complete software product quality model
Software product quality model	A model that describes either each aspect of the software product quality or only a part of it; it is not necessarily a complete model.

Table 1 Definition of Term Introduction

1 Introduction

Execution tracing and logging are frequently used as synonyms in software technology; however, the first one rather serves the software developers to localize errors in applications, while the second one contributes to administration tasks to check the state of software systems [1], [2], [3], [4], [5]. In the scope of this report we also use the two words as synonyms.

Execution tracing dumps the data about the program state and the path of execution for developers for offline analysis, which helps to investigate error scenarios and follow changes in the state of the application. Thus, execution tracing and logging belong to dynamic analysis techniques i.e. testing, investigating live systems, which are integral parts of the maintenance activities. Dynamic analysis techniques can be applied only if the software is built and executable in contrast to static monitoring techniques. However, both methods are applied to achieve the same goal of diagnosing errors, with each technique having its own particular advantages [6], [7], [8], [9].

Spillner, Linz, and Schaeffer differentiate two types of maintenance [10]: (1) corrective maintenance, which aims at eliminating errors in the software, and (2) adaptive maintenance aiming at changing the software according to new requirements. Both kinds of maintenance necessitate setting analysis methods to find errors although this activity dominates in corrective maintenance [10]. The proportion of maintenance costs in the whole software life-cycle amounts to a large part [11], [12], [13], thus decreasing the time devoted to localizing errors can therefore decrease the maintenance costs.

The increasing size and complexity of software systems make localizing software errors more difficult. This difficulty is more challenging with regard to the enormous number of software and hardware combinations. Adding execution trace to some key places of the application can drastically reduce the time spent with debugging because it helps to narrow down the source of the potential errors [14].

Utilizing a debugger is time consuming and does not offer adequate solution

- if performance problems have to be resolved because debugging the source code considerably changes the environment from the point of view of execution performance. Moreover, performance is sensitive to external workload, configuration parameters, underlying hardware, and software components [14] .
- in case of real-time, embedded systems as it might be harmful or impossible to reproduce the error in the case of control applications [15].
- in the case of concurrency, as it changes the race conditions[1] for parallel running execution threads or processes. In addition, multi-core systems also need to be considered which may even have multiple clock domains [15].

A wide survey on concurrency [16], for which 10% of all Microsoft employees from development, test, and program management were selected, also supports that analysing concurrency faults makes up a significant part of their correction costs. 66% of the respondents had to deal with concurrency issues. The reproduction of these issues was classified in a five categorical scale ranging from easy to very hard. 72.9% of all responses classified reproduction of concurrency issues in the two most difficult categories. Moreover, the respondents stated that the severity of these issues, qualifying on an ordinal scale with four categories ranging from least severe to most severe, belongs to the top two: most severe, and severe. In addition, 65% of the respondents expressed the future expectation that the concurrency issues would be more problematic.

Laddad states in [17] that execution tracing is the only adequate tool to help with the analysis of run-time errors in the case of distributed systems and multithreaded applications. In the case of embedded applications, which have no user interface, by means of tracing the developer or system maintainer can answer questions such as what the application is doing [15].

Diagnosing regression test errors, and finding root causes implicate major difficulties.

[1] Execution tracing influences the race conditions to a significantly smaller extent than using a debugger as it does not suspend threads and does not wait for manual interventions from the developer.

Fault localizations can be grouped in three categories [18]: (1) dynamic dependence analysis of the failing program execution, (2) comparison of the failing program execution with a set of error free executions, and (3) comparison of the failing program execution with a program execution which does not manifest the error in analysis. Categories (2) and (3) are based on execution tracing.

An experiment conducted by Karahasanovic and Thomas categorised the difficulties related to the maintainability of object-oriented applications [19]. Program logic was ranked the first in the source of difficulties. Understanding the program logic belongs to the category of software specific knowledge. Execution tracing can greatly enhance understanding of the program logic and it offers a basis for trace visualization and program comprehension [20].

Tracing, logging, or constraint checks represent a significant part of the source code of applications. Spinczyk, Urban, and Lehmann [21] state that the ratio of code lines related to monitoring activities such as tracing reached approximately 25% in their measurements of commercial applications. This ratio shows that a significant amount of source code is written to deal with execution tracing that needs to be considered from the point of view of software quality. However, execution tracing does not need to be tightly coupled to the application code and can be localized in separate modules [22], [17], [23].

All the above indicate that execution tracing and logging have essential impacts on the analysability of software systems. In the case of embedded applications with no user interface in the safety critical domain or server-systems with concurrency issues these tools are inevitable to localize errors or investigate software behaviour.

1.1 Problem Statement

Software product quality measurement is important for ensuring that software applications achieve the desired standard. However, measuring software product quality is difficult. Some *quality measure element*s[2] are easier to measure than others [24]. Software product quality frameworks include the description of qualitative properties in a quantitative manner; as well as *quality measure element*s that cannot be measured directly but only derived from the observation of the behaviour of software developers, maintainers, operators, and users. The observation of behaviour and the difference in measurement difficulty of the *quality measure element*s can lead to the introduction of uncertainty in software product quality frameworks, which has recently been admitted and accepted by defining the subjective measurement method category in ISO/IEC 25021:2007 [25]:

> "Subjective measurement method - Subjective measurements are those where quantification is influenced by human judgement. Subjective measures are used when no formal objective procedures of measurement can be applied. The value of the *quality measure element* is influenced by human judgement as an evaluator. Therefore it is necessary to interpret the results with respect to the number of evaluators and statistical methods used for the measurement result calculation. Both should be stated while presenting the measurement results."

Manifestations of uncertainty can be classified into three broad categories: (1) objective uncertainty that refers to the future, (2) subjective uncertainty that refers to the future, and (3) subjective uncertainty that does not refer to the future but helps to categorize elements [26], [27]. Category 1 is modelled by the classical probability theory, while category 2 is considered as an application area of Bayesian statistics. Category (3) on the other hand, is modelled and studied under the name of fuzzy logic. Thus, we also aim to examine in the scope of the research how far the current quality frameworks can ensure the link to *quality measure*s described by means of fuzzy logic to consider the above subjective uncertainty.

[2] Terminology of ISO/IEC 25021, see Definition of Terms for further details.

1.2 Research Questions

The research conducted and reported here aimed to investigate the following questions:

1. Is execution tracing appropriately considered by present software product quality frameworks?
2. If the present software product quality frameworks do not address execution tracing quality in an appropriate manner, what would be necessary to incorporate execution tracing in these frameworks?
3. Could execution tracing quality be described by a standalone model capturing subjective uncertainty?

1.3 Research Objectives

The research objectives are:

1. To carry out a thorough analysis of software product quality frameworks.
2. To investigate how execution tracing could be appropriately implemented in current software quality frameworks.
3. To conduct a pilot study of a quality model for execution tracing to capture subjective uncertainty.

1.4 Research Methodology

The research comprises of two parts: a non-experimental and an experimental. The non-experimental research was conducted as a descriptive research to critically review the present software product quality frameworks and their relationship to execution tracing quality.

The experimental research comprised of two types: (1) qualitative, (2) quantitative.

Qualitative research was applied in the pilot study to determine the properties on which execution tracing quality depends. The pilot study investigated the feasibility of modelling execution tracing quality when considering subjective uncertainty. Quantitative

experimental research was used to compare the performance of the different models in the pilot study.

1.4.1 Descriptive Research

A comprehensive literature review was performed to gain knowledge of the different software product quality frameworks. Moreover, existing techniques were reviewed for incorporating subjective uncertainty as well as execution tracing in the model. The investigation and its results are introduced in the following sections: 2.1 Software Product Quality Frameworks and 6.1 What Mathematical Tool Can Help to Incorporate Subjective Uncertainty.

1.4.2 Qualitative Research

In the scope of the pilot study, qualitative research was used to find out the quality properties to use to describe and model execution tracing quality. The purpose of the pilot study was to test the performance of different methods in the model creation and not to implement a final model with generalisable quality properties.

The qualitative research comprised of the following steps: (1) forming a brainstorming group of software developers and software maintainers, (2) collecting data by brainstorming with a facilitated group, (3) coding the data, and (4) determining the input parameters of the model for execution tracing quality to help to explore the in-depth experiences of software developers and software maintainers. The research variables are identified as the output of the qualitative research to offer the input variables for the quantitative research.

1.4.2.1 Brainstorming

Brainstorming was developed by A. Osborn and sophisticated by H. C. Clark as a technique to create, collect and express ideas on a topic [28]. The main principle of the method is formed by two fundamental factors: (1) each group member must have the possibility to express ideas without having to expose them to critic at first, (2) the ideas can be developed further by other group members. Consequently, synergistic effects can lead to triggering new ideas by already present ideas. Before and after the idea generation phase an ideation phase must take place. The ideation phase following the idea generation phase needs to contain evaluation of the ideas collected [28]. The critics towards this method mainly focus on the idea generation phase regardless of the previous and afterwards ideation phases. However, Osborn did not propose brainstorming as a substitute of the ideation but to supplement it [29]. In this method the quantity of ideas collected is urged not the quality as the more ideas are collected the more probable it is to have quality ideas among them. The latter has been questioned in [30], which contradicts in some respects [29].

Differences appear in the performance of the groups depending on whether they are moderated, i.e. facilitated, or not [29]. Moreover, while evaluating the number of ideas collected by different groups, Isaksen and Gaulin illustrate that nominal groups[3] are outperformed by facilitated brainstorming groups in their measurements. In contrast, Goldenberg and Wiley [30] also introduces measurements in which nominal groups outperformed facilitated brainstorming groups. However, both research studies conclude that brainstorming remains a popular tool for idea generation and the dissent of the group members can be harnessed.

Brainstorming adheres to the following strict rules [28]:

- Do not express critic during the idea generation phase
- The more ideas the better
- Extend and improve already presented ideas
- The more unusual the idea is the better it is
- The group must be facilitated i.e. a leader needs to moderate the group

[3] The notion of nominal group refers to the same number of individuals working alone [30].

- Optimal group size ranges between 4 and 20 participants to assure that (1) participants should have enough possibility to express ideas and (2) group dynamic effects also need to be in work
- Recording needs to be done by one or more protocol writers
- Duration of the idea generation should be between 5 and 30 minutes

A group can only be considered to be a brainstorming group if the rules of brainstorming are kept, which is not always simple to judge [28].

The facilitator has the following responsibilities [28]:

- To give an introduction to the participants before starting the idea generation
- To reinforce the rules
- To present all ideas at the end of the idea generation in front of the participants
- To evaluate all ideas with the group after the idea generation
- To cluster, to group all ideas during the evaluation phase with the group
- To coordinate the discussion during evaluation phase, critic can also be expressed
- To reject ideas based on the groups opinion

The output of the brainstorming is a list of raw ideas considered to be feasible by the group. The output needs to undergo further investigation to code the data. In the scope of the pilot study, this was performed using constant sum scaling [31].

1.4.2.1.1 Barriers to Effective Brainstorming and Their Overcoming

Isaksen and Gaulin in [29] describe some potential barriers to brainstorming and provide recommendations to overcome them.

Potential barriers:

- Reduction of the participants' motivation in the collective work for any reason including quality of communication among the group members

- Evaluation practiced during the idea generation retards the group
- The process setting encourages only one person to contribute ideas at a time and others are prohibited to talk until the idea is recorded. This can form a barrier in large brainstorming groups.

Recommendation to overcome potential barriers:

- Using group facilitators (1) to reinforce the guidelines, (2) to encourage even participation of the group members, (3) to control the interactions, (4) to record quickly
- Accelerating the recording. It can also be achieved by using the technology to record more ideas simultaneously (e.g. electronic brainstorming system).

In the context of the pilot study the brainstorming group was facilitated and the moderator prevented to evaluate ideas during the idea collection phase. The pace of idea recording was appropriate with regard to the number of the participants in the group. The implementation of the brainstorming and the idea collection, evaluation is covered in more detail in the section 4.1.1 Determining the Inputs and the Output of the Model.

1.4.2.2 Coding Data

Data coding is applied to gain structured information from unstructured data; moreover, it aids to explore or verify relationships between entities in the data corpus [32]. Data coding already starts in the idea generation phase of the brainstorming and it continues in the latter evaluation phase.

The list of feasible ideas collected by the group in the scope of the pilot study underwent analysis by two experts who scored the input candidates according to their importance with regard to execution tracing quality. The experts had to distribute the same amount of scores among the items collected i.e. constant sum scaling was applied [31]. The arithmetic means of the scores assigned by experts were calculated to formulate a final

and collective score value. Each selected input candidate was assigned a relative importance above 10% according to the judgement of the experts. 10% is a subjective limit and means that 90% of the scores have been considered by the inputs. The implementation of data coding performed in the scope of the research is described in detail in the section: 4.1.1 Determining the Inputs and the Output of the Model.

1.4.3 Experimental Research

After identifying the inputs of the quality model for execution tracing with qualitative methods i.e. with brainstorming and data coding to formalise the experiences of a group of software developers and maintainers, experimental research was conducted to incorporate one expert's opinion in the constructed quality model to implement uncertainty and to test the performance of the created models.

In summary, this research part contained the following steps:

1. Defining fuzzy sets and describing the expert's opinion with type-1 fuzzy rules to make possible the formalisation subjective inputs.
2. Mathematical modelling of fuzzy rules by using different membership functions: (1) triangular and (2) Gaussian membership functions.
3. Implementation of fuzzy inference in two different ways: (1) with Mamdani's approach and (2) Takagi-Sugeno-Kang approach. Moreover, the output sets that result from the application of Mamdani's approach were transformed to a crisp domain with different defuzzification techniques.
4. The performance of the models was tested on defined criteria with regard to further optimization.

The steps of the experimental research are described in detail in sections: 4.1.4 Knowledge Base for the Model, 4.1.6 Comparison of the Created Models, 4.2 Validation.

1.5 Related Works

Some researchers have already illustrated how fuzzy mathematics can help to make judgments or predictions in connection with software maintainability [33], [34], [35] or reusability [36], [37].

Canfora, Cerulo, Troiano in [34] applied fuzzy logic to consider the following particularities regarding maintainability:

1. The assessment of software maintainability is influenced by qualitative and quantitative data including their subjective uncertainty.
2. Qualitative data that are often gathered by surveys are not always available.
3. The different sub-characteristics of maintainability contribute to the overall maintainability to different extents.

Aggarwal, Singh, Chandra, Manimala discussed in [33] how an integrated metric of maintainability correlated with the time devoted to error corrections. However, individually none of the investigated inputs of their model correlated with the time spent on error corrections. The model was constructed using type-1 fuzzy logic.

Nerurkar, Kumar, Shrivastava in [36] proposed a model based on type-1 fuzzy logic for reusability of aspect-oriented systems. Singh, Bhatia, Sangwan in [37] examined different soft computing techniques for software reusability assessment. In their publication, type-1 fuzzy logic, neural network, and adaptive neuro-fuzzy inference were compared for evaluating software reusability.

However, these models cannot help with the assessment of software product quality as a whole because they are not linked to extensive software product quality frameworks like ISO/IEC 25010 [25]. In addition, the maintainability models investigated do not handle execution tracing quality.

1.6 Structure of the Thesis

The thesis is structured in the following way: Chapter 2 presents the literature review focused on the present software product quality frameworks and on execution tracing. Section 6.1 would also fit semantically in this chapter but it was not the subject of the research as its output was only used as a tool to describe the subjective uncertainty involved in quality modelling. Therefore this section is placed in the Appendix 6.

Chapter 3 introduces how the present software product quality frameworks could be extended to describe also execution tracing quality.

Chapter 4 describes how execution tracing quality can be modelled with regard to the subjective uncertainty associated to software quality.

Chapter 5 presents the conclusions and possibilities for further work in the area.

A review of fuzzy set theory and fuzzy logic, which was used to capture the subjective uncertainty associated with quality measurements and modelling, is provided in Appendix 6. Matlab diagrams and Matlab code of the models built in the scope of the pilot study, except the one already presented in chapter 4, are also provided in Appendix 6.

2 Review of the Relevant Literature

This chapter contains a review of the literature that comprises the following three main areas: (1) software product quality frameworks, (2) execution tracing, and (3) subjective uncertainty. The aim of the conducted review was to collect and present the relevant literature in connection with the research objectives.

As the research is not a research of uncertainty in itself, but the application of methods that help to describe uncertainty, the part of the literature review related to this area was moved to Appendix 6. Moreover, the relevant techniques from the point of view of applicability in the scope of the present research are considered, i.e. techniques that can describe uncertainty of the present. Consequently, methods related to uncertainty that refer to the future, i.e. probability theory and Bayesian statistics, are out of scope of the literature review.

2.1 Software Product Quality Frameworks

As stated in the Definition of Terms, software product quality frameworks are models that describe each aspect of software product quality. Consequently, the analysis of quality frameworks focused on software product quality models that describe the whole set of software product quality.

The presentation of the software product quality frameworks classifies the models in two categories: (1) early frameworks that appeared in the 1970's, and (2) recent frameworks that appeared after 1990.

2.1.1 Early Frameworks

Early software product quality frameworks appeared in the second half of the 1970's to assess quality and show the way for improvements in software products [38], [39]. These frameworks had a significant influence on the recent software product quality

frameworks published by the ISO standards. They kept the hierarchic nature abstraction of quality.

2.1.1.1 Boehm, Brown, and Lipow' Software Product Quality Model

The first complete model to assess software product quality was developed by Boehm, Brown, and Lipow [38]. They established a set of quality properties, which they call characteristics, and one or more metrics to each of them. They defined the notion of metric as (1) a quantitative measure that describes the degree to which the software product possesses the given characteristic, and (2) the overall software quality must be able to be described by the function of the values of the metrics.

They came to the conclusion in their study that establishing a single overall metric for software product quality would implicate more difficulties than benefits because many of the major individual quality characteristics are conflicting; moreover, the metrics they associate to the quality characteristics are incomplete measures of the quality characteristics. Therefore, they developed a hierarchical model. The hierarchy comprises of eleven high-level characteristics representing different aspects of software product quality [38]: (1) understandability, (2) completeness, (3) conciseness, (4) portability, (5) consistency, (6) maintainability, (7) testability, (8) usability, (9) reliability, (10) structuredness, and (11) efficiency. The lower level in the quality hierarchy provides more primitive characteristics that can be expressed by assignment of metrics in an easier and more trustworthy manner.

The authors define the high-level characteristics as follows [38]:

1. Understandability: Code possesses the characteristic of understandability to the extent that its purpose is clear to the inspector. This implies that variable names or symbols are used consistently, modules of code are self-descriptive, and the control structure is simple or in accordance with a prescribed standard.
2. Completeness: Code possesses the characteristic of completeness to the extent that all its parts are present and each part is fully developed. This implies that

external references are available, required functions are coded and present as designed.

3. Conciseness: Code possesses the characteristic of conciseness to the extent that excessive information is not present. This implies that programs are not excessively fragmented into modules, overlays, functions and subroutines, nor that the same sequence of code is repeated n numerous places, rather than defining a subroutine or macro.
4. Portability: Code possesses the characteristic of portability to the extent that it can be operated easily and well on computer configurations other than its current one.
5. Consistency: Code possesses the characteristic of internal consistency to the extent that it contains uniform notation, terminology, and symbology within itself, and external consistency to the extent that the content is traceable to the requirements. Internal consistency implies that coding standards are homogeneously adhered to. Comments should not be unnecessarily extensive or wordy at one place, and insufficiently informative at another; the number of arguments in subroutine calls should match the subroutine header. External consistency implies that variable names and definitions, including physical units, are consistent with a glossary or, there is a one-to-one relationship between functional flowchart entities and coded routines or modules.
6. Maintainability: Code possesses the characteristic of maintainability to the extent that it facilitates updating to satisfy new requirements or to correct deficiencies. This implies that the code is understandable, testable, and modifiable, e.g.: comments are used to locate subroutine calls and entry points, visual search or locations of branching statements and their targets is facilitated by special formats, or the program is designed to fit into available resources with plenty of margins to avoid major redesign.
7. Testability: Code possesses the characteristic of testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance. This implies that requirements are matched to specific modules, or diagnostic capabilities are provided.

8. Usability: Code possesses the characteristic of usability to the extent that it is reliable, efficient, and human engineered. This implies that the function performed by the program is useful elsewhere, is robust against human errors (e.g., accepts either integer or real representations for type real variables), or does not require excessive core memory.
9. Reliability: Code possesses the characteristic reliability to the extent that it can be expected to perform its intended functions satisfactorily. It implies that the program will compile, load, and execute, producing answers of the requisite accuracy; and that the program will continue to operate correctly, except for a tolerably small number of instances, while in operational use. It also implies that it is complete and externally consistent.
10. Structuredness: Code possesses the characteristic of structuredness to the extent that it possesses a definite pattern of organization of its interdependent parts. This implies that evolution of the program design has proceeded in an orderly and systematic manner, and that standard control structures have been followed in coding the program.
11. Efficiency: Code possesses the characteristic of efficiency to the extent that it fulfils its purpose without waste of resources. This implies that choices of source code constructions are made in order to produce the minimum number of words of object code; or that, where alternate algorithms are available, those taking the least time are chosen; or that information-packing density in core is high. Of course, many of the ways of coding efficiently are not necessarily efficient in the sense of being cost-effective, since portability, maintainability, etc., may be degraded as a result.

The model is language-independent and independent of programming paradigms, however many metrics were tested on the structured language Fortran. Additional metrics to the published ones can easily be defined and the model offers possibilities for tailoring.

2.1.1.2 McCall, Richards, and Walters' Software Product Quality Model

McCall, Richards, and Walters published a different framework [39] to Boehm's model [38] to assess software product quality. The authors describe a global view of software product quality as a combination of three distinct activities: (1) product operation, (2) product revision, and (3) product transition i.e. the description considers also process related properties. The objective of their investigation was to provide a concept to acquisition managers to specify and measure quality in a quantitative manner in software products related to air force applications.

They established a set of software quality properties that describe the overall quality of the software product and they named these properties factors. The quality factors they associated with criteria. Criteria are attributes of the software or software development process by which the factors can be judged and defined. A criterion can have sub-criteria in a hierarchical manner and one criterion may affect more quality factors. The criteria are coupled with metrics that make possible the measurement of the criteria or sub-criteria. The separation between properties that would also qualify for being both criterion and factor the authors made the decision: user-oriented properties are quality factors while software-oriented are criteria as the following example demonstrates:

> "Complexity and modularity are software-oriented rather than user-oriented terms. The user is interested in such things as how fast the program runs (efficiency) and how easy it is to maintain (maintainability), not how modular it is." [39]

For the establishment of criteria to the quality factors the authors named the following purposes [39]:

1. A set of criteria helps to define the factor more precisely.
2. Criteria that impact on more than one quality factor help to display the relationships between the quality factors.
3. The criteria allow a one-to-one relationship between metrics and criteria.
4. The introduction of criteria in the model further emphasises the hierarchical nature of the model.

The authors define quality factors in the following manner [39]:

1. Correctness: Extent to which a program satisfies its specifications and fulfils the user's mission objectives.
2. Reliability: Extent to which a program can be expected to perform its intended function with required precision.
3. Efficiency: The amount of computing resources and code required by a program to perform a function.
4. Integrity: Extent to which access to software or data by unauthorized persons can be controlled.
5. Usability: Effort that is required to learn, operate, prepare the input, and interpret the output of a program.
6. Maintainability: Effort that is required to locate and fix an error in an operational program.
7. Testability: Effort that is required to test a program to insure it performs its intended function.
8. Flexibility: Effort that is required to modify an operational program.
9. Portability: Effort that is required to transfer a program from one hardware configuration and/or software system environment to another.
10. Reusability: Extent to which a program can be used in other applications, related to the packaging and scope of the functions that programs perform.
11. Interoperability: Effort that is required to couple one system with another.

In summary, the quality factors listed on basis of the life-cycle activities are grouped in the following way, emphasising the importance of the factors in the particular activities [39]:

1. Product operation:
    a. correctness,
    b. reliability,
    c. efficiency,
    d. integrity,
    e. usability;
2. Product revision:
    a. maintainability,
    b. testability,

   c. flexibility;
3. Transition:
   a. portability,
   b. reusability,
   c. interoperability;

The authors also investigated the impact of the quality factors on each other, i.e. if a particular factor is present with a high degree of quality what quality is expected for the other factors. Beside the positive relationships, there exist also negative ones between some quality factors. In those cases finding a compromise is crucial, e.g. integrity and interoperability conflict with each other, which means that the more interoperable the system is the more difficult it is to keep its integrity.

Similarly to the previous framework, this model is language-independent because its metrics are language independent and independent of programming paradigms. It leaves room for extension and tailoring.

2.1.2 Recent Frameworks

Recent software product quality frameworks appeared after 1990. They can be divided into three categories on the basis of their philosophy [40], [41], [42], [43]: (1) hierarchic models of the ISO/IEC standards which are strongly influenced by the early frameworks, (2) adaptations of the ISO/IEC standards, and (3) the non-hierarchic framework of Dromey. Their presentation follows in historical order.

2.1.2.1 Software Product Quality Model of ISO/IEC 9126 Standard Family

The quality model of the ISO/IEC 9126 standard family comprises a hierarchic model for software product quality and *quality in use* as Figure 1 and 2 show.

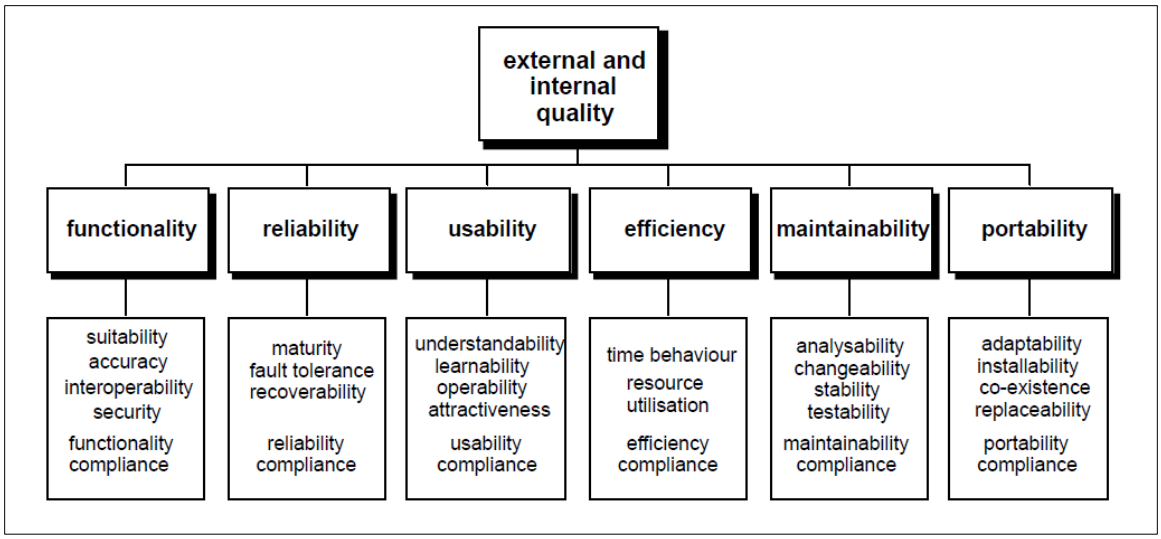

Figure 1. External and Internal Quality Characteristics, source: ISO/IEC 9126-1

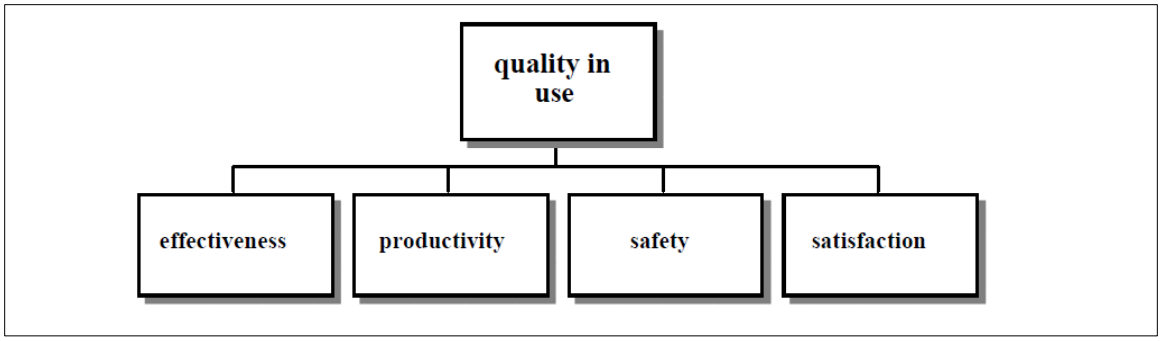

Figure 2. Quality in Use Characteristics, source: ISO/IEC 9126-1

The first version of the standard was issued in 1991 and was superseded ten years later by the next version ISO/IEC 9126-1:2001 [40]. The description of software evaluation was moved from the second version to the multipart standard ISO/IEC 14598 [44] which also introduced some inconsistencies between the definition of terms as quality attribute and quality characteristic of both standards show. The standard ISO/IEC 25010:2011 [41] revised the quality models described by the ISO/IEC 9126 standard family and endeavours to unify also the definition of terms.

Terminology:

- *Quality characteristics*: high-level quality properties which are located at the top of the hierarchy. In the terminology of ISO/IEC 14598 standard family they are called attributes.

- *Sub-characteristics*: Quality properties which are located somewhere in the hierarchy but not at the top-level. Sub-characteristics are always assigned to a higher level characteristic or sub-characteristic.
- *Quality metric*s: Definition of the measurement method of quality properties including the definition of the measurement scale. *Quality metric*s are assigned to sub-characteristics or characteristics.
- *Internal quality* metrics: Metrics whose inputs are formed by the intrinsic properties of the software product.
- *External quality* metrics: Metrics that cannot be measured directly but only derived how the software relates to its environment.
- *Quality of use*: The user's view of quality.

Concepts:

The standard ISO/IEC 9126-1:2001 defines three basic views of the quality: (1) internal view, (2) external view, and (3) user's view. Internal view of the quality means the quality measured by the *internal quality* metrics. This reflects the quality of the source code or documentation. It is very useful if the software product is not developed as far as it could be tested. The external view of the quality is measured by the external metrics. It shows how the product relates to its environment. The user's view of the quality is illustrated by the quality in use reflected by the *quality in use* metrics, however the ISO/IEC 9126-1:2001 standard does not list predefined metrics for the quality in use model.

Internal and external metrics either need to be in cause-effect relationships or they need to correlate with each other. This is called predictive validity i.e. from the measurement by the internal metrics conclusions can be drawn in relation to the *external metrics* and *external quality* of the software.

The software product quality model introduces six high-level characteristics (see Figure 1): (1) functionality, (2) reliability, (3) usability, (4) efficiency, (5) maintainability, (6) portability. In addition to their sub-characteristics, each of these characteristics has an internal and external variant to form an internal and external model as explained above.

The standard defines the six high-level characteristics in the following manner [40]:

1. Functionality: The capability of the software product to provide functions meeting stated and implied needs when the software is used under specified conditions.
2. Reliability: The capability of the software product to maintain a specified level of performance when used under specified conditions.
3. Usability: The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.
4. Efficiency: The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
5. Maintainability: The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.
6. Portability: The capability of the software product to be transferred from one environment to another. The environment may include organisational, hardware or software environment.

The *quality in use* model has four high-level characteristics without sub-characteristics (see Figure 2): (1) effectiveness, (2) productivity, (3) safety, and (4) satisfaction. The external model needs to have predictive validity for the *quality in use* model, i.e. from the external model conclusion can be drawn to predict the quality in use characteristics.

The standard defines the four high-level characteristics in the following manner [40]:

1. Effectiveness: The capability of the software product to enable users to achieve specified goals with accuracy and completeness in a specified context of use.
2. Productivity: The capability of the software product to enable users to expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use. Relevant resources can include time to complete the task, the user's effort, materials or the financial cost of usage.
3. Security: The capability of the software product to achieve acceptable levels of risk of harm to people, business, software, property or the environment in a specified context of use.

4. Satisfaction: The capability of the software product to satisfy users in a specified context of use. Satisfaction is the user's response to interaction with the product, and includes attitudes towards use of the product.

The model is language-independent and independent of programming paradigms because its concept and predefined metrics do not depend on any particular programming paradigm or language.

2.1.2.2 Software Product Quality Model of Dromey

Software does not directly display quality properties but it shows product properties, which contribute to the quality properties in a positive or negative way. Dromey argued that the previously published software product quality models adequately addressed these particularities. He proposed a model where the main focused was on the product properties, which he calls quality-carrying properties, and on the relationship between product and quality properties in a non-hierarchic manner [43].

Terminology:

- *Quality attributes*: high-level quality property of the model
- *Structural forms*: programming language constructs but they can also be other entities related to the software
- *Quality-carrying properties*: binary-value variables which determine the quality

As quality attributes Dromey identified the six high-level quality characteristics of the ISO/IEC 9126:1991 standard and extended this set with the attribute reusability, which he defined in the following manner [43]:

> "A structural form is reusable if it uses standard language features; it contains no machine dependencies and it implements a single, well-defined, encapsulated, and precisely specified function whose computations are all fully adjustable and use no global variables or side-effects. All ranges associated with computations and data structures in a reusable module

> should have both their lower and upper bounds parameterised. Also no variable should be assigned to a number or any other fixed constant and all constants used should be declared."

The definition of the model is language dependent in contrast to the previous software product quality frameworks presented because it uses programming language level constructs as structural forms and their properties as quality-carrying properties.

Concepts:

The model makes possible the specification and analysis of the relationships between quality attributes, quality-carrying properties, and structural forms. The bottom-up approach facilitates for developers to specify or investigate which quality-carrying properties be associated to the structural forms of a particular application. The top-down approach facilitates for designers to specify the quality requirements and attributes the software needs to satisfy and identify the quality-carrying properties for the structural forms to fulfil the quality needs [43].

If all quality-carrying properties associated with a structural form are satisfied, then that structural form will cause no quality defect in the software. In addition, if any of the quality-carrying properties of the structural forms are violated, then each violation will cause a quality defect in the software [43].

Dromey identified the following structural forms in [43]:

1. System (Set of programs)
2. Library (Set of ADTs, functions, procedures)
3. Meta-program (e.g. shell script using I/O)
4. Program
5. User interface
6. Objects (ADT)
7. Module (It encompasses functions and procedures)
8. Sequence
9. Statement
    a. Loop

   b. Selection
   c. Function/procedure call
   d. Assignment
10. Guard
11. Expression
12. Records
13. Variables
14. Constants
15. Types

The identification of the initial set of structural forms only supports the imperative programming paradigm with the procedural programming paradigm. The structural forms are presented in their order of precedence (1= high precedence, 15=low low precedence). Dromey provided no definition to the structural form apart from the broad description that they identify constructs of the imperative programming paradigm [43].

Quality-carrying properties[4] can be divided into four categories presented in their order of precedence [43] (1.a = high precedence, 4.c = low precedence):

Correctness properties [43]:

1. The correctness properties represent specification-independent minimum requirements for correctness, irrespective of the problem being solved.
   a. C1 Computable: Result obeys laws of arithmetic, etc.
   b. C2 Complete: All elements of structural form satisfied
   c. C3 Assigned: Variable given value before use
   d. C4 Precise: Adequate accuracy preserved in computations
   e. C5 Initialized: Assignments to loop variables establish invariant
   f. C6 Progressive: Each branch/iteration decreases variant function
   g. C7 Variant: Loop guard derivable from variant function
   h. C8 Consistent: No improper use or side-effects

Style properties [43]:

The style properties cover characteristics associated with both high and low-level design, and the extent to which the software's functionality at all levels is specified, and

[4] Quality-carrying properties are defined in detail in [43]. A brief introduction is presented here.

documented. This group contains: (1) structural properties, (2) modularity properties, and (3) descriptive properties.

2. Structural properties:
    a. S1 Structured: Single-entry/single-exit
    b. S2 Resolved: Data structure/control structure matching
    c. S3 Homogeneous: Only conjunctive invariants for loops
    d. S4 Effective: No computational redundancy
    e. S5 Non-redundant: No logical redundancy
    f. S6 Direct: Problem-specific representation
    g. S7 Adjustable: Parameterized
    h. S8 Range-independent: Applies to variables (arrays), types, loops
    i. S9 Utilized: To handle representational redundancy
3. Modularity properties:
    a. M1 Parameterized: All inputs accessed via a parameter list
    b. M2 Loosely coupled: Data coupled
    c. M3 Encapsulated: Uses no global variables
    d. M4 Cohesive: The relationships between the elements of an entity are maximized
    e. M5 Generic: Is independent of the type of its inputs and outputs
    f. M6 Abstract: Sufficiently abstract - is no apparent higher-level form
4. Descriptive properties:
    a. D1 Specified: Preconditions and post-conditions provided
    b. D2 Documented: Comments associated with all blocks
    c. D3 Self-descriptive: Identifiers have meaningful names

All the quality-carrying properties that have impact on a structural form are associated with each to support the bottom-up quality implementation. The description is omitted here for space requirements. Dromey does not assign quality-carrying properties to all the structural forms he listed in [43] including system, library, program, meta-program, and user interface. Dromey also presents a minimal subset of all quality-carrying properties for each quality attribute to support the top-down quality design.

Selecting the lowest-level precedence structural form the defect may apply to can perform classification of quality defects. If a quality defect can be associated to more quality-carrying properties, then the one with the higher precedence is to be chosen for defect classification.

In summary, Dromey's model facilitates quality specification, or examination by assigning the quality attributes to quality-carrying properties of the software, and the quality-carrying properties to structural forms. Structural forms and quality-carrying properties are defined on the programming language level; therefore, the model is not language independent and supports in its current form only the imperative programming with the procedural programming paradigm. However, the concepts can also be extended for different programming paradigms and different artefacts including program documentation.

The basic mechanism of the model can be formalized as follows: (1) if each quality-carrying property of a structural form is satisfied, then that structural form will have no quality defect; (2) if a quality-carrying property of a structural form is violated, then it will cause a quality defect in the software.

Quality-carrying properties and structural forms have precedence rules as presented above. If these precedence rules are kept, the model is able to classify software quality defects.

2.1.2.3 Kim and Lee' Software Product Quality Model

Kim and Lee [42] derived a model from the product quality model of the ISO/IEC 9126:2001. The authors determined the relative importance of the six high-level characteristics of the ISO standard from the point of view of the objectives of the project under examination. The order of the relative importance of the six characteristics was computed by applying the Analytic Hierarchy Process [45]. Those characteristics were kept for further investigation, the relative importance of which exceeded a defined

threshold. In their case study they found three such attributes in the particular context: (1) reliability, (2) maintainability, and (3) portability.

They identified internal metrics[5] for static code analysis and assigned these metrics to the three high-level characteristics of the ISO/IEC 9126:2001 model by considering the opinions of experts [42]. The metrics have directly been assigned to the high-level characteristics. Consequently, no intermediate level in the hierarchy with sub-characteristics was defined, i.e. the three characteristics formed categories rather than hierarchies.

The authors also presented the evaluation of a software component to illustrate the use of their model [42]. The critical places for improvement were identified in the component analysed. After performing amendments of the identified quality defects, the evaluation was carried out again, which verified the impact of the corrections.

2.1.2.4 Software Product Quality Model of the ISO/IEC 25010 Standard

The ISO/IEC 25000 standard family supersedes the ISO/IEC 9126 and ISO/IEC 14598 standard families. ISO/IEC 25010:2011 [41] defines a new *quality in use* model and a new software product quality model combining the internal and external models of the ISO/IEC 9126 standard family. However, it keeps the concepts laid down by the previous ISO/IEC 9126-1:2001 standard [40].

Terminology:

- Definition of internal, external view of quality and *quality in use* are taken over from the predecessor ISO/IEC 9126-1:2001 standard [40] but the internal and external software product quality models were combined into one software product quality model.
- *Quality Measure Element* (QME): measurable property of quality defined in ISO/IEC 25021:2007 [25].

[5] Internal metrics make the use of code analysis tools possible as they refer to the intrinsic properties of the software.

- *Quality Measure* (QM): *quality measure element*s and a measurement function to calculate with. It is similar to the term metric in the ISO/IEC 9126-1:2001 standard. Initial list of quality measures was taken over from ISO/IEC TR 9126-2:2003 [46], ISO/IEC TR 9126-3:2003 [47], and ISO/IEC TR 9126-4:2004 [48].
- *Quality attribute*: low-level quality property, in contrast to the ISO/IEC 14598 standard family where the term attribute is used for the high-level quality properties of the ISO/IEC 9126 family.

Concepts:

The software product quality model introduces slight changes in the naming of the six high-level characteristics of the ISO/IEC 9126-1:2001 standard and adds two further high-level characteristics to the previous model: security, compatibility. The whole list of high-level quality characteristics: (1) functional suitability, (2) performance efficiency, (3) compatibility, (4) usability, (5) reliability, (6) security, (7) maintainability, and (8) portability. In addition, the sub-characteristics were partially modified to better reflect realty. However, the sub-characteristics and their definition will not be presented here to comply with the space requirements of the report.

The definitions of the two characteristics that extend the ISO/IEC 9126 product quality model are given [41]:

1. Security: degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization

2. Compatibility: degree to which a product, system or component can exchange information with other products, systems or components, and/or perform its required functions, while sharing the same hardware or software environment

The *quality in use* model defines one additional high-level characteristic with respect to the previous characteristics defined in ISO/IEC 9126-1:2001 and performs slight changes in the naming. The present high-level characteristics of the *quality in use* model

are: (1) effectiveness, (2) efficiency, (3) satisfaction, (3) freedom from risk, and (4) context coverage.

The definitions of the characteristics are [41]:

1. Effectiveness: accuracy and completeness with which users achieve specified goals

2. Efficiency: resources expended in relation to the accuracy and completeness with which users achieve goals

3. Satisfaction: degree to which user needs are satisfied when a product or system is used in a specified context of use

4. Freedom from risk: degree to which a product or system mitigates the potential risk to economic status, human life, health, or the environment

5. Context coverage: degree to which a product or system can be used with effectiveness, efficiency, freedom from risk and satisfaction in both specified contexts of use and in contexts beyond those initially explicitly identified

The new model also defined sub-characteristics that were not part of the previous *quality in use* model. As with the previous models, the sub-characteristics and their definition will not be presented here to comply with the space requirements of the report. The new model description emphasizes the necessity of tailoring the model to the specific objectives of projects.

New members of the ISO/IEC 25000 standard family are expected to be issued in the future: ISO/IEC 25022, 25023, 25024. These standards will suspend the validity of the previous technical reports that define internal, external and *quality in use* metrics: ISO/IEC TR 9126-2, 9126-3, 9126-4.

These models are language-independent and independent of programming paradigms.

2.1.3 Summary

In the current frameworks [41], [40], [49], [39] when the traceability property is present it refers to requirement traceability, i.e. how requirements can be followed during the development of the software. Some *quality metric*s and *measures* defined in ISO/IEC 9126-2:2003, ISO/IEC 9126-3:2003, and in ISO/IEC 25021:2007 show overlaps and similarities to execution tracing but their definitions are ambiguous and difficult to approach from a practical point of view. This is the case, for example, of the computation of the ratio of the number of diagnostic functions and the number of necessary diagnostic functions as the latter implicates vagueness and subjective judgement to a great extent. Such metrics are associated with the Internal and External Analysability sub-characteristic of the characteristic Maintainability: (1) Activity Recording, (2) Readiness of Diagnostic Functions, (3) Audit Trail Capability, (4) Diagnostic Function Support, (5) Failure Analysis Capability, (6) Failure Analysis Efficiency, and (7) Status Monitoring Capability [46], [47] as summarized in Table 2 below.

The findings of this section show that execution tracing is not appropriately considered by the present software product quality frameworks, which answers the first research question in section 1.2 Research Questions. In the next section execution tracing is introduced and its relation to analysis of software errors investigated.

External and Internal Analysability Metrics from the ISO/IEC 9126-2 and ISO/IEC 9126-3 Standards										
	Metric Name	**Purpose of the Metric**	**Method of Application**	**Measurement, Formula and Data Element Computations**	**Interpretation of Measured Values**	**Metric Scale**	**Measure Type**	**Input to Measurement**	**ISO/IEC 12207 Reference**	**Target Audience**
External Analysability Metrics	Audit Trail Capability	Can the user identify specific operations which caused failure? Can the maintainer easily find specific operations which	Observe behaviour of users or maintainers who are trying to resolve failures.	X=A/B A: Number of data actually recorded during operation; B: Number of data planned to be recorded, which is enough to monitor status	0<=X<=1 The value is closer to 1.0 it is the better.	Absolute	A: Count B: Count X:Count/Count	Problem resolution report; Operation report	5.3 Qualification testing; 5.4 Operation 5.5 Maintenance	Developer Maintainer Operator

		caused failure?		of the software in operation						
	Diagnostic Function Support	How capable are the diagnostic functions to support causal analysis? Can the user identify the specific operation which caused failure? (The user may be able to avoid encountering the same failure with alternative operations.) Can the maintainer easily find the cause of failure?	Observe behaviour of users or maintainers who are trying to resolve failures using diagnostic functions.	X=A/B A: Number of failures which maintainers can diagnose using diagnostic functions to understand cause-effect relationship B: Total number of registered failures	0<=X<=1 The value is closer to 1.0 it is the better.	Absolute	A: Count B: Count X:Count/Count	Problem resolution report; Operation report	5.3 Qualification testing; 5.4 Operation 5.5 Maintenance	Developer Maintainer Operator
	Failure Analysis Capability	Can the user identify specific operations which caused the failures? Can the maintainer easily find the cause of failure?	Observe behaviour of users or maintainers who are trying to resolve failures.	X=1-A/B A: Number of failures of which causes are still not found; B: Total number of registered failures	0<=X<=1 The value is closer to 1.0 it is the better.	Absolute	A: Count B: Count X:Count/Count	Problem resolution report; Operation report	5.3 Qualification testing; 5.4 Operation 5.5 Maintenance	User Developer Maintainer Operator
	Failure Analysis Efficiency	Can the user efficiently analyse the cause of failure? (Users sometimes perform maintenance by setting parameters.) Can the maintainer easily find the cause of	Observe behaviour of users or maintainers who are trying to resolve failures.	X=Sum(T)/N[6] T: Time; N: Number of failures	0<=X The value is the shorter it is the better.	Ratio	T: Time	Problem resolution report; Operation report	5.3 Qualification testing; 5.4 Operation 5.5 Maintenance	Developer Maintainer Operator

[6] The standard articulates some recommendations regarding the time and number of failures to be considered.

		failure?								
	Status Monitoring Capability	Can the user identify specific operations which caused failure by getting monitored data during operation?	Observe behaviour of users or maintainers who are trying to get monitored data recording status of software during operation.	X=1-A/B A: Number of cases for which maintainers or users failed to get monitored data; B: Number of cases for which maintainers or users attempted to get monitored data recording status of software during operation.	0<=X<=1 The value is closer to 1.0 it is the better.	Absolute	A: Count B: Count X:Count/Count	Problem resolution report; Operation report	5.3 Qualification testing; 5.4 Operation 5.5 Maintenance	User Developer Maintainer Operator
Internal Analysability Metrics	Activity Recording	How thorough is the recording of the system status?	Count the number of items logged in the activity log as specified and compare it to the number of items to be logged.	X[7]=A/B A: Number of implemented data logging items as specified and confirmed in reviews B: Number of data items to be logged defined in the specifications	0<=X<=1 The closer the value is to 1, the more data are provided about the system status	Absolute	A: Count B: Count X=Count/ Count	Value A comes from the review report. Value B comes from the requirement specifications.	Verification Joint review	User Maintainer
	Readiness of Diagnostic Functions	How thorough is the provision of diagnostic functions?	Count the number of the diagnostic functions specified and compare it to the number of the diagnostic functions required in the specifications.	X[8]=A/B A: Number of diagnostic functions as specified and confirmed in reviews B: Number of diagnostic functions required	0<=X<=1 The closer the value is to 1, the better is the implementation of diagnostic functions	Absolute	A: Count B: Count X=Count/ Count	Value A comes from the review report. Value B comes from the requirement specifications.	Verification Joint review	User Maintainer

Table 2 External and Internal Analysability Metrics, extract from [46], [47]

[7] It is necessary to covert this value to the interval [0;1] if making summarizations of attributes.
[8] It is necessary to covert this value to the interval [0;1] if making summarizations of attributes.

2.2 Execution tracing

As mentioned in Chapter 1, execution tracing, and logging are frequently used as synonyms in software technology; however, the first one serves the software developers to localize errors in applications, while the second one contributes to administration tasks towards checking the state of software systems. Thus, the depth of information they offer deviates.

Monitoring techniques can be approached from several points of views. The following very practical approach for the classification of monitoring techniques is described by Cohen in [50]: (1) step-by-step execution with a debugger, (2) basic execution trace messages, (3) using a dynamic proxy, (4) using a run-time profiler, and (5) aspect-orientation. Cohen's description refers to the Java programming language therefore Table 3 presents the points slightly extended to comprise of several programming languages.

	Advantages	Disadvantages
Step-by-step execution with a debugger	• Usually simple to setup • Access to all data • No code modification required • Flexible: investigation of the critical places only	• Investigating large part of code is impractical • Complex with event handlers • Investigation requires source code
Basic execution trace messages	• Flexible • Practical for event handlers • Speedy execution in comparison to debugging • Can collect the exact data of interest	• Code modification necessary • Implementation effort • Code pollution • Run-time overhead
Using a dynamic proxy (programming language specific)	• One central place for trace messages • Practical for event handlers • Simple to add and remove	• Feasible with public interface methods only • Code modification • Not flexible
Run-time profiler	• No code modification necessary • No source code necessary • Configurability	• Quality profilers are expensive • Learning curve can be long • Usually big performance degradation • Method arguments are not available with each tool
Aspect-orientation (programming language specific)	• Easy to add • Flexible • Access to all data • No code modification necessary	• Can require a rebuild • Learning curve

Table 3 Practical Approaches of Monitoring Techniques

In addition to the above practical approach a more theoretical perspective to monitoring with a higher level of abstraction is provided in the below Table 4. This classification is carried out by the author and based on the literature. It shares some common points related to programming paradigms with Cohen's view as illustrated in the last column of the table.

	Major Characteristics	Links to Table 3
Online Monitoring Techniques	• The application needs to be being executed[9] • The output of the debugger or profiler is not available in a persistent way	• Debugger • Runtime profiler
Offline Monitoring Techniques	• The application needs to be executed at least once but it does not need to be executed during the analysis • Data are provided for offline analysis	• Basic execution trace messages • Dynamic Proxy • Aspect-orientation
Dynamic Monitoring Techniques	• The application needs to be executed at least once • Ensure the analysis of one concrete execution with concrete values of variables • The outcome is valid only for the concrete thread of execution investigated	• All of the above
Static Monitoring	• The application does	• N.a. Table 3. Does not

[9] Debuggers usually make possible the analysis of core files offline if the application was compiled with special compiler flags to keep symbol information. However, major use of debuggers is targeted at online analysis of executing applications.

Techniques	not need to run or being executable • Ensure the analysis of all possible executions with the types and ranges of variables • The outcome has general validity over all inputs	show static analysers as: compilers, pattern checkers or special tools developed with the symbiosis of static and dynamic analysis [8], [7]

Table 4 Monitoring Techniques

Online analysis techniques require the application to be running and available to reproduce the error. Offline analysis techniques collect the runtime data in a persistent storage, i.e. in a file or a database, and therefore do not require the software to be running at the time of the analysis.

Both the debugger and the profiler are the online monitoring tools; moreover, they collect data about the execution in memory not in a persistent storage facilitating an immediate analysis. Profilers are also available which can store the data collected during the execution for offline analysis [51], [52]. Basic execution trace messages, dynamic proxy, and aspect-oriented approach are offline monitoring techniques because the use case in question is tested and then the data collected during the execution is analysed afterwards. These three techniques fit well with the definition of execution tracing given above.

Furthermore, monitoring techniques can be classified as static or dynamic. Dynamic monitoring techniques can be applied only if the software is built and executable in contrast to static monitoring techniques. Each technique has its own particular advantage. Static analysis can produce sound results however with general properties. Compilers of programming languages also perform this kind of analysis. These checks are accurate and valid over all possible execution. However, the static analysis is not precise, it cannot state whether a particular variable value with a specific thread of execution will fail or not. Techniques for checking the source code of applications for bug patterns are also known [6]. In contrast, dynamic monitoring examines the particular

execution of the program by observing its behaviour, which is precise but the results obtained are not applicable to all possible inputs. The literature promotes the synergic use of these techniques [7], [8], [9].

Execution tracing mechanisms comprise of collection of data of interest about the program state and the path of execution for offline analysis. Thus, execution tracing belongs to offline, dynamic monitoring techniques, i.e. testing, and the investigation of live systems all of which are integral parts of the maintenance activities.

Several tools leverage the dynamic monitoring techniques to localise errors, which usually follow one of the patterns given below [18]:

1. Dynamic dependence analysis of the failing program execution to scrutinise the statements of the source code, named also slicing criterion in the literature, which are related to the failure. This technique exploits program slicing to identify the relevant statements for effective human analysis [53], [54]. The technique is also used in static monitoring [55]. Both variants apply graph computations to calculate which statements are reached but the dynamic one considers the failing execution path.
2. Comparison of the failing program execution with all error free executions, which is mainly applied for model checker traces i.e. the software or hardware are described by a model in a mathematical language, in addition to the specification and it checks whether the model satisfies the specification [56]. The algorithm investigates correct and incorrect test runs and identifies the execution trace parts that do not intersect with the correct runs.
3. Comparison of the failing program execution with a program execution that does not manifest the error to localise. The effectiveness of the error localisation depends on the similarity of the two executions, therefore crucial issue is raised by selecting the appropriate successful execution from a pool to which the failing one could be compared [57].

The use of online analysis tools seems to be an attractive choice to substitute execution tracing but, debugging or profiling are not necessarily feasible when (1) applications perform process control, (2) the error is related to parallel processing and race

conditions, or (3) performance problems need to be analysed [14], [15]. In the case of distributed, multithreaded applications execution tracing is the only adequate instrument to help with the error analysis as Laddad states in [17]. In the case of embedded applications, which have no user interface, only by means of execution tracing can the developer or system maintainer answer such questions as what the application was doing [15].

Consequently, the offline and online monitoring techniques are not competitors but complement each other. While the application of online monitoring techniques does not require code changes at the development time, offline monitoring usually does. The execution tracing mechanism needs to be designed with the application to satisfy the data demands for maintenance activities with regard to other non-functional requirements, including application performance, to find a balance among contrasting expectations.

In the scope of this review the following aspects of execution tracing are investigated:

1. Insertion of the code for execution tracing
2. Programming paradigms
3. Application domain of the application to be traced
4. Trace output

2.2.1 Insertion of the Code for Execution Tracing

The insertion of the code that dumps the necessary data of the current thread of execution can be inserted (1) manually or (2) automatically in a declarative manner. Manual insertion of the trace code is the simplest form of execution tracing. However the application code has a strong dependency on the code of execution tracing. In addition, it is an effort-intensive, monotonic activity where it is easy to skip necessary places [17].

Automatic insertion of execution trace code can be performed depending on the underlying programming paradigm, which is discussed in the next section. In the case of automatic trace code insertion, rules identify the places in the application code where execution trace needs to be inserted [17].

2.2.2 Programming Paradigms

Different programming paradigms facilitate different approaches to execution tracing. Only the most frequently used programming paradigms are included in this investigation. TIOB indexes programming languages based on their popularity, which is updated each month [58]. The index is based on popular search engine statistics. Table 5 below shows that the first five programming languages in the ranking either belong to the procedural or to the object-oriented paradigm [59].

Position Dec 2012	Position Dec 2011	Delta in Position	Programming Language	Share Dec 2012	Delta in Share to Dec 2011	Position Dec 2007	Position Dec 1997	Position Dec 1987
1	2	Up	C	18.696%	+1.64%	2	1	1
2	1	Down	Java	17.567%	+0.01%	1	4	-
3	5	Up	Objective-C	11.116%	+4.31%	57	-	-
4	3	Down	C++	9.203%	+0.95%	5	2	6
5	4	Down	C#	5.547%	-2.66%	8	-	-

Table 5 The Most Popular Programming Languages and Their Long-term History, source: [58]

Another investigation performed by Bergman in [60] on the area of semantic tools, including 1010 tools in nearly 50 tool categories with 83% in the open source segment, shows similar results in rating (1) Java, (2) JavaScript, and (3) Don't Know categories on the first three places. JavaScript indicates an exception on this particular area in comparison to the general shares. JavaScript is an object-based, functional language [61]. Other sources state it as object-oriented [62]. However, the functional programming paradigm is not covered in the present investigation because only a negligible number of applications utilise this paradigm, as the statistics in Table 6 shows; moreover, the program state, i.e. the values of assigned variables or values of global variables, does not change and operations have no side effects [63]. Consequently, execution tracing

has less importance in applications where the programming paradigm prevents to change the state of the program.

Category	Ratings Dec 2012	Delta Dec 2011
Object-Oriented Languages	58.5%	+2.1%
Procedural Languages	36.9%	-0.2%
Functional Languages	3.2%	-1.3%
Logical Languages	1.4%	-0.6%

Table 6 Rating of Programming Paradigms, source: [58]

Manual trace code insertion is available with all programming paradigms. However, aspect-oriented programming [17] offers automatic insertion of the code for execution tracing. The most popular programming languages make possible to use an aspect-oriented approach for trace code insertion or even for significantly more complex activities [64], C from the procedural domain [65], [66] and Java, C++, and C# from the object-oriented domain [67], [68], [69]. Even the publications arguing the general use of aspect-orientation admit its applicability for execution tracing [70]. In addition, aspect-oriented functionality can be implemented in most programming languages by means of language preprocessors [71]. Modern functional languages like Scala [72] and F# [73] also show object-oriented features and are called functional and object-oriented languages.

Aspect-oriented programming facilitates the localisation of functionally identical code in a separate module and its insertion, either at compilation time or at run time, into places that are identified by rules. The functionally identical code segments scattered across the application are called "*crosscutting concerns"*, the modules containing the collected crosscutting concerns are called "*aspects"*, the crosscutting concerns localised in the aspect are named "*advices",* meanwhile the rules determining where to add the advices are named "*join points"* in the aspect-oriented terminology [17].

Execution tracing typically results in crosscutting concerns. The UML diagrams below depict this functionality with both conventional, i.e. manual insertion of execution trace, and aspect-oriented approach, i.e. automatic insertion of the code for execution tracing.

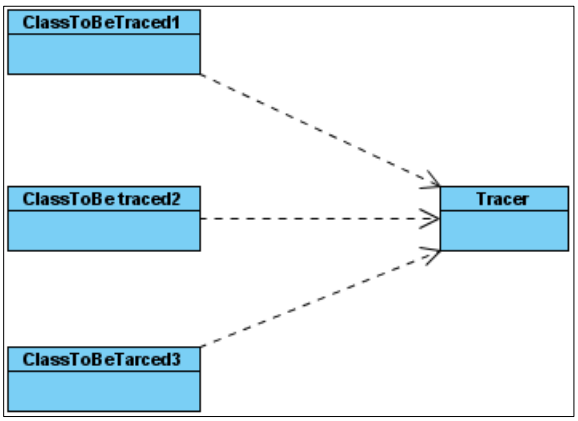

Figure 3 Class Diagram of Conventional Execution Tracing

The schematic illustration shows that the classes containing the functionality of the application depend on an external class the functionality of which lies purely in producing execution traces. The Tracer class can be implemented by using the Singleton Design Pattern [74] or it can also be associated to each application class resulting in a stronger relationship.

This implementation is disadvantageous as the classes containing the functionality of the application depend on or are associated to a functionally unrelated class. Moreover, the trace method invocations are manually inserted in the application classes causing significant implementation effort. If the Tracer class needs to undergo changes including also the signatures of the trace methods, it will implicate changes in all of the classes of the application.

The Java code snippet given below shows a very simple implementation for an execution trace line that needs to be inserted in each method to collect the method arguments and their values:

```
if(Trace.isTraceOn()){
  Trace.trace("B: "+"T:"+Thread.currentThread().getId() +
  "<method name + arguments with values>");
}
```

Legend:

- The `Trace` class localises the execution tracing related functionality
- The condition check in the if-branch tests whether the trace message needs to be written
- The `Trace.trace()` method writes the trace message with all necessary data including thread ID, method name, method arguments

In contrast, the aspect-oriented approach turns the dependency as illustrated on the UML diagram below.

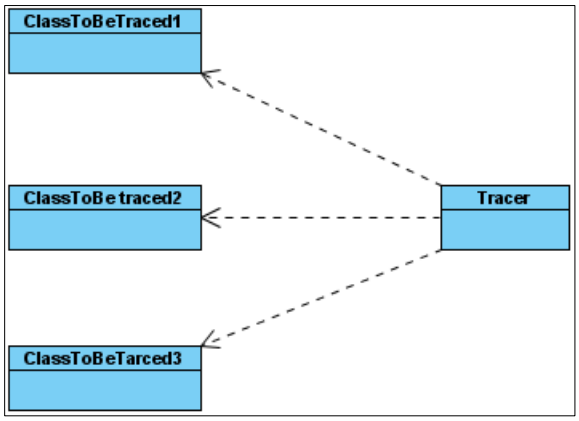

Figure 4 Class Diagram of an Aspect-Oriented Execution Trace Implementation

As the diagram depicts, the classes of the application do not depend on the external Tracer class. This setting ensures that changes in the tracer functionality including changes in the signatures of the trace methods, will not implicate changes in the application classes. Further advantage is achieved by the rule-based trace code insertion, which decreases the implementation effort and increases consistency of tracing, i.e. none of the specified code locations are omitted.

```
before() : traceMethods(){
  if(Trace.isTraceOn()){
    StringBuffer args = new StringBuffer();
    Signature sig = thisJoinPointStaticPart.getSignature();
    args.append(BEGIN).append("T:" +
    Thread.currentThread().getId() + "  "
    ).append(sig.toShortString()).append( " A:{");
args.append(getMethodArgs(thisJoinPoint.getArgs())).append("}
    ");
    Trace.trace( args.toString());
```

Legend:

- `before()` is an advice that is associated with the call of `traceMethod()`. The point cuts that define the join points are not part of the code snippet for simplifying the illustration
- The `Trace` class localises the execution tracing related functionality
- The condition check in the if-branch tests whether trace needs to be written
- The code in the if-branch collects all data for the trace message: method's signature, thread ID, arguments
- The `Trace.trace()` method writes the trace message with all necessary data including thread ID, method name, method arguments

The above code snippet in AspectJ shows the same functionality as the previous listing, however it is inserted automatically at each join point and it also collects the method arguments without the need to change the application classes manually. The code snippet is carried out before the code at the join point runs.

2.2.3 Application Domains

The desired features of execution tracing strongly depend also on the application domain. Beside sharing common criteria, significantly different expectations also need to be satisfied by the tracing mechanism of an embedded application in comparison to the tracing mechanism of a distributed application responding to several thousand users each second. However, the goal of execution tracing, in spite of the differences, is to collect informative data about the program run. In this section the particularities of execution tracing are examined in the following domains:

1. Standalone applications - An application, which runs alone on a host without using network connections and without communicating with other applications, is named a standalone application. A standalone application, which runs on a

special hardware, is called an embedded application regardless of using network connections or not.

2. Distributed applications - An application, the functionality of which is distributed among several processes or even several hosts, is called a distributed application. Distributed applications can involve considerable amount of network traffic. On the basis of the software architecture a distinction is made between client-server architectures and N-tier architectures.

    a. Client-server architectures

    If the application architecture can be divided on two parts: (1) an application component to deliver services to (2) another application component that leverages these services; moreover, if no other application layer or tier separate the serving and the leveraging component, then the architecture can be designated as a client-server architecture. The serving component is named server, while the leveraging component is named client.

    b. N-tier architectures

    N-tier architectures form similar architectures to client-server architectures but the functionality they comprise of is divided in tiers. The tiers interact with each other to ensure the operation of the application as published in the Tier Design Pattern [75]. This architecture is mainly used in large applications designed usually with the following tiers: (1) client-tier, (2) web-tier, (3) business-tier, (4) integration-tier, and (5) data-tier. Application servers complying with the EJB specification [76] offer services to support this design [77].

3. Embedded applications - All applications that run on a special hardware rather than on a standard application host are called embedded applications. The variety of these applications spans from medical devices as X-rays, ECGs over home electronics as blue-ray players, televisions to control applications in cars, and airplanes to mention only a few. These applications have no user interface in several cases and their hardware possesses characteristics significantly different from application hosts.

Execution tracing needs to deliver information for the software maintainers or developers to determine the cause of an error or follow the actions of a thread in the application to determine internal behaviour. Concluding from the particularities of the above architectures, the following requirements are desirable towards execution tracing to support the analysis:

Standalone Application:

1. Tracing the execution path in the application method names, arguments, and return values including thread IDs if the application is multithreaded
2. Configuration of execution tracing, so that the application could produce trace without the need of rebuild
3. Handling of the output file of execution tracing

Distributed Applications, Client-Server Architecture:

1. Tracing the execution path in the application method names, arguments, and return values including thread IDs if the application is multithreaded
2. Configuration of execution tracing, so that the application could produce trace without the need of rebuild
3. Handling of the output file or socket of execution tracing
4. Tracing the data between the client and the server

Distributed Applications, N-Tier Applications:

1. Tracing the execution path in the application method names, arguments, and return values including process IDs, thread IDs if the application is multithreaded
2. Configuration of execution tracing, so that the application could produce trace without the need of rebuild
3. Handling of the output file or socket of execution tracing
4. Tracing the data between the component boundaries
5. Tracing access to the database, including DML commands

Embedded Applications:

1. Tracing the execution path in the application method names, arguments, and return values including thread IDs if the application is multithreaded
2. Configuration of execution tracing, so that the application could produce trace without the need of rebuild
3. Handling of the output file of execution tracing with regard to specific storage requirements
4. Use of the interfaces to access the trace file written
5. Leverage of possible hardware support for execution tracing [78]

The above expectations towards execution tracing follow from the particularities of the above architectures; they are not necessarily the optimal requirements for execution tracing.

2.2.4 Execution Trace Output

For storing the output of execution tracing different options are available: storing data (1) in binary or ASCII file, (2) in database, or (3) transferring data over a network socket [79], [80], [81], [82], [83], [84]. This question is also dealt intensively with professional forum posts among the software developers [85], [86], [87], [88].

The most widespread mechanism to store data of execution tracing is the use of trace files. If the trace is written in binary format, storage space can usually be spared in comparison to text format. The text files written in XML, which makes the output even more expansive, offer additional possibilities for processing including: (1) trace visualisation [20], (2) querying the trace [89], and (3) mining execution traces [90]. Querying execution traces is implemented by query processors, which form a layer of abstraction in the software architecture above access to the execution trace files, and that can accept commands in a query language [91]. This abstraction layer can also be implemented for arbitrary execution trace outputs, although changes in the proprietary execution trace files can implicate significant changes in the parser that processes these

files. ASN.1 facilitates binary coding of the execution traces in a standard format, which can also be converted to XML [92].

2.2.5 Problems with Execution Tracing

The notion of execution tracing is simple but its optimal implementation in the context of a particular application is difficult. This difficulty is briefly expressed by the contrasting non-functional requirements towards execution tracing:

1. Consistent code coverage - If execution trace is written, it should be written consistently with regard to the trace level, e.g. in each method. If the trace file is produced with inconsistent code coverage, the developer or maintainer will have difficulties to decide whether the missing trace entry means that everything is all right it is only not traced or whether a certain method was not invoked, some actions not done.

2. Informative trace to support maintainability - The execution trace should be as informative as possible to support the error analysis that results in the expectation of having to trace much data.

3. Performance - If execution tracing is switched on in an application, it should have only an acceptable negative impact on the performance and should not risk the normal operation of the application. It is extremely critical when analysing live systems of enterprises.

If requirements 1 and 2 are satisfied, then it will have a big negative impact on the performance of the application. Requirement 3 articulates that as little performance impact must be caused as possible.

Furthermore, switching on execution tracing might change the race conditions of parallel running threads due to the effect on performance in the application, which can also result in not being able to reproduce the error in extreme conditions. However, the performance effect of execution tracing is significantly less than that of a debugger.

2.2.6 Summary

The notion execution tracing is independent of programming paradigms and programming languages but different languages offer different potential to adapt it to the context of an application. The most frequently used languages, C, C++, and Java, have the ability to apply the aspect-oriented extension.

Execution traces serve as basis for dynamic analysis techniques which are inevitable in distributed systems and in applications with multiple running threads [57], [53], [54], [17]. Execution tracing opens also analysis facilities in the embedded domain. This is extremely important for embedded applications that do not possess a user interface.

The requirements towards execution tracing based on the particularities of the software architectures are not necessarily consistent and the expectations towards the trace output are conflicting because consistent and informative tracing has a negative impact on the performance.

In addition, this section shows that execution tracing quality of software systems influence the property analysability of the overall software quality. Consequently, execution tracing should be considered as a subordinate quality property of analysability in the existing hierarchical software product quality frameworks. This section contributes to the second research question in 1.2 Research Questions. How execution tracing quality could be incorporated in the present software product quality frameworks is investigated in detail in the following chapter 3 Extension Possibilities of Present Quality Frameworks.

3 Extension Possibilities of Present Quality Frameworks

This chapter presents how software product quality frameworks could be extended to contain execution quality to answer the second research question in 1.2 Research Questions. The results were submitted to the journal APH for publication in April 2013.

The software product quality frameworks presented in previous chapters mainly follow two basic approaches for describing software product quality: (1) the hierarchic approach depicted by the ISO/IEC 9126 and ISO/IEC 25000 standard families which have their roots in the early models; and (2) the non-hierarchic approach described by Dromey. The other frameworks tailor the first approach or its predecessors to the specific context of use. All the frameworks presented are the result of empirical research, which offers possibilities for changes and tailoring them.

For this reason we investigate the extensibility of the ISO/IEC 9126, ISO/IEC 25010 quality frameworks and the framework defined by Dromey.

This investigation includes (1) where the description of execution tracing quality could be placed in the existing models, and (2) what methods the complete frameworks offer to describe execution tracing quality, including the reflection of subjective uncertainty. Nevertheless, the property illustrating execution tracing quality also needs to be able to express the quality of execution tracing as a standalone model without the frameworks presented to offer its use on its own, which relates to the third research question in 1.2 Research Questions and the third research objective in 1.3 Research Objectives.

3.1 Extension of ISO/IEC 9126 Framework

The ISO/IEC 9126 standard allows adaptations of the software product quality model it defines as not all of the high-level quality characteristics and their sub-characteristics are necessary to address the quality needs of each project. Although the product quality model definition in ISO/IEC 9126-1:2001 is superseded by ISO/IEC 25010:2011, the model exists and is in use; moreover, the *quality metric*s remain and will not be

superseded until ISO/IEC 25022, and 25023 are issued. Consequently, this product quality model is included in the investigation.

Taking into account the concepts and terminology of the present software product quality framework, the following steps are necessary for its extension:

- Defining which characteristics and sub-characteristics can locate the execution tracing related quality description.
- Defining internal and external metrics related to the quality property of execution tracing. The internal and external metrics have to correlate and the internal metrics need to have predictive validity towards the external metrics.

3.1.1 Extension Method

Execution tracing quality significantly influences the effort needed for error analysis. This identifies by its nature a property that belongs to maintainability or any of its sub-characteristics.

The high-level characteristic maintainability consists of five sub-characteristics (see Figure 1): (1) analysability, (2) changeability, (3) stability, (4) testability, and (5) maintainability compliance. With regard to the goal of execution tracing the sub-characteristic analysability offers a logical point to link to it because it encompasses all metrics which describe how the software or its behaviour can be analysed [40], [46], [47].

Once finding the location in the hierarchy where execution tracing can be located, appropriate metrics need to be defined as explained above. Because the description of execution tracing quality needs to be able to describe the quality of execution tracing as a standalone model, it is not recommended to define different metrics to the ones that exist as it would create a dependency on the ISO product quality framework. If one new metric is introduced, the execution tracing quality model could easily be linked to the ISO framework without developing dependency on it. For this reason we define an internal metric and an external metric keeping the naming conventions of the standard: (1) Internal Execution Tracing Capability Metric and (2) External Execution Tracing Capability Metric (see Figure 5).

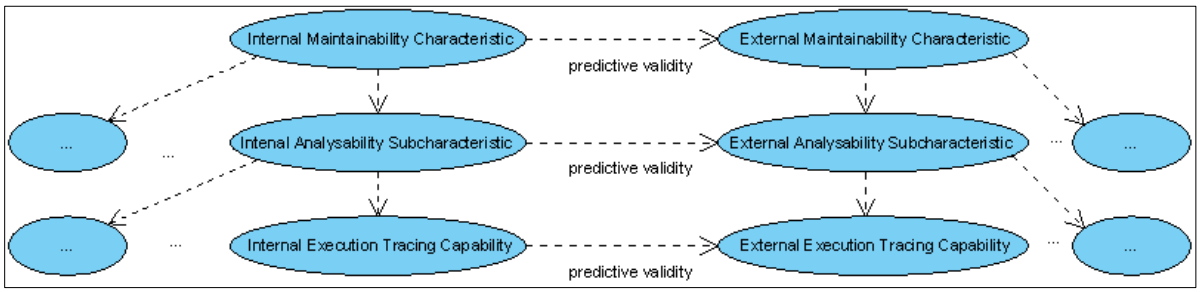

Figure 5 Extending ISO/IEC 9126 with Execution Tracing Capability

The definition of the metric requires appropriate identification of the inputs and the method description how the metric can be calculated from the inputs. According to the terminology of the standard ISO/IEC 25021:2007, the inputs of the metrics are called *quality measure element*s.

3.1.2 Benefits

The expected benefit of this extension is to consider execution tracing quality when the complete software product quality is assessed. In addition, the subjective uncertainty of the inputs, the *quality measure element*s of the metrics, can also be reflected by the mathematical computations, which can involve fuzzy logic as shown in the following chapter.

No detrimental effects of this extension on the framework are known. The standard also encourages tailoring the software product quality model to specific needs of projects. Consequently, the extension is in accordance with the philosophy of the ISO/IEC 9126 standard family.

3.1.3 Existing Extension Results

Carvallo and Franch [93] point out that software evaluation is necessary from a technical point of view, although their examination also shows that non-technical factors related to licensing and supplier characteristics are even more important in case of commercial off-the-shelf (COTS) products. The authors propose to extend the ISO/IEC 9126 standard family to include non-technical factors in a uniform way. In their proposal the authors keep the hierarchical structure of the standard and define three high-level characteristics: (1) supplier, (2) costs, and (3) product, which they decompose in fifteen

sub-characteristics with the third level of the hierarchy even further decomposed resulting in more than two hundred non-technical quality properties. They validated the extension of the model on different projects in the telecommunication industry on which they provide a brief summary in [93].

3.2 ISO/IEC 25010 Framework

As mentioned before, the software product quality framework defined in ISO/IEC 25010:2011 revised the software product quality framework of ISO/IEC 9126-1:2001. This new standard kept the philosophy of the previous model. The changes in the model hierarchy did not affect the node analysability below maintainability. Thus, the extension point does not change in comparison to the ISO/IEC 9126-1 framework.

Nevertheless, potential combination of the internal and external software product quality models in ISO/IEC 25010:2011 needs to be considered. As ISO/IEC 25022 and 25023 are not issued to supersede ISO/IEC 9126-2:2003 and 9126-3:2003, the separation of internal and *external quality* views is also a viable option.

3.2.1 Extension Method

The extension possibilities described at ISO/IEC 9126-1:2001 can be used with the revised software product quality model this new standard introduced. The measures Internal Execution Tracing Capability and External Execution Tracing Capability were merged into a single Execution Tracing Capability measure to comply with the combined internal and external model (see Figure 6) and consequently it possesses both internal and *external quality measure element*s.

Special attention needs to be paid to the definition of the inputs of execution tracing quality and the description of the computation by which the quality of execution tracing can be computed. Definitions of new *quality measure*s and *quality measure element*s are formalised and defined in the standard ISO/IEC 25021:2007.

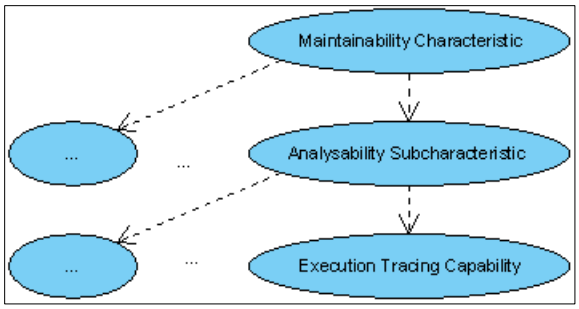

Figure 6 Extending ISO/IEC 25010 with Execution Tracing Capability

3.2.2 Benefits

As with the previous ISO/IEC 9126-1:2001 extension, the primarily expected benefit is to consider execution tracing quality when the complete software product quality is assessed. The subjective uncertainty of the *quality measure element*s of the defined *quality measure*s can also be reflected by mathematical computations including fuzzy logic as presented in the next chapter.

In comparison to extending the ISO/IEC 9126-1:2001 framework, a further advantage in this case is that the dependency on the ISO-framework is reduced to just one measure, thus it supports the standalone application of the quality model describing execution tracing to greater extent.

No detrimental effects of this extension on the framework are known. The standard also declares that tailoring the software product quality model of ISO/IEC 25010 to specific needs of projects is a must, i.e. fair to conclude that tailoring is more emphasised in the revised standard than in its predecessor. Consequently, the extension is in accordance with the philosophy of the standards.

3.2.3 Existing Extension Results

No extension attempts of ISO/IEC 25010 were found in the literature.

3.3 Dromey's Framework

For each section the terminology of the software product quality model in question is used. Dromey's model applies significantly different terminology from the previous models presented. As an example the term "attribute" needs to be mentioned. The terminology is introduced in detail in section 2.1.2.2 Software Product Quality Model of Dromey.

Dromey handles three primary sets of entities in his framework without introducing hierarchies. The relationships of these sets depict the quality requirements and the criteria for assessment. The set of high-level *quality attribute*s contains maintainability making the identification of the category to which execution tracing quality needs to be assigned obvious. Therefore the set of high-level *quality attribute*s need to undergo no changes. Consequently, extension possibilities for sets of quality-carrying properties and structural forms need to be examined.

3.3.1 Extension Method

Because all the structural forms define programming language-level constructs in the original description, higher-level structural forms are also necessary to include entities on component-level or application-level. New quality-carrying properties need to be introduced in the framework to describe the input variables of execution tracing in a binary manner to show whether the property is present in the application under investigation or not.

Execution tracing related quality-carrying properties can be linked to the new structural forms and to the high-level attribute maintainability in order to establish relationships. Then following the bottom-up approach introduced in the model description, the optimal relationships for each structural form need to be defined which guarantee the good quality; moreover, to support the top-down approach the optimal relationships need to be defined between the quality attributes and the quality-carrying properties. These profiles give a measure that can be compared to the actual software under investigation to diagnose quality defects or to set quality targets.

The original definition of the framework only considers the procedural programming paradigm, which has to be kept otherwise the present model needs to be reworked significantly to create new quality-carrying properties. The model's basic principles also facilitate the accommodation to other programming paradigms with the introduction of new quality-carrying properties and structural forms to define new relationships.

3.3.2 Benefits

As with the ISO/IEC 9126-1:2001 extension described before, the primarily expected benefit is to consider execution tracing quality when the complete software product quality is assessed.

A potential detrimental effect to mention is the high number of new relationships between the necessary quality-carrying properties and structural forms, when more than programming language level assessment is necessary. On the other hand, the model only supports the procedural programming paradigm, so extension to further programming paradigms would implicate additional quality-carrying properties resulting in additional relationships between the *quality attribute*s, structural forms and quality-carrying properties. The high number of possible relationships to process during quality assessment could make it unmanageable.

3.3.3 Existing Extension Results

No extension attempts of Dromey's framework were found in the literature.

3.4 Discussion

The frameworks investigated in the previous section allow extensions to include execution tracing quality but their implementations differ significantly.

Dromey's model only describes code-level constructs and their quality considering the procedural programming paradigm. The principle of the model, however, can also be applied for higher-level constructs and additional programming paradigms. From the point of view of execution tracing, procedural programming does not cause difficulties although the usability of the model would significantly be reduced if no other programming paradigms could be represented. To encompass additional programming paradigms and higher-level artefacts, Dromey's model requires considerable amounts of new quality-carrying properties and new structural forms. The high number of elements in both sets enhances the number of combinations through which relationships need to be expressed. Consequently, the execution tracing quality can be described at the cost of introducing more complexity in the model. In addition, the direct assignment of binary quality-carrying properties to high-level attributes leaves no room to uncertainty computations.

In contrast, ISO/IEC 9126 and 25010 offer an extension possibility and a sub-characteristic to which the description of execution tracing can be linked: maintainability and its analysability sub-characteristic. Linking is simple and requires considerably less effort than incorporating the illustrated changes in Dromey's model. Moreover, the *quality measure* or *metric* definitions complying with the standards allow the use of mathematical functions, by which subjective uncertainty computation can also be implemented.

If execution tracing quality were to be described by means of Dromey's framework, then it could not be used as an independent model because the framework requires a specific implementation. On the contrary, linking the description of execution tracing quality to the ISO/IEC software product quality frameworks facilitates its existence as an independent model.

ISO/IEC product quality models are more widespread than Dromey's model and they are known to a larger audience, as evidenced by the high number of publications relating to these standards, moreover by the models based on the ISO/IEC framework presented in the previous chapter. In addition, execution tracing quality can be encompassed with significantly less effort in the ISO/IEC standards than in Dromey's model.

3.5 Summary

Execution tracing is an important property that needs to be considered in quality frameworks to truly reflect the overall view of software product quality. Dromey's model allows extensions to include execution tracing quality although it requires significant changes in the present model. The model's philosophy does not support mathematical operations on quality-carrying properties and, therefore implementing subjective uncertainty computations is infeasible at present.

Software product quality frameworks of the ISO/IEC standards allow extensions and have a defined method to do so. Moreover, they also offer a natural linking point for execution tracing quality with the analysability sub-characteristic of maintainability. They can also allow mathematical computations that make the implementation of subjective uncertainty computations possible as shown in the next chapter.

In conclusion, execution tracing quality should be linked to the software product quality framework of the ISO/IEC 25010 standard with the observation of the rules for defining *quality measure*s and *quality measure element*s. This would facilitate the consideration of execution tracing quality when the whole software product quality is assessed; furthermore, it would ensure a framework for incorporating the impacts of subjective uncertainty resulting from the quality measurement process.

This chapter provides answers on what is necessary to incorporate execution tracing in the present software product quality frameworks as posed by the second research question in section 1.2 Research Questions.

4 Pilot Study

This chapter contains the results of the investigation that answer the third research question in 1.2 Research Questions and show how execution tracing quality could be described by a standalone model able to capture subjective uncertainty. Moreover, it also presents results on how the standalone model for execution tracing quality could be linked to the present software product quality frameworks.

The content of the chapter is being published in the journal Acta Polytechnica Hungarica [94]. The background of the problem to be solved was presented in the first section of the chapter 0, while the review of the mathematical tools and methods applied for capturing subjective uncertainty in the model is summarised in the Appendix 6.1.

Execution tracing quality is crucial to the overall software product quality that the present quality frameworks neglect. In the scope of this pilot study the author introduces a process to create a pilot model for describing execution tracing as a quality property; moreover, the performance comparison of four different models created is also carried out. The process and the models presented are capable to capture subjective uncertainty, which is intrinsic to the quality measurement process.

The initial experiment of the pilot study, and its less thorough analysis, was submitted by the author as a practical assignment for the postgraduate module IMAT5119 Fuzzy Logic at the De Montfort University (UK). The pilot study used the results of this assignment.

The remainder of this chapter is structured as follows: the next section describes how the quality model pilot was built. It includes identification of inputs, outputs and construction of the knowledge base. The following section presents the validation of the quality model. This is followed by an analysis of the limitations of the pilot study; moreover, an outlook to the final model is provided. Related works are also presented. Finally, the contributions of the work are summarised and the future work on this area is outlined.

4.1 Constructing the Model

The model here presented reflects the results of an empirical research that comprises of two parts: (1) a qualitative part to determine the model's inputs, i.e. the quality properties on which execution tracing quality depends, and (2) a quantitative part to describe the relationships between the inputs and the output.

The qualitative research consists of a brainstorming session and further processing of the output of this session. Brainstorming served as the data collection method for the pilot study. Developed by A. Osborn and sophisticated by H. C. Clark as a technique to create, collect, and express ideas on a topic [28]. The main principle of the method relies in the following two fundamental factors: (1) each group member must have the possibility to express ideas without having to expose them to critic at first, and (2) ideas can be developed further by other group members. Consequently, synergistic effects can lead to the triggering of ideas by those already present [28]. Before and after the idea generation phase an ideation phase must take place. In the first ideation phase, the participants think over the brainstorming question individually as a preparation for the brainstorming [29]. The idea generation is followed by a second ideation phase where evaluation of the collected ideas takes place [28]. Critics of this method mainly focus on the idea generation phase regardless of the ideation phase that takes place before and after it; however, Osborn did not propose brainstorming as a substitute of the ideation process but as its supplement [29]. In this method the quantity of ideas is not limited. The more ideas that are collected the more probable it is to have quality ideas among them. The latter has been questioned in [30], contradicting in some respects the views held in [29]. The brainstorming was performed following the recommendations to effective brainstorming as described in section 1.4.2.1.1.

The output of the brainstorming is a list of raw ideas considered to be feasible by the group [28]. This list forms the possible input candidates of the model, which need to undergo further analysis.

The quantitative part of the research formalises the relationships of the inputs and the output. The collection of this information was achieved by using the knowledge of an expert software developer with several years of experience in dealing with software maintenance. The quantitative part of the research needs to use methods to deal with

subjective uncertainty. The subjective uncertainty of the expert is modelled in the pilot study by using fuzzy logic to describe the input-output relationships, which offers tolerance towards imprecision [95].

Fuzzy logic offers basically two kind of theoretical approaches to the problem: type-1 and type-2 fuzzy logic. Type-1 fuzzy logic can consider a certain amount of subjective uncertainty and it usually performs well in process control but shows less positive results in decision making where the same concept can be viewed differently by different people. In contrast, type-2 fuzzy logic performs well in both situations but its mathematics and inference mechanisms are more complex and computationally more expensive than those of type-1 fuzzy logic. In the pilot study described in this paper type-1 fuzzy logic is used [31], [96], [97], [98].

Fuzzy modelling makes it possible to directly incorporate human expertise in the model [99], [100]. Castillo and Melin recommend the following modelling steps [100]:

1. Determining the relevant input and output variables
2. Choosing the type of the fuzzy inference system
3. Determining the number of linguistic terms associated with each input and output variable
4. Designing the fuzzy if-then rules
5. Choosing memberships functions
6. Interviewing human experts to determine the parameters of membership functions
7. Refine the parameters of membership functions

As four fuzzy models have been built and tested in the scope of this pilot study, the above steps were not performed in the same order as they stand in the list. In addition, tuning the membership functions did not take place so that a fair performance comparison could be carried out of the different models by using the same membership functions.

4.1.1 Determining the Inputs and the Output of the Model

The output of the model, i.e. execution tracing quality, originates from the goals of the research; meanwhile, brainstorming identified the possible inputs, i.e. quality properties on which execution tracing quality depends.

The brainstorming group was constructed of software developers and software maintainers with several years of experience. The list of feasible ideas collected by the group underwent analysis by two experts who scored the input candidates according to their importance with regard to execution tracing quality. The experts had to distribute the same amount of scores among the items collected i.e. constant sum scaling was applied [31].

The arithmetic means of the scores assigned by experts were calculated. Each input candidate that has been selected as input has a relative importance above 10% according to the judgement of the experts. In this way the chosen inputs of the execution tracing quality model are:

1. Processability
   Processability refers to such properties of the execution traces whether (1) the trace possesses appropriate granularity for the examination of the execution path, (2) communication dialogs can uniquely be identified, (3) threads can uniquely be identified, (4) process IDs are traced, (5) error severity is traced, (6) component interfaces can be traced, (7) trace entries are marked with a timestamp with appropriate granularity.

2. Code Coverage
   The property code coverage indicates maximally how many per cent of the source code is covered with execution tracing.

3. Configurability
   Configurability encompasses how easily and sophisticatedly the execution tracing can be configured. This property includes such judgements whether (1) execution tracing can be set to different levels of granularity, (2) the configuration change in execution tracing requires complex actions from the operators or

developers, maintainers, (3) it is possible to configure a performance trace which only traces method invocations at the component boundaries to have less impact on the performance, (4) it is possible to trace in different outputs including file, database, network socket, (5) it is possible to trace in different formats including: plain text, xml, html, proprietary binary, ASN.1 BER, ASN.1 PER.

4. Consequent Naming
   Consequent naming refers to the property whether the same events are always traced with the same pattern in the output, including whether (1) exceptions are always designated with the same identifiers, (2) the same level of errors and warnings are consequently used, (3) method entry and exit points are consequently traced.

4.1.2 Linguistic Variables

Lotfi Zadeh introduced the notion of the linguistic variables in order to model imprecision and offer a basis for natural language computation [101]. The formalism implemented by these variables and the if-then rules establish an effective modelling language [95].

Before identifying the appropriate linguistic variables, each input and output needed to undergo partitioning to determine the granularity with which the system has to be described. A high number of partitions makes sophisticated description possible at an increased complexity because the number of necessary fuzzy rules increase. Moreover, incorporating human expertise with a high number of linguistic variables exposes difficulties because contradictions can be introduced in the model and inconsistent results could derive [102], [103], [104]. Finding a consensus between the possibility of a sophisticated model description and the reduction of the possibility of introducing contradictions in the model, three input partitions and five output partitions have been defined. The linguistic variables for the defined partitions have been identified in the following way:

Linguistic variables for all inputs: {*poor*, *medium*, *good*}
Linguistic variables for the output: {*very poor*, *poor*, *medium*, *good*, *very good*}

4.1.3 Membership Functions

Linguistic variables were modelled by means of membership functions to make inference possible. While developing the model for execution tracing quality, two types of membership functions were used: (1) triangular, and (2) Gaussian, both types with overlaps as illustrated in Figure 7.

Each membership function maps the interval *[0, 100]* to the interval *[0, 1]*. The domains of the membership functions can be interpreted as percentage values, while the codomain depicts the degree of membership in the given category.

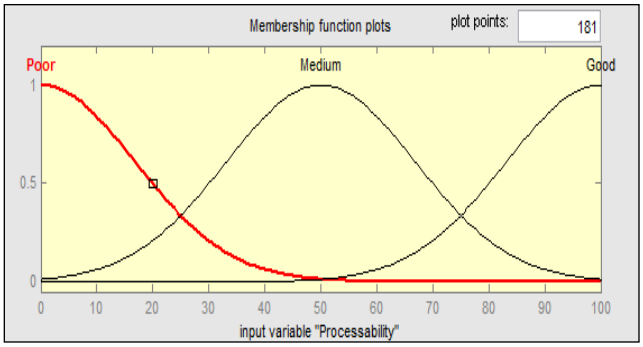

Figure 7 Membership Functions of the Input: Processability

4.1.4 Knowledge Base for the Model

The knowledge of one expert with regard to execution tracing quality has been described with the formalism offered by the if-then rules and the linguistic variables [95]. The knowledge base is summarised in Table 7. This is not a complete rule set, i.e. it does not contain each variation of all linguistic variables of all inputs. It is noted that a complete rule set is not necessary to achieve appropriate performance [105]. The model was assessed as described in section Validation.

	Antecedent Linguistic Variables are Connected by Logical AND Operation				Consequent
ID	Processability	Code Coverage	Configurability	Consequent Naming	Execution Trace Quality
1.	*poor*	*poor*	n.a.	n.a.	*very poor*

2.	*medium*	*poor*	n.a.	n.a.	*poor*
3.	*poor*	*medium*	n.a.	n.a.	*poor*
4.	*medium*	*medium*	*Poor*	*poor*	*poor*
5.	*medium*	*medium*	*Poor*	*medium*	*medium*
6.	*medium*	*medium*	*Medium*	*medium*	*medium*
7.	*medium*	*medium*	*Good*	*medium*	*medium*
8.	*medium*	*medium*	*Good*	*poor*	*medium*
9.	*medium*	*medium*	*good*	*good*	*good*
10.	*medium*	*medium*	*Poor*	*good*	*medium*
11.	*good*	*medium*	*Poor*	*poor*	*poor*
12.	*good*	*medium*	*Medium*	*poor*	*medium*
13.	*good*	*medium*	*Good*	*poor*	*medium*
14.	*good*	*medium*	*Poor*	*medium*	*medium*
15.	*good*	*medium*	*Medium*	*medium*	*medium*
16.	*good*	*medium*	*Good*	*medium*	*medium*
17.	*good*	*medium*	*Poor*	*good*	*good*
18.	*good*	*medium*	*Medium*	*good*	*good*
19.	*good*	*medium*	*Good*	*good*	*good*
20.	*good*	*good*	*Poor*	*poor*	*medium*
21.	*good*	*good*	*Medium*	*poor*	*medium*
22.	*good*	*good*	*Good*	*poor*	*good*
23.	*good*	*good*	*Poor*	*medium*	*medium*
24.	*good*	*good*	*Medium*	*medium*	*medium*
25.	*good*	*good*	*Good*	*medium*	*good*
26.	*good*	*good*	*Poor*	*good*	*medium*
27.	*good*	*good*	*Medium*	*good*	*good*
28.	*good*	*good*	*Good*	*good*	*very good*
29.	*medium*	*good*	*Good*	*good*	*medium*
30.	*poor*	n.a.	n.a.	*good*	*medium*
31.	n.a.	*poor*	n.a.	*medium*	*poor*

Table 7 Antecedent and Consequent Parts of the Fuzzy Rules

4.1.5 Type-1 Fuzzy Inference Techniques

The two most widespread fuzzy methods for inference have been considered [106], [99]: (1) Mamdani's approach, and (2) the approach of Takagi-Sugeno-Kang. Tsukamoto's method has been excluded as it requires monotonic consequent membership functions.

4.1.6 Comparison of the Created Models

For the purpose of comparison, four models were created with the same inputs and output variables: (1) type-1 fuzzy logic with Mamdani's approach with triangular membership functions, (2) type-1 fuzzy logic with Mamdani's approach with Gaussian membership functions, (3) type-1 fuzzy logic with the approach of Takagi-Sugeno-Kang with triangular membership function, (4) type-1 fuzzy logic with the approach of Takagi-Sugeno-Kang with Gaussian membership functions. In addition, Mamdani's approach was also tested with two different defuzzification techniques: (1) mean of maxima (MOM), and (2) centroid of gravity (COG). The validation charts are presented for the best performing method, which in this context was implemented by the inference mechanism of Takagi-Sugeno-Kang with Gaussian membership functions in the pilot study, while the charts of the other approaches are placed in the Appendix 6.1. The outcomes of the other approaches are briefly introduced below.

The acceptance criteria towards the model and its output can be summarized in the following way:

1. Representation of expert's knowledge
2. Appropriate response for the changes in inputs
3. No oscillation in the output for input changes
4. Full output range needs to be used
5. The smoothness of the output is desired as it satisfies the problem better then fitting 2D planes together which build sharp edges where they join causing drastic responses in the output for small changes at certain points of the input.

4.1.6.1 Mamdani's Approach

Inference was performed with the min-max method [106]. The model built with Gaussian and triangular membership functions did not show significant differences, nevertheless,

the surfaces achieved with Gaussian membership functions were slightly smoother. See the appendix for further details.

The defuzzification methods applied indicated considerable deviations when the inputs reached the limit values of the input range: the COG method did not use the full output range in contrast to the MOM method, which used the full output range. The MOM method can cause oscillation in the output [107].

The model built according to Mamdani's approach also shows sharp edges on the surfaces of the validation charts. With triangular membership functions thirty one rules were applied to describe the system and thirty rules were used with Gaussian membership functions for the same purpose.

4.1.6.2 Approach of Takagi-Sugeno-Kang

In the course of constructing the Takagi-Sugeno-Kang model, zero order functions (constants) were applied in the output range. This approach does not require defuzzification. For obtaining the output values weighted averages were calculated. Inference was performed with the product and probabilistic OR method.

The input Gaussian membership functions in comparison to the triangular ones resulted in more even transients between the different surface areas of the functions constructed from the input variables. The model with triangular membership functions contained thirty rules; meanwhile the model with Gaussian membership functions contained thirty one rules. Fine tuning of both models can be subject of further investigations. See the appendix for further details.

The model built with the approach of Takagi-Sugeno-Kang with Gaussian membership functions provided the best performance compared to the other models on basis of the above listed acceptance criteria. This inference technique helped to avoid sharp edges on the surfaces of the functions between the input and output variables.

Research also shows that the overlap of the antecedent membership functions determines the smoothness of the output behaviour with this inference method [99]. Further investigation of Jassbi, Serra, Ribeiro, and Donati confirms that the Takagi-

Sugeno-Kang method shows more tolerance towards input noise than Mamdani's method [108], which is an advantageous property in the problem domain of the current research.

4.2 Validation

As the best results were produced with the Takagi-Sugeno-Kang approach with Gaussian membership functions, the validation of this model is presented in this section. The other charts are placed in the Appendix 6.1. The model possesses four inputs; consequently, six different combinations of the input pairs are possible to depict the influence of the inputs on the output, i.e. on execution tracing quality. Face validity [109] was applied to validate the model. An expert checked, whether potential changes in the inputs cause appropriate response changes in the output according to the charts.

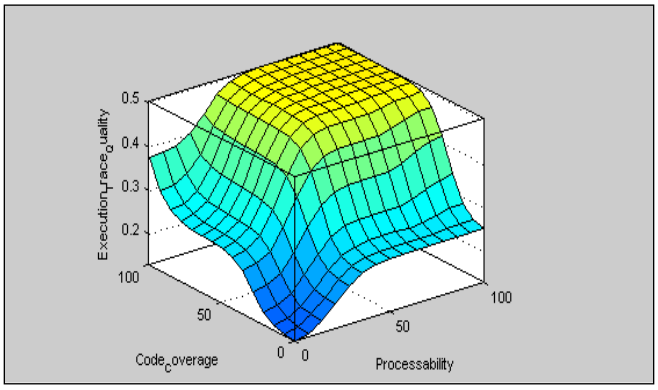

Figure 8 Code Coverage and Processability vs. Execution Trace Quality

Figure 8 shows that the decrease of the inputs "*Processability*" and "*Code Coverage*" below the medium level have a drastic impact on the execution tracing quality which reflects also the expert's opinion. On the other hand, maximum quality of "*Processability*" and "*Code Coverage*" cannot cause a more than 50% increase in execution tracing quality, which supports the idea that these two inputs in themselves cannot cause the output to reach its maximum value.

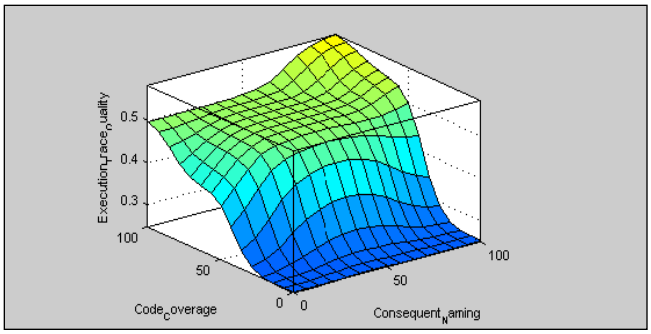

Figure 9 Code Coverage and Consequent Naming vs. Execution Trace Quality

Figure 9 illustrates that "*Code Coverage*" has a far stronger impact on the output than "*Consequent Naming"*. The system needs some fine tuning with regard to "*Consequent Naming*" in the *medium* range as the surface has a slight enhancement which slowly falls back when the value of "C*onsequent Naming"* increases. The maximum of "*Code Coverage*" and "C*onsequent Naming"* in themselves cannot cause the output to reach its maximum value. The diagram reflects the expert's opinion.

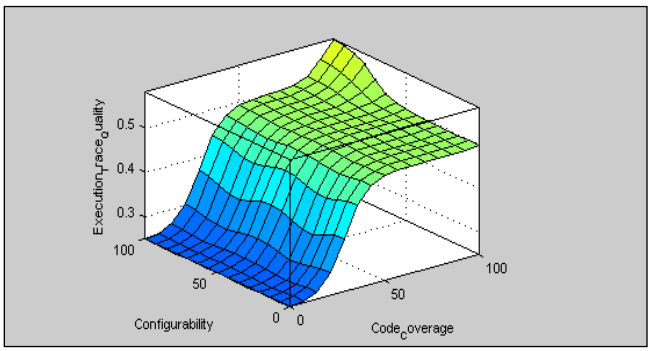

Figure 10 Configurability and Code Coverage vs. Execution Trace Quality

Figure 10 depicts that "*Configurability"* has a far smaller impact on the execution tracing quality than "*Code Coverage*". Significant decrease of the output can be observed if "*Code Coverage*" is below *medium*, which reflects the expert's opinion. The maximum quality of "*Configurability"* and "*Code Coverage"* without the other inputs cannot cause the output to reach its maximum value.

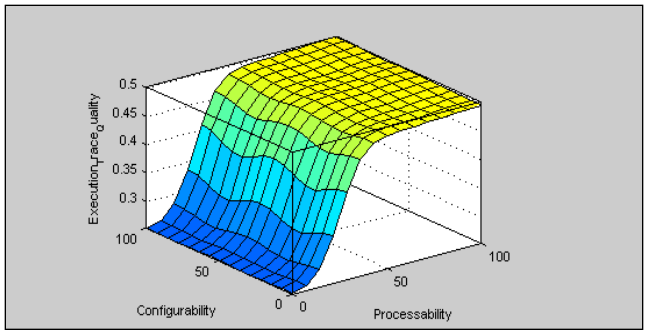

Figure 11 Configurability and Processability vs. Execution Trace Quality

Figure 11 shows that "*Processability"* contributes more to the execution trace quality than "C*onfigurability"*. With regard to the "*Processability*"-"Configurability" input pair the diagram shows that *"Configurability"* has nearly no influence on the output in comparison to *"Processability"*. The fuzzy rules need to undergo fine tuning to remove the slight waves from the chart, when *"Configurability"* changes. Moreover, it can also be observed that *"Configurability"* has little more than zero influence on the output in comparison to *"Processability"*, which has to be reflected by the model.

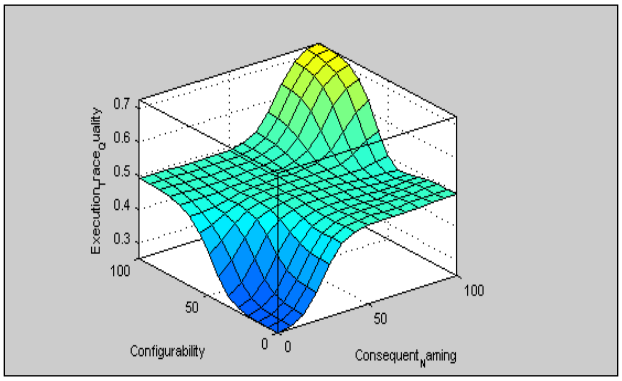

Figure 12 Configurability and Consequent Naming vs. Execution Trace Quality

Figure 12 shows that "*Configurability"* and "*Consequent Naming"* contribute to the output approximately to the same extent. Moreover, in comparison to the previously presented input pairs this combination has the highest influence on the output in the *good-good* range. However, even if both inputs carry the highest value, the execution tracing quality is limited, i.e. it depends on the other inputs too, as with the previously investigated pairs.

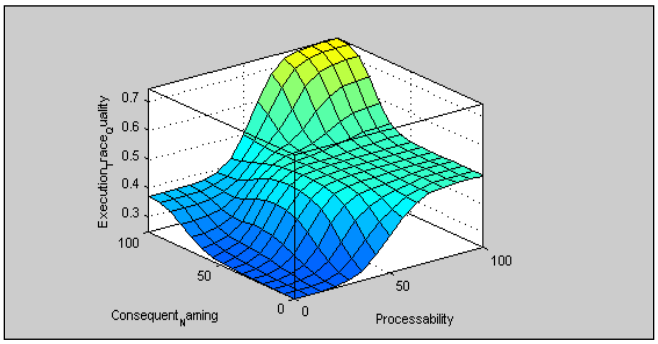

Figure 13 Consequent Naming and Processability vs. Execution Trace Quality

Figure 13 illustrates that both "*Consequent Naming"* and "*Processability"* have strong impacts on the output. The influence of the input pair reaches the same extent on the output as the "*Configurability"-"Consequent Naming*" input pair combination. The *medium-medium* ranges require fine tuning to avoid a slight local maximum on this area.

Table 8 summarises the main results from the validation study.

Summary of the validation charts		
ID	Diagram	Conclusion
1.	From Figure 8. to Figure 13.	Changes of the inputs produce appropriate responses in the output.
2.	Figure 8.	The inputs Code Coverage and Processability have a significant impact on Execution Trace Quality.
3.	Figure 9.	Code Coverage influences Execution Trace Quality to a bigger extent than Consequent Naming.
4.	Figure 10.	Code Coverage influences Execution Trace Quality to a bigger extent than Configurability.
5.	Figure 11.	Processability influences Execution Trace Quality to a bigger extent than

		Configurability.
6.	Figure 12.	The inputs Consequent Naming and Configurability have approximately the same impact to Execution Trace Quality.
7.	Figure 13.	Processability has a bigger impact on Execution Trace Quality than Consequent Naming.
8.	Figure 13.	The fuzzy rules or the parameters of the membership functions need to undergo fine tuning to avoid the local maximum in the *medium-medium* range of the input variables Consequent Naming and Processability.

Table 8 Summary of the Validation Charts

4.3 Related Works

Canfora, Aggarwal, Nerurkar amongst others have already illustrated how fuzzy mathematics can help to make judgements or predictions in connection with software maintainability [33], [34], [35] or reusability [36], [37]. However, these models cannot help with the assessment of software product quality as a whole because they are not linked to extensive software product quality frameworks like ISO/IEC 25010 [41]. In addition, the maintainability models investigated do not handle execution tracing quality.

Canfora, Cerulo, Troiano in [34] applied fuzzy logic to consider the following particularities in maintainability:

1. The assessment of software maintainability is influenced by qualitative and quantitative data including their subjective uncertainty.
2. Qualitative data that are often gathered by surveys are not always available.

3. The different sub-characteristics of maintainability contribute to the overall maintainability to different extents.

Aggarwal, Singh, Chandra, Manimala discussed in [33] how an integrated metric of maintainability correlated with the time devoted to error corrections, however individually none of the investigated inputs of their model correlated with the time spent on error corrections. The model was constructed by means of type-1 fuzzy logic.

Nerurkar, Kumar, Shrivastava in [36] proposed a model based on type-1 fuzzy logic for reusability of aspect-oriented systems. Singh, Bhatia, Sangwan in [37] examined different soft computing techniques for software reusability assessment. In their publication type-1 fuzzy logic, neural network, and adaptive neuro-fuzzy inference were compared for evaluating software reusability.

4.4 Summary

The pilot results illustrate that fuzzy modelling can be deployed to create a model for execution tracing quality to encompass the subjective uncertainty associated with the measurement process of software product quality.

In addition, modelling the knowledge of experts manually even if this knowledge is formalised with only thirty rules, introduces the chance for contradictions in the rule base. The number of these contradictions can considerably be reduced if the knowledge of several experts is considered in order to find a consensus and if automatic rule generation is used with adaptive neuro-fuzzy inferencing.

The experimental models furthermore showed that the Gaussian membership functions performed better under the same settings because they contributed to avoiding sharp transients on the three-dimensional validation charts. Moreover, the most preferential smoothness in the output was achieved with the inference of Takagi-Sugeno-Kang when using overlapping Gaussian membership functions. In addition, Mamdani's inference method with the COG or MOM defuzzification techniques could not be applied because it does not satisfy the acceptance criteria introduced.

The pilot has been validated by face validity. For the pilot study the purpose was to test the research methodology and analysis to show the feasibility of the approach to model execution tracing quality. For this purpose face validity was sufficient to show that the selected approach is workable and can yield usable results. For the final model of execution tracing quality a more rigorous validation will be required.

This section answers the third research question of the present investigation and shows that execution tracing quality can be described by a model which considers subjective uncertainty.

5 Conclusion

In this chapter the findings of the research are presented in a brief manner pointing also at the outcome of the literature review and the pilot study.

5.1 Software Product Quality Frameworks and Execution Tracing

Present software product quality frameworks do not exhibit any property to describe execution tracing although they usually offer the potential to be extended to achieve this. In this report we analyse such extension points and articulate concrete possibilities for extension in the context of the current investigation. Software product quality frameworks form complete models to support the description and assessment of the quality of software products. As research shows [49], conformance with process quality models does not guarantee good-quality software products, motivating the application of software product quality frameworks in synergy with process quality models.

In summary, contemporary quality frameworks are not able

1. to exhibit the variables on which execution tracing quality depends,
2. to model execution tracing quality,
3. to capture the subjective nature uncertainty associated with execution tracing,
4. to support the improvement of execution tracing with respect to their specific quality requirements.

The increasing size and complexity of software systems considering their varying workload makes localizing software errors more difficult. This difficulty is more challenging with regard to the enormous number of software and hardware combinations. Adding execution trace to some key places of the application can drastically reduce the time spent with debugging. Consequently, execution tracing has direct impact on the development and maintenance costs [14].

In addition, debugging is not necessarily a feasible option when (1) applications perform process control, (2) the error is related to parallel processing and race conditions, or (3)

performance problems need to be analysed [14], [15]. In the case of distributed, multithreaded applications execution tracing is the only adequate instrument to help with the error analysis as Laddad states in [17]. In the case of embedded applications, which have no user interface, only by means of execution tracing can the developer or system maintainer answer such questions as what the application is doing [15].

Moreover, execution tracing significantly influences program comprehension, the importance of which arises if the program documentation is deficient or of poor quality. Nelson and Shi cite the study of Fjeldstad and Hamlen in which comprehension of existing software systems is estimated to consume between 47% and 62% of all maintenance resources [110], [20]. An experiment conducted by Karahasanovic and Thomas introduced in [19] categorised the difficulties related to the maintainability of object-oriented applications. Program logic was ranked the first in the source of difficulties. Understanding the program logic belongs to the category of software specific knowledge and it can greatly be enhanced by execution tracing, offering a basis for trace visualisation and program comprehension [20].

Tracing, logging, or constraint checks represent a significant part of the source code of applications. Spinczyk, Lehmann, and Urban in [21] state that the ratio of code lines related to monitoring activities reached approximately 25% in their measurements targeted at certain commercial applications. This ratio shows that a significant amount of source code is written to deal with such tasks, which evidence that execution tracing in itself is an important quality factor.

In conclusion, the above indicate that execution tracing has significant impact on the analysability of software systems. As the author has already pointed out in [111] present software product quality frameworks need to be extended to describe and implement execution tracing quality. Moreover, measuring quality is difficult, some properties are easier to measure than others even if they are well defined [24]. Quality frameworks include the description of qualitative properties in a quantitative manner and *quality measure element*s that cannot be measured directly but only derived. Consequently, the measurement process implicates subjective uncertainty, a fact that has been admitted by the standard ISO/IEC 25021:2007 regarding software product quality by defining the subjective measurement method. In the scope of this investigation the author introduces

a pilot study to describe execution tracing quality by means of a model able to encompass subjective uncertainty. The model itself does not perform quality assessment but it can be used to define quality targets against which a product can be assessed.

5.2 Fuzzy Logic

The literature review conducted suggests that both type-1 and type-2 fuzzy logic are feasible, either with Mamdani's or Takaki-Sugeno-Kang's inference methods, to be used in this area. Moreover, the literature review showed that adaptive techniques for generating the rule base automatically or semi-automatically for a fuzzy system are more mature in the type-1 fuzzy domain than in the type-2 domain.

In the context of the research, the pilot study investigating the different fuzzy inference methods and defuzzification techniques showed that Takagi-Sugeno-Kang approach provided the best performance. In addition, the pilot model also proved that it is easy to introduce contradictions in the rule base, in spite of possessing relatively few rules formalizing one expert's knowledge only. Synthesising the rule base in an adaptive manner would help to avoid contradictions among the rules. These findings result in the selection of type-1 fuzzy logic with the ANFIS approach for performing the model construction for execution tracing to incorporate several experts' opinion.

5.3 Limitations of the Research and Potential Future Work

We need to make a distinction between the research methods applied for the pilot model introduced here and the potential final model. Both approaches are empirical in nature and comprise of qualitative and quantitative research methods. The qualitative research part determines the inputs of the quantitative research i.e. the quality properties on which execution tracing quality depends in both cases. In addition, the quantitative research determines the impacts of these properties in execution tracing quality.

5.3.1 Reliability and Validity

The reliability of the model strongly depends on the reliability of the data collected and its consistent processing. The data of the pilot originate from the output of one brainstorming session processed by two experts in the field; while, the knowledge base formalises the knowledge of one expert. In contrast, the data of the final model need to be based on a well-defined study population. Participants of the brainstorming sessions need to be selected from this study population for the qualitative research. Several brainstorming sessions have to take place until a saturation point is reached [32]. To implement this, at least two coders need to look for synonyms in the outputs of the brainstorming sessions and intercoder reliability has to be kept on a high level during the synonyms identification [109]. Moreover; the data collected need to undergo first and second cycle coding to establish the quality properties as reported in [32]. Coding also assumes calculating intercoder reliability for the coding process between the coders with high values.

Regarding the quantitative stage, the defined study population needs to be sampled with an appropriate sampling method to ensure a $p<.05$ statistical significance [112], [113]. The knowledge base, i.e. the rule set, of the model could therefore be constructed from the knowledge gained from the sample.

This validity would be based on statistical evidence not just on face and content validity [109]. Furthermore, the final model is to be constructed by using the adaptive neuro-fuzzy approach (ANFIS) to keep internal consistency by creating the model on half of the randomly selected data and using the other half for control purposes [99]. Application of ANFIS is also considered being necessary due to automatic processing the lager amount of data planned to be collected during the quantitative research.

5.3.2 Extending Present Frameworks

Software product quality frameworks of the ISO/IEC standards allow extensions and have a defined method to do so. Moreover, they also offer a natural linking point for execution tracing quality with the analysability sub-characteristic of maintainability.

Dromey's model describes only code-level constructs and their quality considering the procedural programming paradigm. The principle of the model, however, can also be applied for higher-level constructs and additional programming paradigms. From the point of view of execution tracing procedural programming does not cause difficulties but the usability of the model would significantly be reduced if no other programming paradigms could be represented. To encompass additional programming paradigms and higher-level artefacts, Dromey's model requires considerable amounts of new quality-carrying properties and new structural forms. The high number of elements in both sets enhances the number of combinations through which relationships need to be expressed. Consequently, the execution tracing quality can be described at the cost of introducing more complexity in the model. However, the direct assignment of binary quality-carrying properties to high-level attributes does not facilitate the modelling and implementation of calculations necessary to describe uncertainty. Therefore extending Dromey's model is not appropriate for accommodating the execution tracing property.

The model introduced in our pilot study is a standalone model that offers the possibility to be linked to the analysability sub-characteristic of the characteristic maintainability of ISO/IEC 9126-1 [40] or ISO/IEC 25010 software product quality models [41]. Linking the developed model to the standards is possible after formal description of the inputs, required by ISO/IEC 25021 [25], and after applying decomposition according to the internal-external view of the software product quality expressed by the ISO/IEC 9126-1 standard if it is the target. ISO/IEC 25010 does not require this decomposition.

6 Appendix

The appendix comprises of the description how subjective uncertainty can be considered by software product quality models in the section 6.1, moreover, it presents the Matlab charts and code of the pilot study which were not placed in the body of the thesis in the sections: 6.2 and 6.3. Finally, the appendix closes with the references in 6.4.

6.1 What Mathematical Tool Can Help to Incorporate Subjective Uncertainty in Quality Models

This section presents how subjective uncertainty referring to the present could adequately be captured by techniques adaptable in the quality models. In software product quality measurement uncertainty is associated with categorizing information based on human behaviour or making judgements on achieving criteria in certain categories. This section focuses on dealing with this kind of uncertainty.

The most accepted view [27] of uncertainty establishes three broad categories: (1) objective uncertainty of events which refer to the future, (2) subjective uncertainty of events which refer to the future, and (3) subjective uncertainty which is not related to the future [26]. Objective uncertainty is addressed by means of probability theory, the use of which is widespread; mathematical statistics is also based on that. The theory of subjective probability is concerned with the subjective uncertainty which refers to the future. This manifestation of uncertainty is considered an application area of Bayesian statistics. In contrast, the subjective uncertainty which does not refer to the future is not dealt within the scope of subjective probability but within the area of fuzzy set theory and fuzzy logic. Logical statements which have at least one variable which is a fuzzy set form fuzzy logic [26].

The current research considering the outlook to the future work deals with both objective and subjective nature of uncertainty as the selection of the sample is based on statistical procedures which developed from probability theory and the model construction uses fuzzy logic.

More kinds of mathematical techniques could be involved as alternatives of fuzzy logic if the research could be traced back to a multi-criteria decision making problem. Analytic Hierarchy Process [45] produces a ranking of alternatives based on subjective inputs. Saaty in [114] sees the difference between exact sciences and decision making in the point that the first applies objective measures and scales and later interprets the results which implicate subjectivity; in contrast, decision making performs subjective judgments and following these judgments it carries out objective evaluation based on this subjective information. Moreover, Outranking Relations [115] would also qualify as further alternative of fuzzy logic in the decision making context as introduced in [116] to assign software products to predefined quality profiles.

6.1.1 Fuzzy Logic

Set theory helps to make distinctions between groups of things which share the same characteristics. Classical propositional logic is designed to describe propositions and reasoning to distinguish the truth from falsity; moreover, predicate logic allows to make distinction between singular propositions and general proposition. From this point of view formal logic and set theory serve the same purpose [26].

Fuzzy logic can be regarded as a system of principles to deal with approximate reasoning. Fuzzy logic is an application domain of fuzzy set theory. However, for being able to use fuzzy set theory for reasoning, the connection between degrees of membership and the degrees of truth of a fuzzy proposition needs to be established. Thus, the membership degree needs be interpreted as the degree of truth with regard to linguistic expression [26].

The concept of fuzzy sets was established by L. Zadeh in [117] proceeding his works at the beginning of the 1960s. The concept developed and the notion of fuzzy rules was also introduced to capture human knowledge. The technique gained acceptance in the 1970s. Mamdani introduced the first inference method in 1975 which was followed by the inference methods of Takagi, Sugeno, and Kang in 1980 forming the two most familiar inference techniques [106], [99], [26].

The evolution of the perception of uncertainty resulted in notion of type-2 fuzzy logic [100]. Fuzzy logic became a widespread technology in the last three decades with many branches from process control to decision making and large number of publications appears each year focusing on its research. The present literature review concentrates on the introduction of fuzzy logic and its particularities which are related to the scope of the current research.

6.1.1.1 Fuzzy Sets

Classical set theory applies sets with crisp boundaries. Consequently, an element in classical set theory is either member of the set or not member of the set. Classic sets therefore do not reflect the human thinking which tends to be abstract and imprecise [99]. Fuzzy sets are sets without crisp boundaries implicating that belonging to a set is gradual [99]. The transition between being a member and being a non-member of a certain set is characterized by a function called membership function [99]. The characteristic function of set theory has the same goal as the membership function of fuzzy set theory.

A fuzzy set can be defined either (1) by listing its elements and the membership grade of each element if the set is discrete and finite, or (2) by specifying a membership function if the set is continuous.

Let X be a finite universe $X = \{x_1, x_2, x_3, x_4, \dots\ x_n\}$, then a fuzzy set A can be represented as follows [118]:

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \frac{\mu_A(x_3)}{x_3} + \frac{\mu_A(x_4)}{x_4} + \cdots + \frac{\mu_A(x_n)}{x_n}$$

Let X be a continuous universe, where $x \in X$, then a fuzzy set A can be represented as follows [118]:

$$A = \int \frac{\mu_A(x_i)}{x_i}$$

In the above description the integral sign, the division and addition signs do not mean arithmetic integral, division and addition but they are merely used to describe fuzzy sets. Thus, construction of a fuzzy set depends (1) on the identification of a suitable universe of discourse and (2) on the specification of an appropriate membership function if the set is continuous [99].

Hamrawi summarizes this as follows [119]:

> "Let, $x \in X$, be an element of the universe X that belongs to the fuzzy set, $A \in X$. The grade of membership, μ_x, associated with each element, x, is a value in the unit interval, $U = [0, 1]$, i.e. $\mu_x \in U$. In general, a fuzzy set can be represented as the union of sets containing ordered pairs associating discrete points of the domain with their corresponding membership grades $A = \{(x, \mu_x) | \forall x \in X, \mu_X \in U\}$."

6.1.1.2 Principle of Incompatibility

Principle of incompatibility was published by Zadeh [101]. High precision is incompatible with complexity. The complexity of a system and the precisity with which it can be examined have an inverse relation to each other. Conventional techniques applying precise manipulation of numerical data are intrinsically not capable to express the complexity of human thinking and decision making. Consequently, science needs to become more tolerant towards approaches which are approximate in nature to be effective for systems which are too complex.

6.1.1.3 Relating Crisp Sets to Fuzzy Sets

The listed unary operations performed on fuzzy sets result in crisp sets.

6.1.1.3.1 Alpha-cut

The alpha-cut of a fuzzy set is a crisp set which can be defined as follows [119]:

$$A_\alpha = \{x \in X \mid A(x) > \alpha, \alpha \in [0,1]\}$$

6.1.1.3.2 Core

The core of a fuzzy set A is a crisp set which contains all elements of A whose membership grade equals to 1 [100].

$$core(A) = \{x \in X \mid A(x) = 1\}$$

6.1.1.3.3 Support

The support of a fuzzy set A is a crisp set which contains all elements of A where the membership grade is not zero [100].

$$supp(A) = \{x \in X \mid A(x) > 0\}$$

6.1.1.4 Properties of Fuzzy Sets

The most frequently used properties of fuzzy sets can be summarized as illustrated below.

6.1.1.4.1 Cardinality

Cardinality of a finite fuzzy set A on the universe X is the number of elements of the set [118]:

$$|A| = \sum \mu_A(x)$$

6.1.1.4.2 Containment

A fuzzy set A is contained in fuzzy set B i.e. A is a subset of B if the following equation holds [100]:

$$\mu_A(x) \leq \mu_B(x)$$

6.1.1.4.3 Crossover points

Crossover point of a fuzzy set A on the universe X is an x for which $\mu_A(x) = 0.5$ is true.

$$crossover(A) = \{x \in X \mid \mu_A(x) = 0.5\}$$

6.1.1.4.4 Supremum

Supremum is the height of a fuzzy set symbolized by the largest membership grade reached by any element in the set [119].

$$h(A) = \sup(A)$$

6.1.1.4.5 Convexity

A fuzzy set A on the universe X is convex if the following equation holds for $\forall x_1, \forall x_2 \in X, \quad \forall \varepsilon \in [0,1]$ [118]:

$$\mu_A(\varepsilon x_1 + (1-\varepsilon)x_2) \geq \min\left(\mu_A(x_1), \mu_A(x_2)\right)$$

6.1.1.4.6 Normality

A fuzzy set A is normal if its core is not empty, thus there exists an $x \in X$ where $\mu_A(x) = 1$ [100].

6.1.1.5 Fundamental Operations of Fuzzy Sets

In this section the fundamental operations of fuzzy sets are briefly summarized. The fuzzy sets are defined on the universe X; moreover, $x \in X$.

6.1.1.5.1 Union

The union of a fuzzy set A and B is a fuzzy set C where [100]:

$$\mu_C(x) = \max\big(\mu_A(x), \mu_B(x)\big).$$

The same operation can also be depicted in the following manner [100]:

$$C = A \cup B \Leftrightarrow A\ OR\ B$$

6.1.1.5.2 Intersection

The intersection of a fuzzy set A and B is a fuzzy set C where [100]:

$$\mu_C(x) = \min\left(\mu_A(x), \mu_B(x)\right).$$

The same operation can also be depicted in the following manner [100]:

$$C = A \cap B \Leftrightarrow A\ AND\ B$$

6.1.1.5.3 Complement

The complement or negation with other word, of a fuzzy set A can be designated with the ¬ sign [100]:

$$\neg\mu_A(x) = 1 - \mu_A(x)$$

The following designations are also possible [100]:

$$\neg A \Leftrightarrow \mathrm{NOT}\ A$$

6.1.2 Extension Principle

Extension principle provides a general procedure to extend the crisp domains of mathematical expressions to fuzzy domain [99]. For being able to interpret crisp

functions on fuzzy sets, crisp functions need to undergo fuzzification. This is required if fuzzy sets are involved in calculations [26].

Let f be a function which maps from the universe X to Y. Let A be a discrete fuzzy set on X defined as

$$A = \left\{\frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \frac{\mu_A(x_3)}{x_3} + \ldots + \frac{\mu_A(x_n)}{x_n}\right\}$$

The extension principle [99] claims that the mapping of f from A can be expressed as a fuzzy set B, where

$$B = f(A) = \left\{\frac{\mu_A(x_1)}{y_1} + \frac{\mu_A(x_2)}{y_2} + \frac{\mu_A(x_3)}{y_3} + \ldots + \frac{\mu_A(x_n)}{y_n}\right\}$$

and $y_i = f(x_i) \mid i = 1 \ldots n.\ f(A)$, which is a subset of Y, is also called the image of A by f [118].

If the function f implements a one-to-one mapping, then simply can be written [118], [99]:

$$\mu_B(y) = \mu_A(x)$$

If the function f implements a many-to-one mapping, then more, different x values have the same y values. In this case the membership grade in B is the maximum of the membership grades in A which possess the same y value [99]. More formally:

Let $x_1, x_2 \in X \text{ and } x_1 \neq x_2 \text{ and } f(x_1) = f(x_2) = y$, then $\mu_B(y) = sup_{x=f^{-1}(y)}\, \mu_A(x)$ [99].

In other words, if there is a function f which maps from the universe X to Y, then the extension principle provides rules by which the function can be extended to map either fuzzy sets on X to fuzzy sets on Y or to form an inverse function to map fuzzy sets on Y to fuzzy sets on X [26].

6.1.3 Notion of Linguistic Variables and Rules

As a consequence of the principle of incompatibility, which implicates that conventional techniques are unsuited to describe the complexity of human thinking including judgements, emotions and perception, Zadeh proposed the concept of linguistic variables [101], [100].

Linguistic variables are variables with approximate boundaries the exact meaning of which are described by membership functions. Linguistic variables establish an efficient tool for quantitative modelling of words in an artificial or natural language [99].

Linguistic rules are statements which include linguistic variables; moreover, an antecedent and a consequent part. They are also called if-then rules. Linguistic variables and linguistic rules constitute an effective modelling language [120].

Example of a linguistic rule:

If the cherry is red and sweet, then it is ripe.

Where red, sweet and ripe are linguistic variables.

The extent how sophisticated a linguistic rule can be stated depends also on the granularity of the linguistic variables. Returning to the above example, the more variables we define in colour for the cherry between white and red, moreover, between non-sweet and sweet, the more fine-graded statements are possible. Thus, the description depends also on this granularity. The more fine-granular the linguistic variables are, the more rules are necessary to describe the whole system. Determining the granularity of the linguistic variables is related to the input space called input space partitioning.

Basically three kind of input space partitioning is widespread: (1) grid partitioning which describes the whole input space with linguistic terms with clear meaning, (2) tree partitioning implemented by decision trees but the linguistic terms do not have clear meaning, (3) scatter partitioning which maps only a part of the input space. Grid partitioning can produce too high number of rules, if the partitions of the grid are small [99], [100].

However, not all of the possible rules need to be considered by the description of the system [103], [104]. Parsimonious systems can be constructed which possess significantly less linguistic rules than the possible combinations would ensure but the performance shows only minor deviations from the full description [105]. Subtractive clustering [121] and single value decomposition [122] can efficiently be used in the type-1 fuzzy domain. Chopra, Mitra and Kumar reduced the number of rules of a fuzzy

controller from 48 to 8 with keeping an acceptable performance [121]. The reduction of linguistic rules in the type-2 fuzzy domain is also a desired achievement and the latest researches show attractive results [105], [123].

6.1.4 Fuzzy Reasoning

The process of forming conclusions is called reasoning. Correct reasoning means that true conclusions are drawn from true premises. Inference patterns can be grouped in inductive and deductive patterns. Deductive reasoning possesses absolutely certain relationship between the premises and the conclusion while this relationship is not absolute certain for inductive reasoning [26].

> "If an inference is a correct deductive inference, then it is impossible for its premises to be true and its conclusion to be false. Thus, the relationship between premises and conclusion is one of certainty." [26]

The below table summarizes basic inference forms of deductive reasoning. The symbol ∴ means "therefore" while the ⇒ sign means "if … then…".

If p, then q
p

_____

Therefore q

The same statement, premise and conclusion can also be written as

$p \Rightarrow q$

p

_____

∴ q

Conjunction (Conj.) 1. p 2. q ∴ p ∩ q	Simplification (Simp.) 1. p ∩ q ∴ p
Addition (Add.) 1. p ∴ p ∪ q	Disjunctive Syllogism (DS) 1. p ∪ q 2. ⌐ p ∴ q
Modus Ponens (MP) 1. p ⇒ q 2. p ∴ q	Modus Tollens (MT) 1. p ⇒ q 2. ⌐p ∴⌐q
Constructive Dilemma (CD) 1. $(p \Rightarrow q) \cap (r \Rightarrow s)$ 2. p ∪ r ∴ q ∪ s	Destructive Dilemma (DD) 1. $\left(p \Rightarrow q\right) \cap \left(r \Rightarrow s\right)$ 2. ⌐q ∪ ⌐s ∴ ⌐p ∪ ⌐q
Hypothetical Syllogism (HS) 1. p ⇒ q 2. q ⇒ r ∴ p ⇒ r	Absorption (Abs.) 1. p ⇒ q ∴ p ⇒ (p ∩ q)

Table 9 Basic Inference Forms for Deductive Reasoning, source: [99, page 35]

Fuzzy logic utilizes deductive reasoning with the inference form of Modus Ponens[10] or Modus Tollens[11].

The variables of the above example in classical logic could only have two values either true or false; however, human reasoning uses Modus Ponens in an approximate manner [99] i.e. the variables are not bi-valued: “more or less”, “to some extent” and synonyms are expressed during the reasoning process.

The Modus Ponens for approximate reasoning:

$p \Rightarrow q$

p'

$\therefore q'$

This means that the input p’ matches the antecedent p to some extent, therefore the induced consequent q’ matches q only to some extent. The intersection of the membership functions between p and p’ determines the degree of compatibility or with other words the firing strength of the rule. The firing strength determines the induced consequent from the consequent membership function where the impact of the antecedent is reflected. The induced consequent is also named qualified consequent. The overall output encompasses the aggregated induced membership functions [99], [100].

6.1.5 Fuzzy Logic

Fuzzy logic is a logical statement where at least one variable is a fuzzy set. Such logical statements ensure (1) tolerance for imprecision, (2) offer the possibility to use words

[10] Modus ponendo ponens, Latin: The way that affirms by affirming.
[11] Modus tollendo tollens, Latin: The way that denies by denying.

instead of numbers when available information is too imprecise i.e. modelling the human mind is feasible, (3) tractability and better approach of the reality with low solution cost. These results cannot be done equally well with other methodologies including predicate logic, probability theory, neural network theory, Bayesian networks [95].

The reason for existence of fuzzy logic is surrounded with a long history with passionate discussions. There are several misconceptions about fuzziness [120].

> "Fuzzy logic is not fuzzy. Basically, fuzzy logic is a precise logic of imprecision and approximate reasoning. […] In fact, one of the principal contributions of fuzzy logic - a contribution which is widely unrecognized - is its high power of precisiation." [120]

Fuzzy logic attempts to formalize two human capabilities: (1) the capability of reasoning and making rational decisions in an environment of imperfect information including imprecision, uncertainty, incompleteness of information, partiality of truth and partiality of possibility; moreover, (2) the capability of performing tasks without measurements and computations [120].

Major contributions of fuzzy logic are articulated in [120] as follows:

- Generalisation: any bivalent-logic-based theory can be fuzzy logic generalized;
- Linguistic variables and if-then rules;
- Capability of precisiation in the sense of formulation of definitions of scientific concepts and formalization of human-centric fields such as economics, linguistics, law, conflict resolution, psychology and medicine;
- Natural-language computation in the sense of computing with words which is relevant also for the computation with imprecise probabilities. It is needed to handle second order uncertainty i.e. uncertainty about uncertainty;

6.1.5.1 Fuzzy Inference Systems

Fuzzy systems possess three conceptual components: (1) rule base: selection of fuzzy rules, (2) data base: membership functions used in the fuzzy rules, (3) reasoning

mechanism: the inference procedure. These systems are also referenced by other names as (1) fuzzy expert systems, (2) fuzzy model, (3) fuzzy associative memory and (4) fuzzy systems [100].

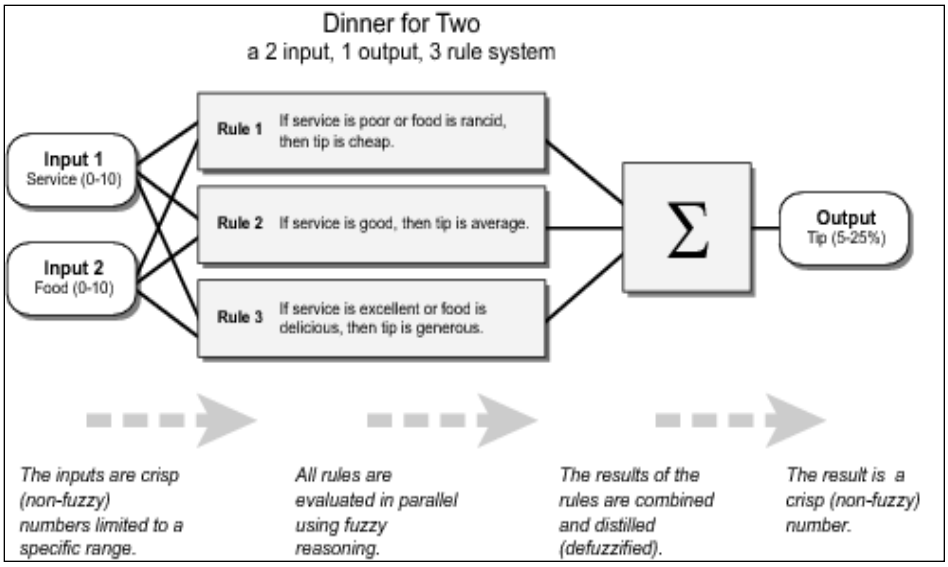

Figure 14 Fuzzy Inference System, source: [124]

Fuzzy inference systems (FIS) are based on (1) fuzzy set theory, (2) fuzzy if-then rules, and (3) fuzzy reasoning. A FIS can implement non-linear mapping from the input space to the output space with crisp input and output [99].

Several inference methods exist. For the literature review the three most widespread methods [99], [106] are presented which show major differences in handling the consequent part of the linguistic rule:

- Method of Mamdani
- Method of Takagi-Sugeo-Kang
- Method of Tsukamoto

Inference Method of Mamdani

Crisp inputs need to be fuzzified as Mamdani's method applies antecedents and consequents which are fuzzy sets. Usually min-max compositional operators participate

in the inference process but other compositional operators are also possible. The output set needs to be defuzzified to obtain a crisp value.

The inference process includes the following steps [99]:

1. Fuzzification of the inputs,
2. Calculating firing strength of the antecedents,
3. Implication,
4. Aggregating the result of each linguistic rule,
5. Defuzzification of the output fuzzy set.

Advantages [99], [125]:

- It follows the fuzziness of the underlying problem from the input to the output, illustrating the uncertainty.
- It is well suited to incorporate human input.
- It has wide acceptance.

Disadvantages:

- Defuzzification is computationally expensive [99].
- Some defuzzification methods can produce oscillation in the output [107].

Mamdani's method follows the fuzziness of the underlying problem from the input to the output, illustrating the uncertainty which can be considered an advantage of the inference method. As disadvantage the computational costs of the defuzzification needs to be mentioned.

The formal description of Mamdani's approach can be summarised by means of linguistic rules as follows:
IF input1=x and input2=y THEN output=z
where z is the defuzzified value of the system.

The inference process is illustrated below by the example from MathWorks Inc. to show how the amount of tip can be determined in a restaurant [124].

6.1.5.1.1 Fuzzification

The inputs supplied to the FIS are crisp values. These values will be converted to fuzzy sets by means of the membership functions. The below example possesses two inputs: food and service.

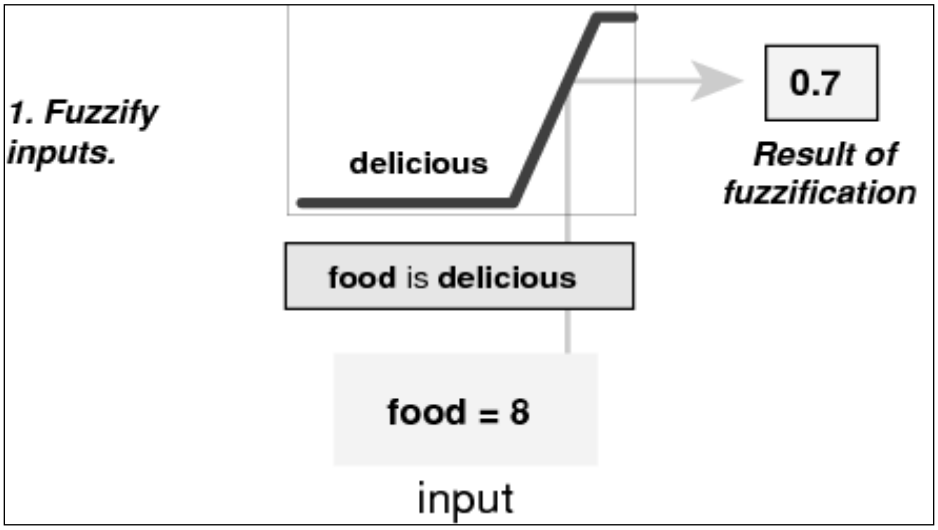

Figure 15 Fuzzification of the Input, Source: [124]

6.1.5.1.2 Calculating firing strength of the antecedents

The antecedent part of each linguistic rule can have more fuzzy variables. The composed impact of these variables needs to be considered to determine the firing strength of the rules.

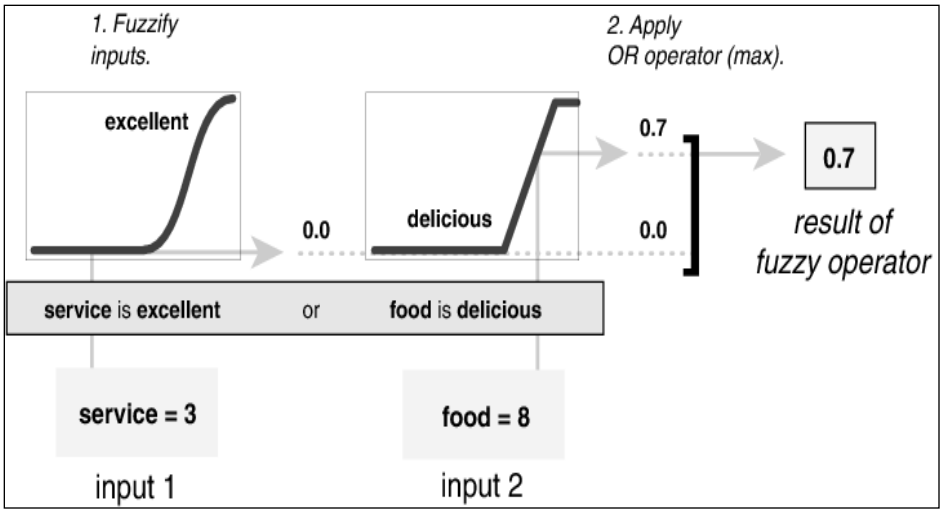

Figure 16 Composition of the Antecedents, Source: [124]

6.1.5.1.3 Implication

The implication shows the impact of the antecedent on the consequent of a linguistic rule as illustrated on the figure below.

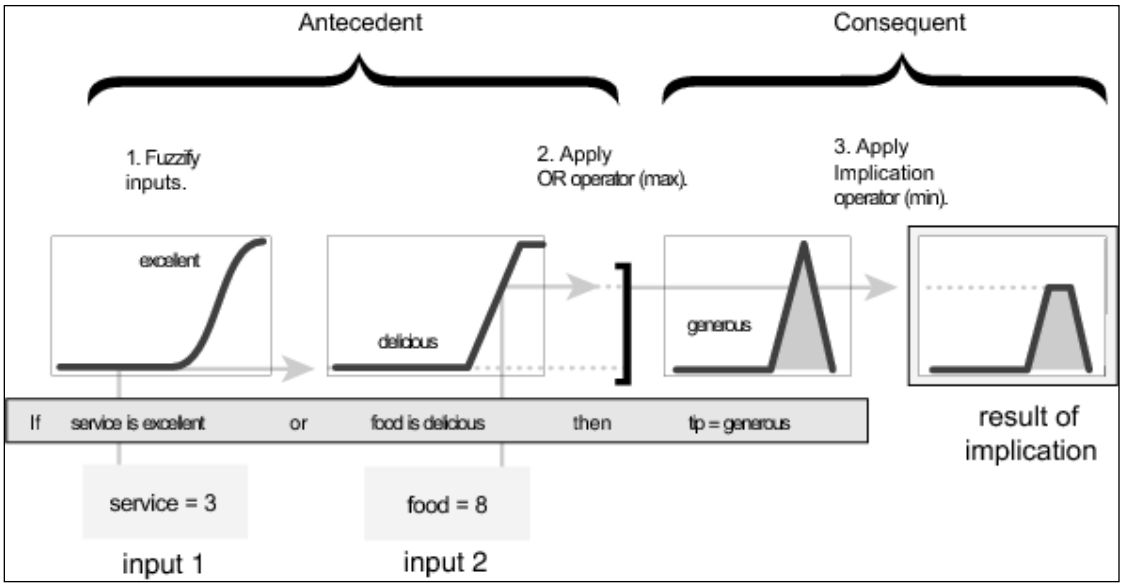

Figure 17: Implication Mechanism, Source: [124]

6.1.5.1.4 Aggregation

The result of each linguistic rule must be collected to be considered in overall output. This process is called aggregation.

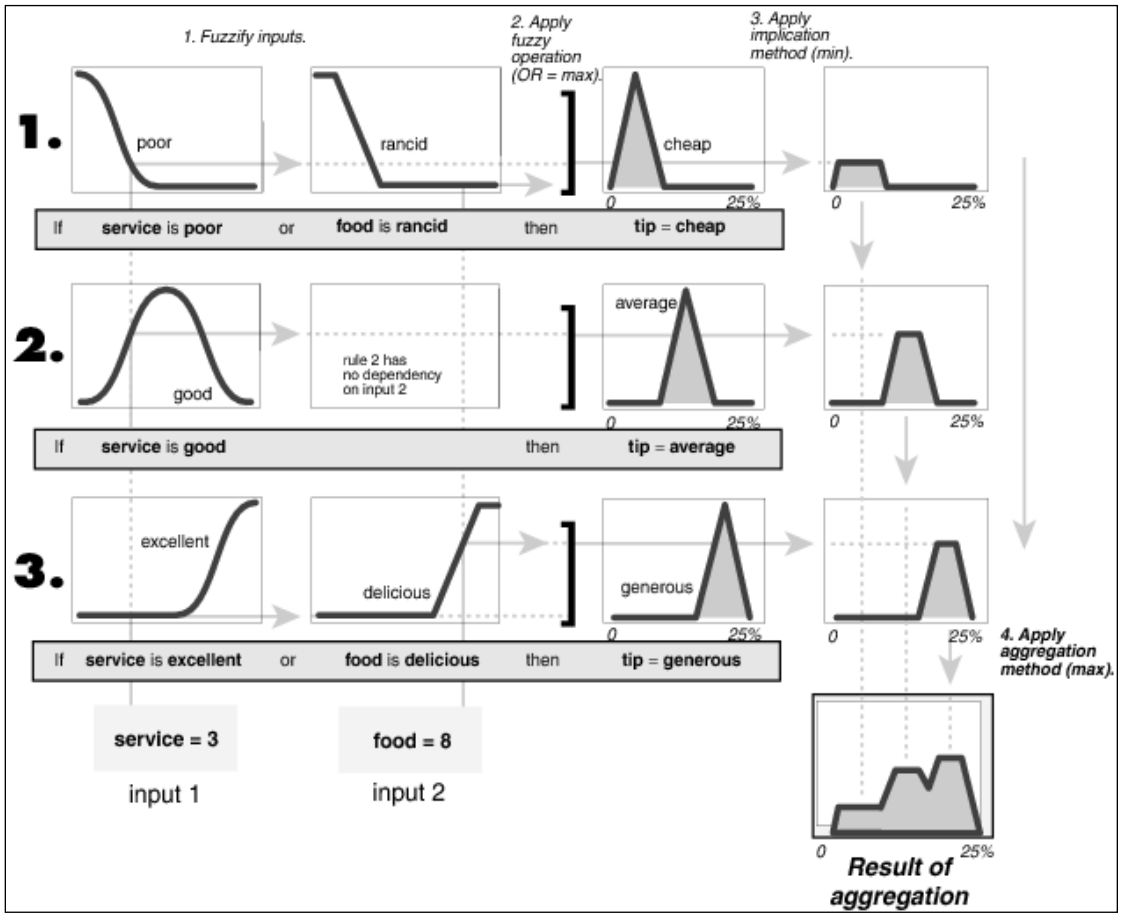

Figure 18 Aggregation of Results of the Fuzzy Rules, Source: [124]

6.1.5.1.5 Defuzzification

The output of the aggregation process is a fuzzy set but the expected output of a fuzzy system is usually a crisp value. The way how the fuzzy set is mapped to a value which is the most representative to the set is called defuzzification.

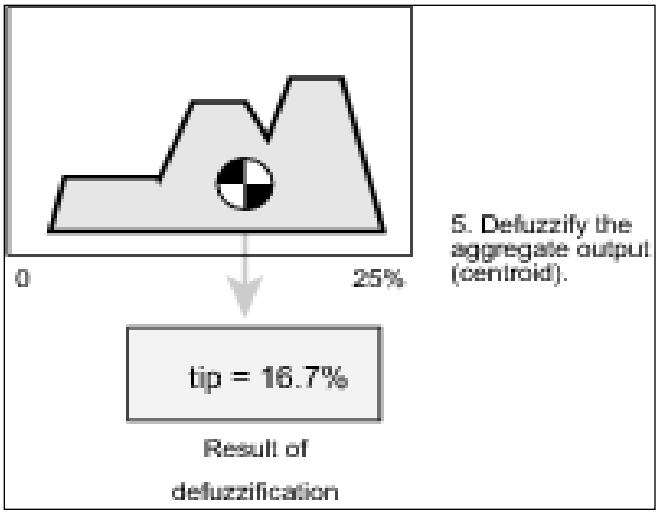

Figure 19 Defuzzification, Source: [124]

6.1.5.1.6 Defuzzification methods

Many types of defuzzification methods were developed. The two most widespread ones [107]: centre of gravity (COG) and mean of maxima (MOM) represent two different approaches. While COG, like other centriod methods, considers the area under the membership function, MOM, like other maxima methods: first of maxima, mean of maxima i.e. average, centre of maxima i.e. median, last of maxima, considers only the maxima of the membership function.

Centre of Gravity [107]:
Let A be fuzzy set on the universe X, where $x \in X$, then:

$$COG(A) = \frac{\int \mu_A(x) x dx}{\int \mu_A(x) dx}$$

COG produces monotonic output thus no oscillations occur in the defuzzified value and the output range is not used in full length. COG is a computationally expensive operation [107].

Mean of Maxima:

Let A be a fuzzy set on the universe X, where $x_i \in X$ and represents the places of maxima of μ_A then:

$$MOM(A) = \frac{\sum_{i=1}^{n} \mu_A(x_i)}{n}$$

MOM can produce oscillation in the output value but the method uses the full output range. MOM is a computationally inexpensive operation [107].

The selection of an appropriate defuzzification method of a fuzzy system can span from trial-and-error approach to more formal approaches based on application specific properties. Runkler in [107] classified the properties of defuzzification methods in four different categories: (1) static, (2) dynamic, (3) statistical, (4) implementation properties. The designer of the fuzzy system needs to analyse that the specific application which type of defuzzification properties has to possess and select the appropriate method to satisfy the specified properties.

6.1.5.1.7 Inference Method of Takagi, Sugeno, and Kang

Crisp inputs need to be fuzzified as antecedents are fuzzy sets but the consequents are described by zero or first order functions. Usually the min compositional operator is used to determine the firing strength of the antecedents. The output is calculated as weighted average of the impacts of linguistic rules. Consequently, output does not need to be defuzzified as the consequents of the linguistic rules are crisp functions. The overlap of the antecedent membership functions determines the smoothness of the output behaviour [99].

The steps of inference are the same as for Mamdani's method apart from the aggregation and defuzzification where weighted averages are calculated instead:

1. Fuzzifying the inputs,
2. Applying compositional operators for the antecedents
3. Implication

4. Calculating averages coming from the output of each linguistic rule.

Advantages [99], [125]:

- It is computationally efficient as the expensive defuzzification is not part of the inference process.
- It works well with optimization and adaptive techniques. Automatic rule generation is supported from input and output data sets.
- It has guaranteed continuity of the output surface.
- It works well with linear techniques as PID control

Disadvantages [99]:

- It does not follow the fuzziness of the underlying problem from the input to the output as the consequent parts of the linguistic rules are described by crisp functions.

The formal description of linguistic rule for the Takagi-Sugeo-Kang approach can be summarized as the following [126]:

$IF\ input1 = x\ AND\ input2 = y\ THEN\ output1 = ax + by + c$

Where a, b and c are constants. In the case of zero-order functions only a c constant is applied i.e. a=b=0.

$Output = \frac{\sum w*z}{\sum w}$ [126]

The same example used also for illustrating Mamdani's method is shown below to depict the inference method of Takagi, Sugeno and Kang.

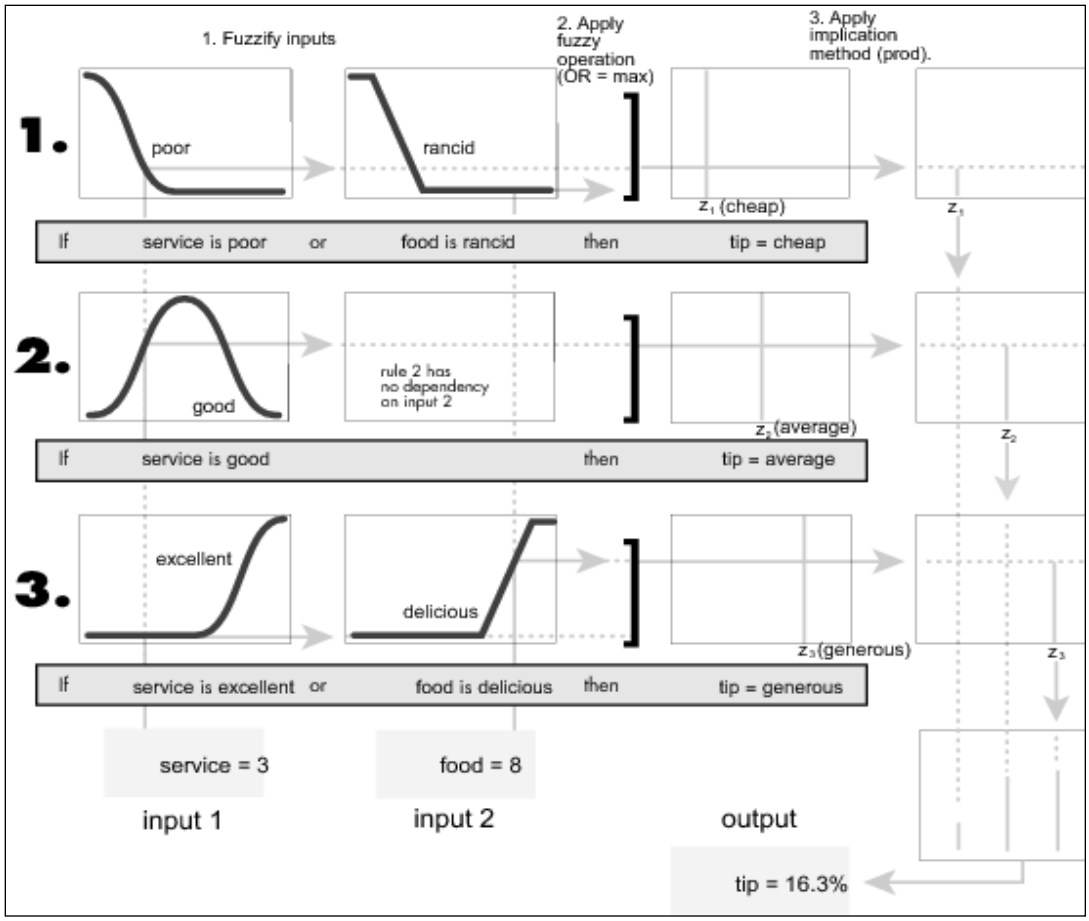

Figure 20 Takagi-Sugeno-Kang Inference Process, Source: [126]

6.1.5.1.8 Inference Method of Tsukamoto

Tsukamoto's method is similar to the method of Takagi, Sugeno and Kang but the consequent part of the linguistic rules are described with fuzzy sets, which have monotonic membership functions. The overall output value is calculated by the weighted average of the crisp values resulted from the output membership functions [100], [106].

The inference method attempts to merge the advantages of the previous two methods and avoid the disadvantages. The computationally expensive defuzzification is omitted and the fuzziness is followed from the input to the output. The method is not widespread [100].

6.1.5.2 Type-2 Fuzzy Logic

Type-1 fuzzy logic possesses crisp membership functions as introduced in the review before. This has the connotation that type-1 fuzzy systems cannot directly handle such kind of uncertainties as

- meaning of words used in the if-the rules,
- if the knowledge is gained from experts who disagree,
- noise is added to the input measurement data,
- data to adjust parameters of a type-1 fuzzy logic can also contain noise [97].

An endeavour of type-2 fuzzy logic is to consider the effect of uncertainties in type-1 fuzzy logic. Type-2 fuzzy logic is able to deal with this effect and minimize it because type-2 fuzzy logic can model the listed source of uncertainties [100].

Type-2 fuzzy logic possesses membership functions which are not crisp but encompass an area called footprint of uncertainty [97]. A secondary grade of membership is associated to each point of this area forming a three-dimensional function. If the secondary membership values are all one unit, then the logic described is an interval type-2 fuzzy logic.

Let, $x \in X$, be an element of the universe X that belongs to the fuzzy set, $\tilde{A}$. The grade of membership can be expressed by a type-2 membership function: $\mu_{\tilde{A}}(x,u)$, where $u \in [0,1]$. J_x is the set of possible u values and is called the primary membership of x; moreover, it is the domain of the secondary membership functions [97]:

$$\tilde{A} = \{((x,u), \mu_A(x,u)) | \forall x \in X, \quad \forall u \in J_x \in [0,1], \ \forall \mu_A(x,u) \in [0,1]\}$$

6.1.5.2.1 Terms

Upper membership function [100]: the membership function which describes the highest primary membership values.

Lower membership function [100]: the membership function which describes the lowest primary membership values.

Footprint of Uncertainty: the area of primary grade membership values surrounded by the lower and upper membership functions. It can mathematically be expressed using the above context in the following manner [97]:

$$FOU(\tilde{A}) = \bigcup_{x \in X} J_x$$

6.1.5.2.2 Basic Operations

The basic operations union, intersection and complement will be introduced below. Union is also called join meanwhile intersection is also called meet.

6.1.5.2.2.1 Union, Join

Let $\widetilde{A}$ and $\tilde{B}$ be two type-2 fuzzy sets on X, where $x \in X$. The grade of membership can be expressed by a type-2 membership function: $\mu_{\tilde{A}}(x,u)$ or $\mu_{\tilde{B}}(x,u)$, where $u \in [0,1]$. J_x is the set of possible u values and is called the primary membership of x; moreover, it is the domain of the secondary membership functions.

The fuzzy sets need first be discretised. The following equation describes the union operation which is also called join [97]:

$$\tilde{A} \cup \tilde{B} \Leftrightarrow \mu_{\tilde{A} \cup \tilde{B}}(x,u) = \sum_{x \in X} \frac{\mu_{\tilde{A} \cup \tilde{B}}(x)}{x} \equiv \sum_{u,w \in J_x} \frac{f_x(u) \star g_x(w)}{u \vee w} \equiv \mu_{\tilde{A}}(x) \sqcup \mu_{\tilde{B}}(x)$$

The $\star$ operator designates the t-norm operation. Several t-norm implementations are known, the simplest one is: min. Klir and Youan present an extensive list to realize the t-norm operation [127][12].

6.1.5.2.2.2 Intersection, Meet

Let $\widetilde{A}$ and $\tilde{B}$ be two type-2 fuzzy sets on X, where $x \in X$. The grade of membership can be expressed by a type-2 membership function: $\mu_{\tilde{A}}(x,u)$ or $\mu_{\tilde{B}}(x,u)$, where $u \in [0,1]$. J_x is the set of possible u values and is called the primary membership of x; moreover, it is the domain of the secondary membership functions.

The fuzzy sets need first be discretised. The following equation describes the intersection operation which is also called meet [97]:

$$\tilde{A} \cap \tilde{B} \Leftrightarrow \mu_{\tilde{A}\cap\tilde{B}}(x,u) = \sum_{x\in X} \frac{\mu_{\tilde{A}\cap\tilde{B}}(x)}{x} \equiv \sum_{u,w\in J_x} \frac{f_x(u) \star g_x(w)}{u \wedge w} \equiv \mu_{\tilde{A}}(x) \sqcap \mu_{\tilde{B}}(x)$$

The $\star$ operator designates the t-norm operation. Several t-norm implementations are known, the simplest one is: min. Klir and Youan present an extensive list to realize the t-norm operation [127][13].

6.1.5.2.2.3 Complement

Let $\widetilde{A}$ a type-2 fuzzy set on X, where $x \in X$. The grade of membership can be expressed by a type-2 membership function: $\mu_{\tilde{A}}(x,u)$ where $u \in [0,1]$. J_x is the set of possible u values and is called the primary membership of x; moreover, it is the domain of the secondary membership functions.

The fuzzy set needs first be discretised. The following equation describes the complement operation [97]:

$$\bar{\tilde{A}} \Leftrightarrow \mu_{\bar{\tilde{A}}}(x,u) = \sum_{x\in X} \frac{\mu_{\bar{\tilde{A}}}(x)}{x} \equiv \sum_{u\in J_x} \frac{f_x(u)}{1-u} \equiv \urcorner \mu_{\tilde{A}}(x)$$

[12] Page 74
[13] Page 74

6.1.5.2.3 Defuzzification

Defuzzification of type-2 fuzzy systems forms an intensively researched area. Major interval type-2 fuzzy sets may undergo the following defuzzification techniques:

- Karnik-Mendel Iterative Procedure [128]
- Greenfield-Chiclana Collapsing Defuzzifier [129]
- Nie-Tan Method [130]

Major general type-2 fuzzy systems may apply the following procedures:

- Exhaustive Defuzzification [131][14]
- Wavy Slice Representation [97]
- Geometric Defuzzification [132]
- Alpha-Plane Method [133]
- Sampling Method [131]

Karnik-Mendel Iterative Procedure is widely used and improvements are also introduced [134]. The Greenfield-Chiclana Collapsing Defuzzifier outperformed the Karnik-Mendel Iterative Procedure in runtime and even in accuracy for certain test sets [129]. Nie-Tan Method has lower computational costs then Karnik-Mendel Iterative Procedure [123].

Exhaustive defuzzification mechanism has high computational costs which prevents it to be used in real-life applications. Coupland and John achieved the accuracy and performance desired for fuzzy logic controllers with geometric defuzzification [132]. Coupland, Gongora, John and Wills performed a comparative study [96] to investigate the accuracy of type-1, interval type-2 and general type-2 fuzzy controllers where the defuzzification mechanism was implemented by geometric defuzzification for the type-2 controllers. The authors measured that all controllers performed the tasks within acceptable boundaries but the accuracy showed remarkable differences:

1. The type-1 fuzzy controller achieved the largest average error but the extent of the error was consistent

[14] Page 7

2. The interval type-2 fuzzy controller produced smaller average error but the extent of the error was inconsistent
3. General type-2 fuzzy controller achieved the smallest average error and the extent of the error was consistent

Experiments in [135] verified that both Karnik-Mendel Iterative Procedure, which was combined with the Alpha-Plane [133] method, and the Sampling Method achieved acceptable accuracy in comparison to the Exhaustive Defuzzification while maintaining the desired performance for fuzzy logic controllers. Alpha-Plane and Wavy Slice representations are from theoretical point of view for type-2 fuzzy sets equivalent [133].

Avoiding the complexity of the defuzzification methods implicated also use cases where two type-1 fuzzy controllers, one of which performed calculations with lower membership functions and the other one with the upper membership functions, were applied to implement an interval type-2 fuzzy controller [100], which showed better results than the two controllers individually.

6.1.5.3 Type-n Fuzzy Logic

Fuzzy logic can also be extended to type-n. Type-n fuzzy logic is where the membership functions are type n-1 functions. Basics of fuzzy logic do not change from type-1 to type-n. The higher type only indicates a greater number of fuzziness. The structure of the linguistic rules is the same because the greater uncertainty is associated with the membership functions. The structure of a type-2 fuzzy inference system is the same as the structure of type-1 fuzzy systems; however, defuzzification needs to be extended [100].

6.1.5.4 Fuzzy Modelling

Constructing a fuzzy inference system is called fuzzy modelling [99]. Fuzzy modelling exhibits the features below [99]:

1. Incorporating human expertise which would cause difficulties with other modelling techniques
2. Input-output data sets can also be used as with other mathematical modelling techniques

The steps of modelling [99], [100]:

1. Selecting relevant input and output variables
2. Choosing a specific type of implication method
3. Determine the number of linguistic terms to be associated with each input and output. In the case of Takagi-Sugeno-Kang inference the determination of the order of output equations is also necessary.
4. Designing the collection of if-then rules
5. Choosing appropriate type of membership functions
6. Interviewing human experts to determine the parameters of the membership functions in the rule base
7. Refining the parameters of the membership functions using regression and optimization techniques if input and output data sets are available.

The order of the above steps provides one possible implementation. Fuzzy modelling can also be performed with the same steps in different order or depending on the nature of the problem some of the steps can be omitted.

6.1.5.5 Adaptive Fuzzy Systems

Traditional methods for developing type-1 membership functions are difficult. Knowledge acquisition from experts is usually followed by making consensus among the different opinions. If no consensus is possible, then statistical methods can be utilized to build the membership functions. This method is outperformed by application of neural networks to gain the membership functions from a set of data. Developing the membership functions with one expert and optimizing them with neural network is also possible [136].

In addition, if the input space partitioning is fine-granular, then many variations of the linguistic variables are possible for linguistic rule construction. Thus, either parsimonious

fuzzy systems need to be created to avoid the manual construction with untreatable number of fuzzy rules or adaptive techniques need to be exploited to generate the fuzzy rules.

Jang published an adaptive-neural-network-based type-1 fuzzy inference system in [137]. The described fuzzy inference system became widespread under the name Adaptive Neuro-Fuzzy Inference System (ANFIS). The neural network must be a feed-forward type network with five layers. It is possible to use it with all three kinds of reasoning, the extension from Takagi-Sugeno-Kang method to the method of Tsukamoto is straightforward but the application of Mamdani's model is complex [99].

If the input-output data set is large, then fine-tuning the membership functions is recommended. However, if the input-output data set is small, then human-determined linguistic rules and membership functions need to be added and fixed throughout the learning process as they might represent important information which is missed by the data sets.

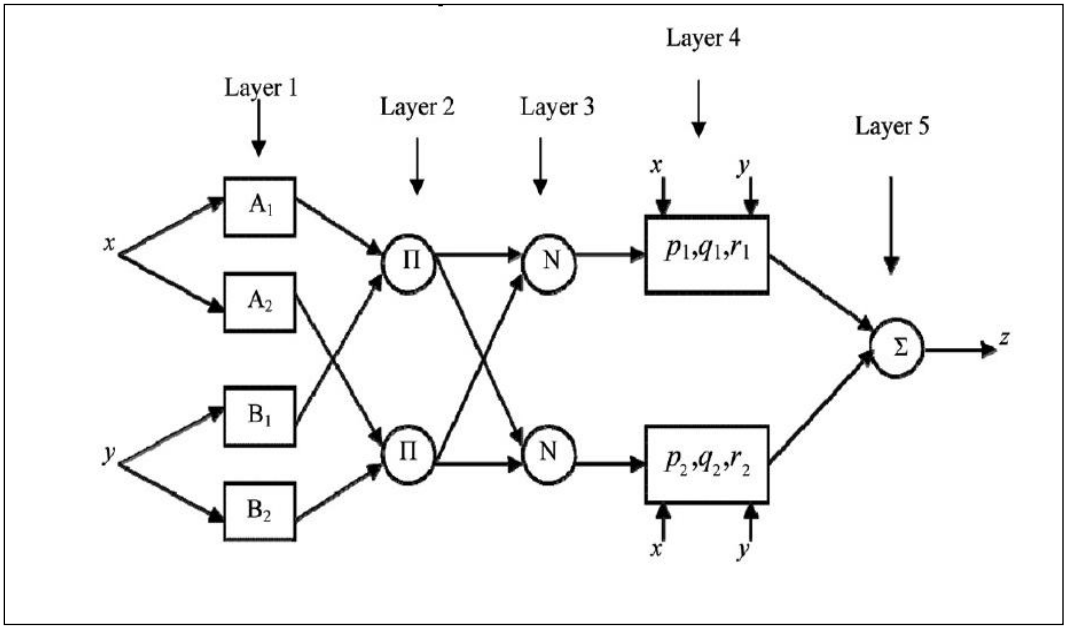

Figure 21: Adaptive Neuro-Fuzzy Inference System, Source: [138]

The above illustration depicts a Takagi-Sugeno-Kang type ANFIS where the output is described by a first-order function: $px + yq + r$

Jang et al. present a comparison of the back-propagation multilayer perceptron (MLP) with quick propagation and ANFIS [99]. Quick propagation is considered to be one of the best training algorithms. ANFIS produced better results and outperformed also MLP with predicting chaotic time series.

Method	Training Cases	Error Index Value
ANFIS	500	0.007
Cascaded-correlation Neural Network	500	0.06
Backpropagation MLP	500	0.02
6^{th} Order Polynomial	500	0.04
Linear Predictive Method	2000	0.55

Table 10 Comparison of the Generalization Capability of Neural Networks, source: [99, page 360]

The remarkable generalization capability of ANFIS derives from the facts [99]:

- ANFIS can achieve a highly non-linear mapping therefore it suits better than common linear methods in modelling non-linear time series
- The membership functions are not based on a priori knowledge but they are intuitively reasonable and cover the whole input space; this results in fast convergence to good parameter values
- Support for Minimal Disturbance Principle i.e. the adaptation should not only reduce the error of the current training pattern but also cause minimal disturbance to the responses already learned.

ANFIS forms a linguistically understandable fuzzy inference system which can embed a prior knowledge and allows the possibility to understand the results of learning in

contrast to the neural networks where the network realises a black-box as the weights cannot be interpreted in a linguistically understandable manner [99].

6.1.5.5.1 Fuzzy Inference System Construction wit ANFIS

Constructing a Takagi-Sugeno-Kang fuzzy inference system with ANFIS comprises of the following steps [139]:

1. Identification of ANFIS inputs
2. Collection of training and testing data
3. Selecting the type of membership functions
4. Determining the order of the output function
5. Selection of learning algorithms
6. Testing with different approaches
7. Optimizing the number of fuzzy rules

Constructing the fuzzy inference system is an iterative process as it also includes several attempts of the training and with changes in the number of the membership functions of the input and output to minimize the root mean square error (RMSE[15]). The testing data need to be different from the training data. Finally, conclusions on testing need to be drawn to determine whether the generalization of the model is acceptable i.e. whether it can be used in general with minimal errors [139].

6.1.5.5.2 Adaptive Fuzzy Perception Learner

John in [136] published an adaptive type-2 fuzzy system, under the name Adaptive Fuzzy Perception Learner (AFPL), which applies the philosophy of ANFIS in type-2 fuzzy context. Instead of numbers words are used to model human perceptions [136].

Adaptive type-2 fuzzy systems are not widespread; however, they would be very timely. The tool support for AFPL is less mature than the support for ANFIS, which has been existed for longer time.

[15] RMSE is an indicator used frequently in statistics [140].

6.1.6 Summary

On basis of the literature review both type-1 and type-2 fuzzy logic proved to be feasible either with Mamdani's or Takaki-Sugeno-Kang's inference methods. Moreover, the literature review showed that adaptive techniques for generating the rule base automatically or semi-automatically for a fuzzy system are more mature in the type-1 fuzzy domain.

With individual collection of the fuzzy rules involving interviews with many experts, it is easy to introduce contradictions in the rule base; on the other hand, the method is effort-intensive. Synthesising the rule base in an adaptive manner helps to avoid contradictions among the rules. These findings result in the proposal of the selection of type-1 fuzzy logic with the ANFIS approach for performing the model construction for execution tracing.

This section shows that fuzzy logic is able to deal with the subjective uncertainty referring to the present which is addressed in the context of this research as an intrinsic attribute of the quality measurement process. Moreover, this section gives a brief review on fuzzy logic but not on other mathematical tools that deal with the objective manifestation of uncertainty or with its subjective manifestation referring to the future.

6.2 Matlab Charts of the Pilot Models

This section introduces the Matlab charts and Matlab code of the quality models built for execution tracing in the scope of the pilot study because chapter 4 Pilot Study presents the model only which performed the best against the defined criteria, moreover provides a comparison and a short summary on them.

6.2.1 Mamdani's Approach with Mean of Maxima Defuzzification Technique with Gaussian Membership Functions

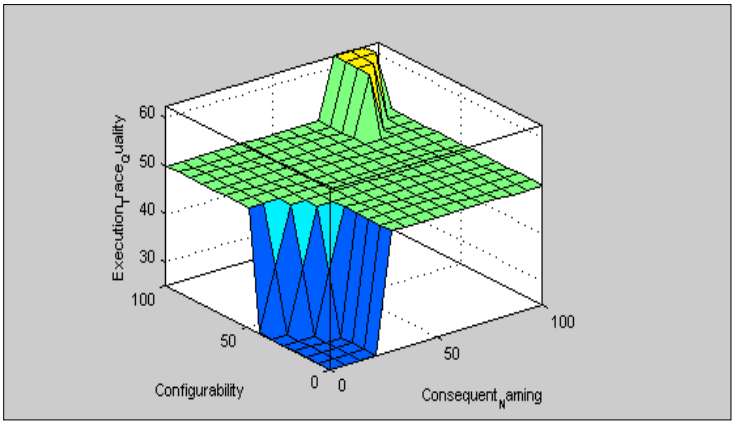

Figure 22 Mamdani's Approach with Gaussian Membership Functions (Configurability and Consequent Naming vs. Execution Tracing Quality)

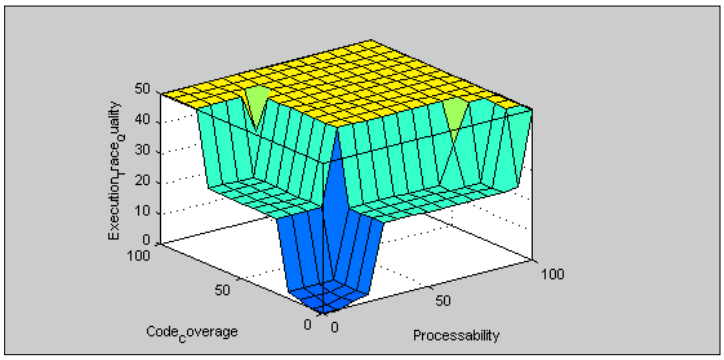

Figure 23 Mamdani's Approach with Gaussian Membership Functions (Code Coverage and Processability vs. Execution Tracing Quality)

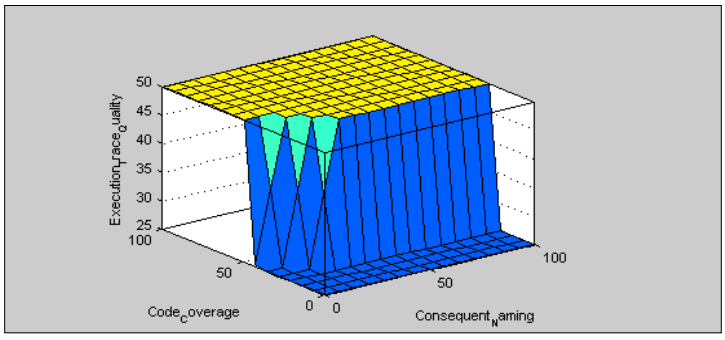

Figure 24 Mamdani's Approach with Gaussian Membership Functions (Code Coverage and Consequent Naming vs. Execution Tracing Quality)

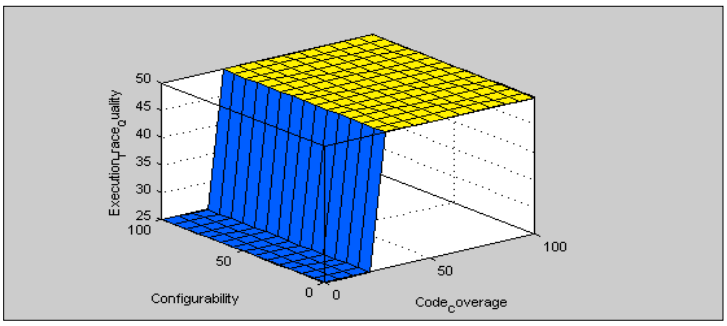

Figure 25 Mamdani's Approach with Gaussian Membership Functions (Configurability and Code Coverage vs. Execution Tracing Quality)

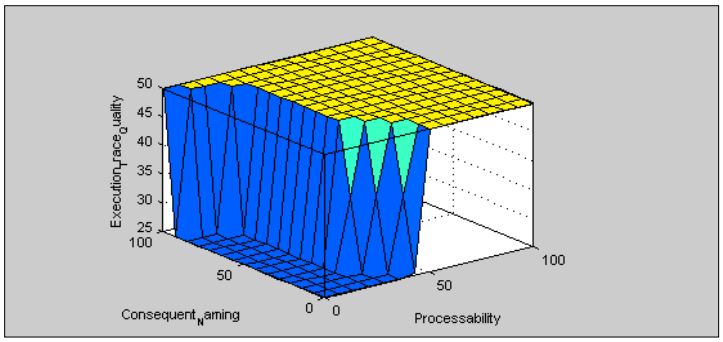

Figure 26 Mamdani's Approach with Gaussian Membership Functions (Consequent Naming and Processability vs. Execution Tracing Quality)

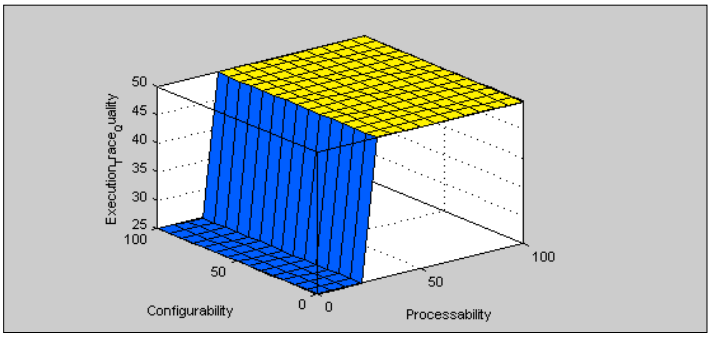

Figure 27 Mamdani's Approach with Gaussian Membership Functions (Configurability and Processability vs. Execution Tracing Quality)

6.2.2 Mamdani's Approach with Mean of Maxima Defuzzification Technique with Triangular-shaped Membership Functions

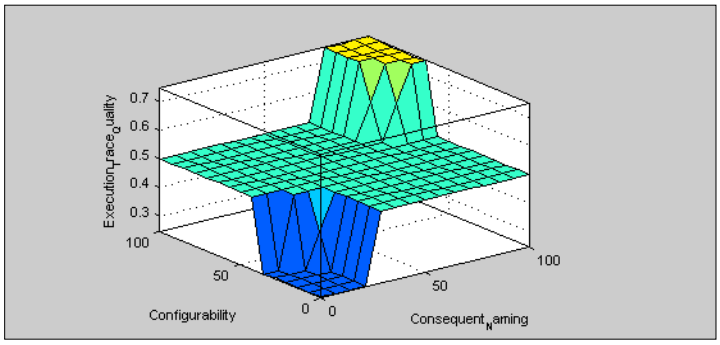

Figure 28 Mamdani's Approach with Triangular-shaped Membership Functions (Configurability and Consequent Naming vs. Execution Tracing Quality)

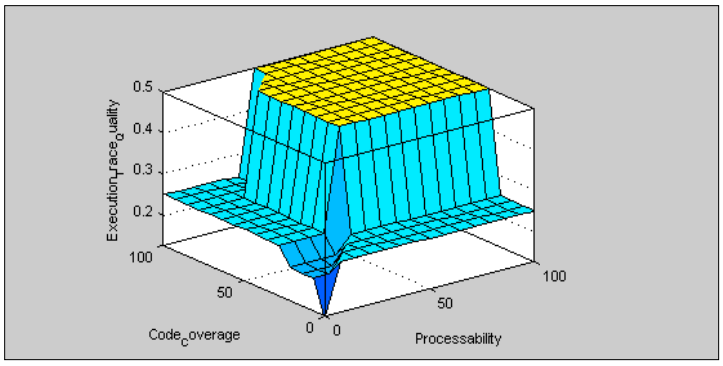

Figure 29 Mamdani's Approach with Triangular-shaped Membership Functions (Code Coverage and Processability vs. Execution Tracing Quality)

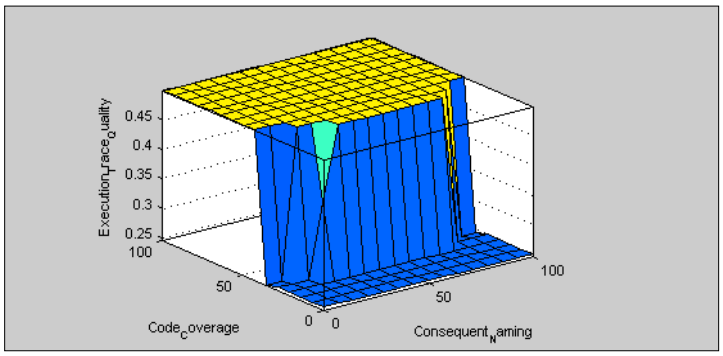

Figure 30 Mamdani's Approach with Triangular-shaped Membership Functions (Code Coverage and Consequent Naming vs. Execution Tracing Quality)

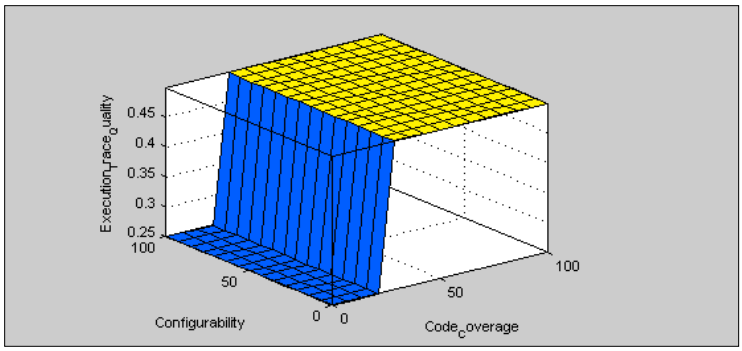

Figure 31 Mamdani's Approach with Triangular-shaped Membership Functions (Configurability and Code Coverage vs. Execution Tracing Quality)

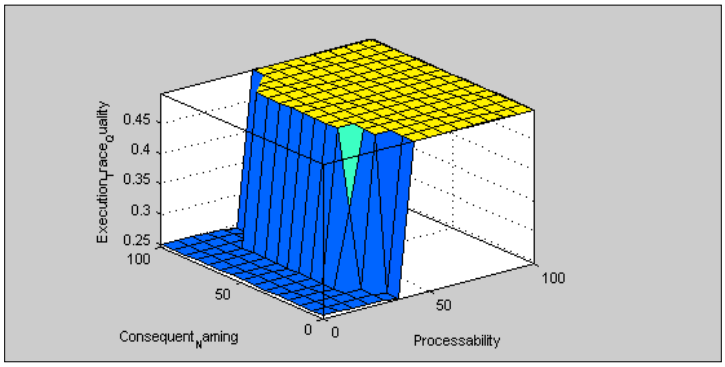

Figure 32 Mamdani's Approach with Triangular-shaped Membership Functions (Consequent Naming and Processability vs. Execution Tracing Quality)

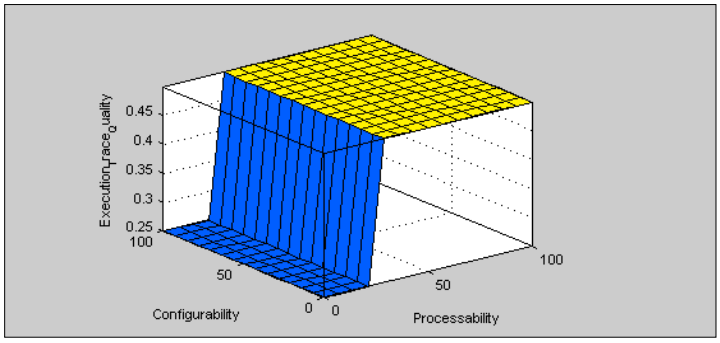

Figure 33 Mamdani's Approach with Triangular-shaped Membership Functions (Configurability and Processability vs. Execution Tracing Quality)

6.2.3 Approach of Takagi-Sugeno-Kang with Triangular-shaped Membership Functions and with Zero-order Functions in the Output

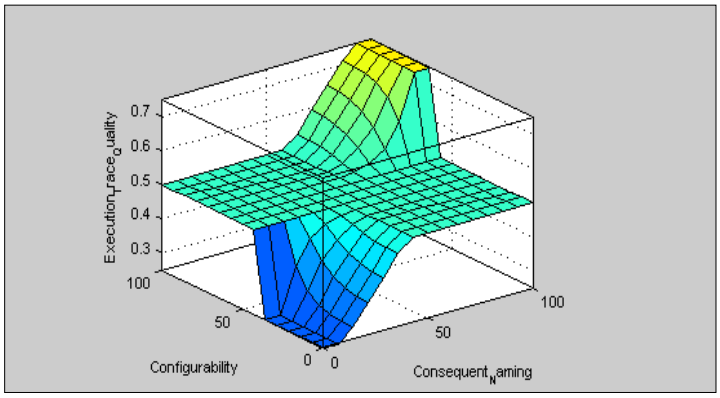

Figure 34 Approach of Takagi-Sugeno-Kang with Triangular-shaped Membership Functions (Configurability and Consequent Naming vs. Execution Tracing Quality)

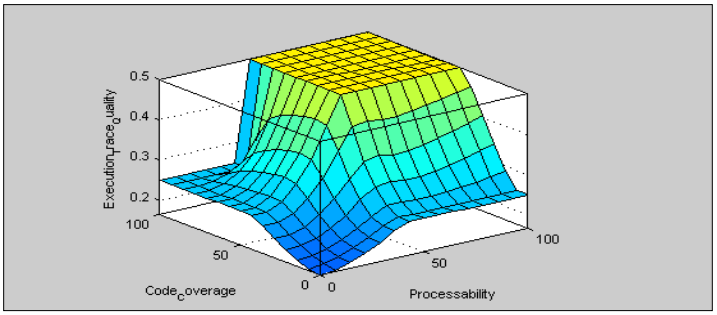

Figure 35 Approach of Takagi-Sugeno-Kang with Triangular-shaped Membership Functions (Code Coverage and Processability vs. Execution Tracing Quality)

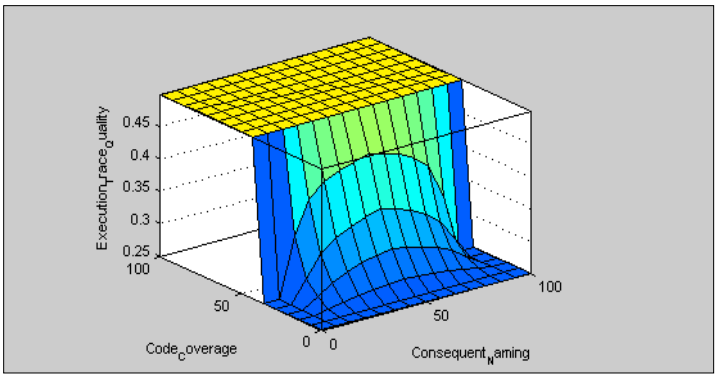

Figure 36 Approach of Takagi-Sugeno-Kang with Triangular-shaped Membership Functions (Code Coverage and Consequent Naming vs. Execution Tracing Quality)

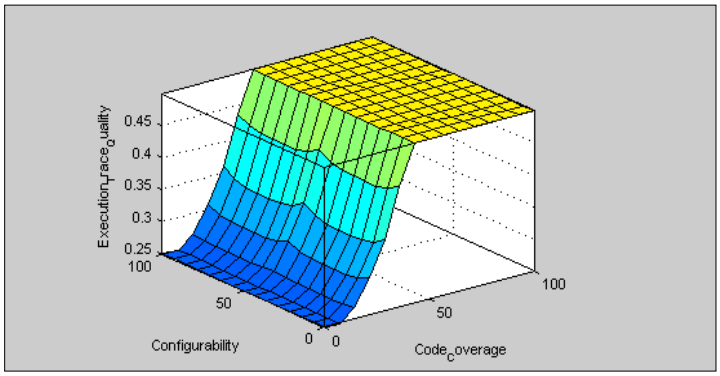

Figure 37 Approach of Takagi-Sugeno-Kang with Triangular-shaped Membership Functions (Configurability and Code Coverage vs. Execution Tracing Quality)

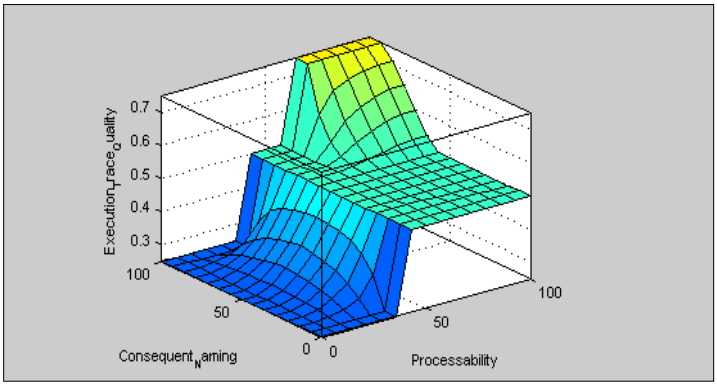

Figure 38 Approach of Takagi-Sugeno-Kang with Triangular-shaped Membership Functions (Consequent Naming and Processability vs. Execution Tracing Quality)

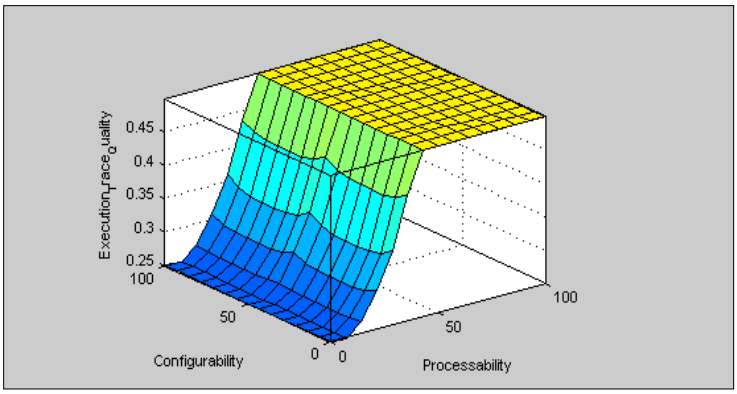

Figure 39 Approach of Takagi-Sugeno-Kang with Triangular-shaped Membership Functions (Configurability and Processability vs. Execution Tracing Quality)

6.3 MatLab Code of the Presented Fuzzy Inference Systems

6.3.1 Mamdani's Approach with Gaussian Membership Functions with Mean of Maxima Defuzzification Technique

```
[System]
Name='assignmentMamdani2GaussianMF_MoM'
Type='mamdani'
Version=2.0
NumInputs=4
NumOutputs=1
NumRules=30
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='mom'

[Input1]
Name='Processability'
Range=[0 100]
NumMFs=3
MF1='Poor':'gaussmf',[16.99 -4.441e-016]
MF2='Medium':'gaussmf',[16.99 50]
MF3='Good':'gaussmf',[16.99 100]
```

```
[Input2]
Name='Code_Coverage'
Range=[0 100]
NumMFs=3
MF1='Poor':'gaussmf',[16.99 -4.441e-016]
MF2='Medium':'gaussmf',[16.99 50]
MF3='Good':'gaussmf',[16.99 100]

[Input3]
Name='Configurability'
Range=[0 100]
NumMFs=3
MF1='Poor':'gaussmf',[16.99 -4.441e-016]
MF2='Medium':'gaussmf',[16.99 50]
MF3='Good':'gaussmf',[16.99 100]

[Input4]
Name='Consequent_Naming'
Range=[0 100]
NumMFs=3
MF1='Poor':'gaussmf',[16.99 -4.441e-016]
MF2='Medium':'gaussmf',[16.99 50]
MF3='Good':'gaussmf',[16.99 100]

[Output1]
Name='Execution_Trace_Quality'
Range=[0 100]
NumMFs=5
MF1='Very_Poor':'gaussmf',[5.516 -1.495]
MF2='Medium':'gaussmf',[10.62 50]
MF3='Very_Good':'gaussmf',[15.67 101.5]
MF4='Poor':'gaussmf',[10.36 24.97]
MF5='Good':'gaussmf',[10.62 74.74]

[Rules]
1 1 0 0, 1 (1) : 1
```

```
2 1 0 0, 4 (1) : 1
1 2 0 0, 4 (1) : 1
2 2 1 1, 4 (1) : 1
2 2 1 2, 2 (1) : 1
2 2 2 2, 2 (1) : 1
2 2 3 2, 2 (1) : 1
2 2 3 1, 2 (1) : 1
2 2 3 3, 5 (1) : 1
2 2 1 3, 2 (1) : 1
3 2 1 1, 4 (1) : 1
3 2 2 1, 2 (1) : 1
3 2 3 1, 2 (1) : 1
3 2 1 2, 2 (1) : 1
3 2 2 2, 2 (1) : 1
3 2 3 2, 2 (1) : 1
3 2 1 3, 2 (1) : 1
3 2 2 3, 2 (1) : 1
3 2 3 3, 5 (1) : 1
3 3 1 1, 2 (1) : 1
3 3 2 1, 2 (1) : 1
3 3 3 1, 5 (1) : 1
3 3 1 2, 2 (1) : 1
3 3 2 2, 2 (1) : 1
3 3 3 2, 5 (1) : 1
3 3 1 3, 2 (1) : 1
3 3 2 3, 5 (1) : 1
3 3 3 3, 3 (1) : 1
0 2 0 3, 2 (1) : 1
1 3 0 0, 2 (1) : 1
```

6.3.2 Mamdani's Approach with Triangular-shaped Membership Functions with Mean of Maxima Defuzzification Technique

```
[System]
Name='assignmentMamdani1TriangularMF_MoM'
Type='mamdani'
Version=2.0
NumInputs=4
NumOutputs=1
NumRules=31
AndMethod='min'
OrMethod='max'
ImpMethod='min'
AggMethod='max'
DefuzzMethod='mom'

[Input1]
Name='Processability'
Range=[0 100]
NumMFs=3
MF1='Poor':'trimf',[-40 0 40]
MF2='Medium':'trimf',[10 50 90]
MF3='Good':'trimf',[60 100 140]

[Input2]
Name='Code_Coverage'
Range=[0 100]
NumMFs=3
MF1='Poor':'trimf',[-40 0 40]
MF2='Medium':'trimf',[10 50 90]
MF3='Good':'trimf',[60 100 140]

[Input3]
Name='Configurability'
Range=[0 100]
NumMFs=3
MF1='Poor':'trimf',[-40 0 40]
MF2='Medium':'trimf',[10 50 90]
```

```
MF3='Good':'trimf',[60 100 140]

[Input4]
Name='Consequent_Naming'
Range=[0 100]
NumMFs=3
MF1='Poor':'trimf',[-40 0 40]
MF2='Medium':'trimf',[10 50 90]
MF3='Good':'trimf',[60 100 140]

[Output1]
Name='Execution_Trace_Quality'
Range=[0 1]
NumMFs=5
MF1='Very_Poor':'trimf',[-0.399 0 0.1]
MF2='Medium':'trimf',[0.25 0.5 0.75]
MF3='Very_Good':'trimf',[0.9 1 1.399]
MF4='Poor':'trimf',[0 0.25 0.4934]
MF5='Good':'trimf',[0.4974 0.7474 0.9974]

[Rules]
1 1 0 0, 1 (1) : 1
2 1 0 0, 4 (1) : 1
1 2 0 0, 4 (1) : 1
2 2 1 1, 4 (1) : 1
2 2 1 2, 2 (1) : 1
2 2 2 2, 2 (1) : 1
2 2 3 2, 2 (1) : 1
2 2 3 1, 2 (1) : 1
2 2 3 3, 5 (1) : 1
2 2 1 3, 2 (1) : 1
3 2 1 1, 4 (1) : 1
3 2 2 1, 2 (1) : 1
3 2 3 1, 2 (1) : 1
3 2 1 2, 2 (1) : 1
3 2 2 2, 2 (1) : 1
3 2 3 2, 2 (1) : 1
```

```
3 2 1 3, 2 (1) : 1
3 2 2 3, 2 (1) : 1
3 2 3 3, 5 (1) : 1
3 3 1 1, 2 (1) : 1
3 3 2 1, 2 (1) : 1
3 3 3 1, 5 (1) : 1
3 3 1 2, 2 (1) : 1
3 3 2 2, 2 (1) : 1
3 3 3 2, 5 (1) : 1
3 3 1 3, 2 (1) : 1
3 3 2 3, 5 (1) : 1
3 3 3 3, 3 (1) : 1
1 3 0 0, 4 (1) : 1
3 1 0 0, 4 (1) : 1
0 1 0 2, 4 (1) : 1
```

6.3.3 Approach of Takagi-Sugeno-Kang with Gaussian Membership Functions

```
[System]
Name='assignmentSugeno2GaussianMF'
Type='sugeno'
Version=2.0
NumInputs=4
NumOutputs=1
NumRules=31
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'

[Input1]
Name='Processability'
Range=[0 100]
NumMFs=3
MF1='Poor':'gaussmf',[16.99 -4.441e-016]
MF2='Medium':'gaussmf',[16.99 50]
MF3='Good':'gaussmf',[16.99 100]

[Input2]
Name='Code_Coverage'
Range=[0 100]
NumMFs=3
MF1='Poor':'gaussmf',[16.99 -4.441e-016]
MF2='Medium':'gaussmf',[16.99 50]
MF3='Good':'gaussmf',[16.99 100]

[Input3]
Name='Configurability'
Range=[0 100]
NumMFs=3
MF1='Poor':'gaussmf',[16.99 -4.441e-016]
MF2='Medium':'gaussmf',[16.99 50]
```

```
MF3='Good':'gaussmf',[16.99 100]

[Input4]
Name='Consequent_Naming'
Range=[0 100]
NumMFs=3
MF1='Poor':'gaussmf',[16.99 -4.441e-016]
MF2='Medium':'gaussmf',[16.99 50]
MF3='Good':'gaussmf',[16.99 100]

[Output1]
Name='Execution_Trace_Quality'
Range=[0 1]
NumMFs=5
MF1='Very_Poor':'constant',[0]
MF2='Poor':'constant',[0.25]
MF3='Medium':'constant',[0.5]
MF4='Good':'constant',[0.75]
MF5='Very_Good':'constant',[1]

[Rules]
1 1 0 0, 1 (1) : 1
2 1 0 0, 2 (1) : 1
1 2 0 0, 2 (1) : 1
2 2 1 1, 2 (1) : 1
2 2 1 2, 3 (1) : 1
2 2 2 2, 3 (1) : 1
2 2 3 2, 3 (1) : 1
2 2 3 1, 3 (1) : 1
2 2 3 3, 4 (1) : 1
2 2 1 3, 3 (1) : 1
3 2 1 1, 2 (1) : 1
3 2 2 1, 3 (1) : 1
3 2 3 1, 3 (1) : 1
3 2 1 2, 3 (1) : 1
3 2 2 2, 3 (1) : 1
3 2 3 2, 3 (1) : 1
```

```
3 2 1 3, 4 (1) : 1
3 2 2 3, 4 (1) : 1
3 2 3 3, 4 (1) : 1
3 3 1 1, 3 (1) : 1
3 3 2 1, 3 (1) : 1
3 3 3 1, 4 (1) : 1
3 3 1 2, 3 (1) : 1
3 3 2 2, 3 (1) : 1
3 3 3 2, 4 (1) : 1
3 3 1 3, 3 (1) : 1
3 3 2 3, 4 (1) : 1
3 3 3 3, 5 (1) : 1
2 3 3 3, 3 (1) : 1
1 0 0 3, 3 (1) : 1
0 1 0 2, 2 (1) : 1
```

6.3.4 Approach of Takagi-Sugeno-Kang with Triangular-shaped Membership Functions

```
[System]
Name='assignmentSugeno1TriangularMF'
Type='sugeno'
Version=2.0
NumInputs=4
NumOutputs=1
NumRules=30
AndMethod='prod'
OrMethod='probor'
ImpMethod='prod'
AggMethod='sum'
DefuzzMethod='wtaver'

[Input1]
Name='Processability'
Range=[0 100]
NumMFs=3
MF1='Poor':'trimf',[-40 0 40]
MF2='Medium':'trimf',[10 50 90]
MF3='Good':'trimf',[60 100 140]

[Input2]
Name='Code_Coverage'
Range=[0 100]
NumMFs=3
MF1='Poor':'trimf',[-40 0 40]
MF2='Medium':'trimf',[10 50 90]
MF3='Good':'trimf',[60 100 140]

[Input3]
Name='Configurability'
Range=[0 100]
NumMFs=3
MF1='Poor':'trimf',[-40 0 40]
MF2='Medium':'trimf',[10 50 90]
```

```
MF3='Good':'trimf',[60 100 140]

[Input4]
Name='Consequent_Naming'
Range=[0 100]
NumMFs=3
MF1='Poor':'trimf',[-40 0 40]
MF2='Medium':'trimf',[10 50 90]
MF3='Good':'trimf',[60 100 140]

[Output1]
Name='Execution_Trace_Quality'
Range=[0 1]
NumMFs=5
MF1='Very_Poor':'constant',[0]
MF2='Poor':'constant',[0.25]
MF3='Medium':'constant',[0.5]
MF4='Good':'constant',[0.75]
MF5='Very_Good':'constant',[1]

[Rules]
1 1 0 0, 1 (1) : 1
2 1 0 0, 2 (1) : 1
1 2 0 0, 2 (1) : 1
2 2 1 1, 2 (1) : 1
2 2 1 2, 3 (1) : 1
2 2 2 2, 3 (1) : 1
2 2 3 2, 3 (1) : 1
2 2 3 1, 3 (1) : 1
2 2 3 3, 4 (1) : 1
2 2 1 3, 3 (1) : 1
3 2 1 1, 2 (1) : 1
3 2 2 1, 3 (1) : 1
3 2 3 1, 3 (1) : 1
3 2 1 2, 3 (1) : 1
3 2 2 2, 3 (1) : 1
3 2 3 2, 3 (1) : 1
```

```
3 2 1 3, 4 (1) : 1
3 2 2 3, 4 (1) : 1
3 2 3 3, 4 (1) : 1
3 3 1 1, 3 (1) : 1
3 3 2 1, 3 (1) : 1
3 3 3 1, 4 (1) : 1
3 3 1 2, 3 (1) : 1
3 3 2 2, 3 (1) : 1
3 3 3 2, 4 (1) : 1
3 3 1 3, 3 (1) : 1
3 3 2 3, 4 (1) : 1
3 3 3 3, 5 (1) : 1
0 1 0 2, 2 (1) : 1
1 0 0 2, 2 (1) : 1
```

6.4 Bibliography

[1] A. Kahn, Simulation of Message Passing Programs, [Online] http://may.cs.ucla.edu/projects/sesame/publications/sundeep_diss_html/node43.html, University of California, 1997.

[2] IBM, “Understanding Execution Traces, [Online], [Accessed: 05.02.2013], Available from: http://pic.dhe.ibm.com/infocenter/brdotnet/v7r1/index.jsp?topic=%2Fcom.ibm.websphere.ilog.brdotnet.doc%2FContent%2FBusiness_Rules%2FDocumentation%2F_pubskel%2FRules_for_DotN”.

[3] A. Martin, “Debugger, Real-Time Trace, Logic Analyser, Long-Term Trace ETMv3, [Online], [Accessed: 05.02.2013], Available from: http://www.lauterbach.com/publications/long_term_trace_etmv3.pdf, Lauterbach GmbH,” 2009.

[4] SAP, “Using the Technical Trace and Log, [Online], [Accessed: 05.02.2013], Available from: http://help.sap.com/saphelp_nwpi71/helpdata/en/3a/63e540aa827e7fe10000000a1550b0/content.htm”.

[5] Microsoft Co., “Tracing WMI Activity (Windows), [Online], [Accessed: 05.02.2013], Available from: http://msdn.microsoft.com/en-us/library/windows/desktop/aa826686%28v=vs.85%29.aspx,” 2012.

[6] D. Hovemeyer and W. Pugh, “Finding Bugs Is Easy,” in *OOPSLA: Object-Oriented Programming, Systems, Languages & Applications, [Online], [Accessed: 14.02.2012], Available from: http://www.cs.nyu.edu/~lharris/papers/findbugsPaper.pdf*, 2004.

[7] M. D. Ernst, “Static and Dynamic Analysis: Synergy and Duality,” *In Proceedings ICSE Workshop on Dynamic Analysis,* pp. 24-27., 2003..

[8] M. Young, “Symbiosis of Static Analysis and Program Testing,” *In Proc. 6th International Conference on Fundamental Approaches to Software Engineering,* pp. 1-5, 2003..

[9] C. Csallner and Y. Smaragdakis, “DSD-Crasher: A Hybrid Analysis Tool for Bug Finding,” in *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2008.

[10] A. Spillner, T. Linz and H. Schaefer, Software Testing Foundations, Santa Barbara, CA: Rockey Nook Inc., 2007.

[11] J. Koskinen, “Software Maintenance Costs,” [Online], 2010, [Accessed: 23.01.2012], Available from: http://users.jyu.fi/~koskinen/smcosts.htm.

[12] R. D. Banker and S. Slaughter, “A Field Study of Scale Economies in Software Maintenance,” *Management Science,* vol. 43, no. 12, pp. 1709-1725, 1997.

[13] M. S. Krishnan, T. Mukhopadhyay and C. H. Kriebel, “A Decision Model for Software Maintenance,” *Information Systems Research,* vol. 15, no. 4, p. 396–412, 2004.

[14] I. Buch and R. Park, “Improve Debugging and Performance Tuning with ETW,” *MSDN Magazine, [Online], [Accessed: 01.01.2012], Avaliable from: http://msdn.microsoft.com/en-us/magazine/cc163437.aspx,* 2007.

[15] V. Uzelac, A. Milenkovic, M. Burtscher and M. Milenkovic, “Real-time

Unobtrusive Program Execution Trace Compression Using Branch Predictor Events," *CASES 2010 Proceedings of the 2010 international conference on Compilers, Architectures and Synthesis for Embedded Systems, ISBN: 978-1-60558-903-9,* 2010.

[16] P. Godefroid and N. Nagappan, "Concurrency at Microsoft – An Exploratory Survey, [Online], 2007, [Accessed: 24.01.2012.], Available from: http://research.microsoft.com/en-us/um/people/pg/public_psfiles/ec2.pdf," Microsoft Research.

[17] R. Laddad, AspectJ in Action, Manning, MEAP, Second Edition, 2009.

[18] D. Qu, A. Roychoudhury, Z. Lang and K. Vaswani, "Darwin: An Approach for Debugging Evolving Programs," , [Online], 2009, [Accessed: 24.01.2012], Availavle from: http://research.microsoft.com/apps/pubs/default.aspx?id=80898.

[19] A. Karahasanovic and R. Thomas, "Difficulties Experienced by Students in Maintaining Object-oriented Systems: An Empirical Study," *Proceedings of the 9th Australasian Conference on Computing Education,* pp. 81-87, 2007.

[20] Z. Shi, "Visualizing Execution Traces, Master Thesis," *[Online], [Accessed: 17.05.2011], Available from: http://www.mcs.vuw.ac.nz/comp/graduates/archives/mcompsc/reports/2004/Zhenyu-Shi-final-report.pdf,* 2005..

[21] O. Spinczyk, D. Lehmann and M. Urban, "AspectC++: an AOP Extension for C++," *Software Developers Journal,* pp. 68-74, 2005.

[22] D. Panda, R. Rahman and D. Lane, EJB 3 in Action, Manning Publications Co., 2007.

[23] D. Winterfeldt, "Spring by Example," [Online], [Accessed: 19.12.2012], Available from: http://www.springbyexample.org/examples/aspectj-ltw-spring-config.html.

[24] Research Triangle Institute, "RTI Project Number 7007.011, The Economic Impacts of Inadequate Infrastructure for Software Testing," National Institute of Standards and Technology, U.S Department of Commerce, Technology Administration, [Online], 2002, [Accessed: 20.01.2012], Avaliable from: http://www.nist.gov/director/planning/upload/report02-3.pdf, 2002.

[25] International Organization for Standardization, "ISO/IEC 25021:2007, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- Quality measure elements," 2007.

[26] G. J. Klir, U. H. St.Clair and B. Yuan, Fuzzy Set Theory Fundations and Applications, Prentice Hall Ptr, 1997.

[27] L. Csernyak, G. Horvath, J. Horvath, S. Lorincz, S. Molnar and A. Stern, Matematika a Kozgazdasagi Alapkepzes Szamara (Translated Title: Mathematics for the Bachelor Curricula in Economics, Probability Theory), Budapest: Nemzeti Tankonyvkiado, 2007, pp. 84-92.

[28] Univeristy of Cologne, Methodenpool, : Brainstorming, [Online], [Accessed: 27.07.2012], Available from: http://methodenpool.uni-koeln.de/brainstorming/frameset_brainstorming.html.

[29] S. G. Isaksen and J. P. Gaulin, "A Reexamination of Brainstorming Research: Implications for Research and Practice," *Gifted Chiled Quarterly The Official Journal of the National Association for Gifted Children,* vol. 49., no. 4., 2005..

[30] O. Goldenberg and J. Wiley, "Quality, Conformity, and Conflict: Questioning the Assumptions of Osborn's Brainstorming Technique," *The Journal of Problem*

Solving, vol. 3., no. 2., 2011..

[31] J. Mendel, “Type-2 Fuzzy Sets: Some Questions and Answers,” *IEEE Neural Networks Society,* pp. 10-13., 2003..

[32] J. Saldana, The Coding Manual for Qualitative Researchers, Sage, 2009..

[33] K. K. Aggarwal, Y. Singh, P. Chandra and M. Puri, “Measurement of Software Maintainability Using a Fuzzy Model,” *Journal of Computer Sciences,* pp. pp. 537-541, 2005..

[34] G. Canfora, L. Cerulo and L.Troiano, “Can Fuzzy Mathematics enrich the Assessment of Software Maintainability?,” *ICEISSAM - Software Audit and Metrics,* pp. 85-89., 2004..

[35] H. Mittal and P. Bhatia, “Software Maintainability Assessment Based on Fuzzy Logic Technique,” *ACM SIGSOFT Software Engineering Notes,* vol. Volume 34, no. 3, 2009..

[36] N. Nerurkar, A. Kumar and P. Shrivastava, “Assessment of Reusability in Aspect-Oriented Systems using Fuzzy Logic,” *ACM SIGSOFT Software Engineering Notes,* vol. Volume 35, no. 5, 2010..

[37] Y. Singh, P. K. Bhatia and O. Sangwan, “Software Reusability Assessment Using Soft Computing Techniques,” *ACM SIGSOFT Software Engineering Notes,* vol. Volume 36, no. 1, 2011..

[38] B. W. Boehm, J. R. Brown and M. Lipow, “Quantitative Evaluation of Software Quality,” in *Proceedings of the 2nd International Conference on Software Engineering*, 1976.

[39] J. A. McCall, P. K. Richards and G. F. Walters, “Factors in Software Quality, Concept and Definitions of Software Quality,” [Online], [Accessed: 21.10.2011], Available from: http://handle.dtic.mil/100.2/ADA049014, 1977.

[40] International Organization for Sandardization, “ISO/IEC 9126-1:2001, Software engineering -- Product quality -- Part 1: Quality model,” 2001..

[41] International Organization for Sandardization, “ISO/IEC 25010:2011, Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models,” 2011..

[42] C. Kim and K. Lee, “Software Quality Model for Consumer Electronics Product,” in *Proceedings of the 9th International Conference on Quality Software*, 2008.

[43] R. Dromey, “A Model for Software Product Quality,” in *IEEE Transactions on Software Engineering*, 1995.

[44] International Organization for Standardization, “ISO/IEC 14598:1999, Information technology -- Software product evaluation -- Part 1: General overview,” 1999.

[45] T. L. Saaty and J. M. Katz, “How to Make A Decision: The Analytic Hierarchy Process,” *European Journal of Operational Research,* pp. 9-26, 1990.

[46] International Organization for Standardization, “ISO/IEC TR 9126-2:2003, Software engineering -- Product quality -- Part 2: External metrics,” 2003.

[47] International Organization for Standardization, “ISO/IEC TR 9126-3:2003,,” *Software engineering -- Product quality -- Part 3: Internal metrics,* 2003.

[48] International Organization for Standardization, “ISO/IEC TR 9126-4:2004, Software engineering -- Product quality -- Part 4: Quality in use metrics,” 2004.

[49] B. Kitchenham and S. Pfleeger, “Software Quality: the Elusive Target,” vol. 13, no. 1, pp. 12-21, 1996.

[50] Z. Cohen, “Five ways for Tracing Java Execution, [Online], [Accessed: 11.01.2013], Available from: http://blog.zvikico.com/2007/11/index.html,” 2007.

[51] Mutek Solution, “Bug Trapper, [Online], [Accessed: 22.01.2013], Available from: http://geyra.com/mutek/bugTrapper.htm”.

[52] IBM, “Purify, [Online], [Accessed: 22.01.2013], Available from: http://www-01.ibm.com/software/awdtools/purify/”.

[53] X. Zhang, N. Gupta and R. Gupta, “Pruning Dynamic Slices with Confidence,” in *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2006.

[54] M. Sridharan, S. Fink and R. Bodik, “Thin slicing,” in *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2007.

[55] S. Horwitz, T. Reps and D. Binkley, “Interprocedural Slicing Using Dependence Graphs,” in *Best of PLDI ACM SIGPLAN Notices*, 2004.

[56] T. Ball, M. Naik and S. Rajamani, “From Symptom to Cause: Localizing Errors in Counterexample Traces,” in *Proceedings of the 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2003.

[57] L. Guo, A. Roychoudhury and T. Wang, Accurately Choosing Execution Runs for Software Fault Localization, Lecture Notes in Computer Science, Springer, 2006.

[58] TIOB, “TIOBE Programming Community Index for December 2012, [Online], [Accessed: 03.01.2013.], Available from: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html,” 2012.

[59] K. Normark, Functional Programming in Scheme With Web Programming Examples, [Online], [Accessed: 25.01.2013], Available from: http://people.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigm-overview-section.html, 2010.

[60] M. K. Bergman, “AI3 Adaptive Information, [Online], [Accessed: 03.01.2013], Available from: http://www.mkbergman.com/991/the-state-of-tooling-for-semantic-technologies/,” 2011.

[61] S. Tilkov and S. Vinoski, “The Functional Web: Node.js, Using JavaScript to Build High-Performance Network Programs,” *IEEE Internet Computing Online,* pp. 80-83, 2010.

[62] Mozilla Co., “Mozilla Developer Network, JavaScript, [Online], [Accessed: 03.01.2013], Available from: https://developer.mozilla.org/en-US/docs/JavaScript”.

[63] K. Sipe, “An Introduction to Functional Languages, [Online], [Accessed: 03.01.2013], Available from: http://java.dzone.com/articles/introduction-functional,” 2009.

[64] S. Breu and J. Krinke, “Aspect Mining Using Event Traces,” in *Proceedings of the 19th IEEE International Conference Automated Software Engineering*, 2004.

[65] A. Zaidman, B. Adams, K. D. Schutter, S. Demeyer, G. Hoffman and B. D. Ruyck, “Regaining Lost Knowledge through Dynamic Analysis and Aspect Orientation — An Industrial Experience Report —,” in *Proceedings of the 10th European Conference on Software Maintenance and Reengineering*, 2006.

[66] University of Totonto, Middleware Systems Research Group, “AspectC Compiler Project, [Online], [Accessed: 08.01.2013], Available from: http://www.msrg.utoronto.ca/,” 2010.

[67] Eclipse Foundation, “AspectJ Project, [Online], [Accessed: 08.01.2013], Available from: http://www.eclipse.org/aspectj/”.

[68] O. Spinczyk, G. Blaschke, C. Borchert, D. Lohmann, R. Sand, H. Schiermeier, U. Spinczyk and M. Urban, “AspectC++ Compiler, [Online], [Accessed: 08.01.2013], Available from: http://www.aspectc.org/Home.1.0.html”.

[69] SharpCrafters s.r.o, “Postsharp Plugin, [Online], [Accessed: 08.01.2013], Available from: http://www.sharpcrafters.com/postsharp/features”.

[70] J. Kienzle and R. Guerraoui, AOP: Does it Make Sense? - The Case of Concurrency and Failures, In Object-Oriented Programming Lecture Notes in Computer Science, Springer, 2006.

[71] C. Diggins, “Aspect-Oriented Programming & C++,” *Dr Dobb's Journal,* 2004.

[72] EPFL, “Scala, [Online], [Accessed: 25.01.2012], Available from: http://www.scala-lang.org/”.

[73] Microsoft Co., “F#, [Online], [Accessed: 25.01.2013], Available from: http://research.microsoft.com/en-us/projects/fsharp/”.

[74] OODesign, “Object-oriented Design Patterns, Singleton, [Online], [Accessed: 25.01.2013], Available from: http://www.oodesign.com/singleton-pattern.html”.

[75] D. Simons, “N-Tier Design Pattern,” *MSDN Magazine,* 2009.

[76] Oracle, “EJB Technology, [Online], [Accessed: 25.01.2013], Available from: http://www.oracle.com/technetwork/java/javaee/ejb/index.html”.

[77] W. Crawford and J. Kaplan, J2EE Design Patterns, O'Reilly Media, 2003.

[78] M. Tancreti, M. S. Hossain, S. Bagchi, V. Raghunathan and Aveksha, “Hardware-Software Approach for Non-Intrusive Tracing and Profiling of Wireless Embedded Systems,” in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, New York, 2011.

[79] C. Josephes, “Writing Apache's Logs to MySQL, [Online], [Accessed: 26.04.2012], http://onlamp.com/pub/a/apache/2005/02/10/database_logs.html?page=1,” 2005.

[80] Oracle, “MySQL 5.0 Log Binary Format, [Online], [Accessed: 15.02.2012], Available from: http://dev.mysql.com/doc/refman/5.0/en/binary-log.html”.

[81] Microsoft Co., “Configure Logging Options at the Site Level (IIS 7), [Online], [Accessed: 15.02.2012], Available from: http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/d0846401-e4e6-4d2c-a307-a2fb2f51e0ab.mspx?mfr=true”.

[82] Microsoft Co., “Post-processing and Viewing IIS Request-Based Tracing Data, [Online], 2005, [Accessed: 15.02.2012], Available from: http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/d0846401-e4e6-4d2c-a307-a2fb2f51e0ab.mspx?mfr=true”.

[83] Websense Inc., “Content Gateway, Choosing binary or ASCII, [Online], [Accessed: 15.02.2012], Available from: http://www.websense.com/content/support/library/web/v75/wcg_help/ASCII.aspx” .

[84] Apache Foundation, “Log4j, [Online], [Accessed: 28.01.2013], Available from: http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/net/SocketAppender.html”.

[85] Stackoverflow, “Log to Database instead of Lof Files, [Accessed: 26.04.2012],

http://stackoverflow.com/questions/1160720/log-to-database-instead-of-log-files".

[86] Stackoverflow, "Storage of My Log Files, [Online], 2009, [Accessed: 26.04.2012], http://stackoverflow.com/questions/1037024/storage-of-many-log-files".

[87] Stackoverflow, "Is Writing Server Log Files to Database a Good Idea?, [Online], 2008, [Accessed: 26.04.2012], http://stackoverflow.com/questions/290304/is-writing-server-log-files-to-a-database-a-good-idea".

[88] Stackoverflow, "Using a SQL Server for application logging Pros/Cons, [Online], 2008, [Accessed: 26.04.2012], http://stackoverflow.com/questions/209497/using-a-sql-server-for-application-logging-pros-cons".

[89] J. Hoxley and O. Wilkinson, Using XML Technologies for Enhancing Log Files In: Beginning Game Programming, A GameDev.net Collection, 2009, pp. 441-469.

[90] Stackoverflow, "Best XML format for log events in terms of tool support for data mining and visualization, [Online], [Accessed: 16.02.2012] Availale from: http://stackoverflow.com/questions/465329/best-xml-format-for-log-events-in-terms-of-tool-support-for-data-m," 2009.

[91] U. Erlingsson, M. Peinado, S. Peter and M. Budiu, "Fay: Extensible Distributed Tracing from Kernels to Clusters," in *ACM*, 2011.

[92] O. Dubuisson, ASN.1, Communication between Heterogeneous Networks, 2000.

[93] J. Carvallo and X. Franch, "Extending the ISO/IEC 9126-1 Quality Model with Non-Technical Factors for COTS Components Selection," in *Proceedings of the 2006 International Workshop on Software Quality*, 2006.

[94] T. Galli, F. Chiclana, J. Carter and H. Janicke, "Modelling Execution Tracing Quality by Type-1 Fuzzy Logic," *Acta Polytechnica Hungarica,* vol. 8, no. 10, pp. 49-67, 2013.

[95] L. A. Zadeh, "Fuzzy logic = computing with words," *IEEE Transactions on Fuzzy Systems,* vol. 4., no. 2., pp. 103-111, 1996.

[96] S. Coupland, M. Gongora, R. John and K. Wills, "A Comparative Study of Fuzzy Logic Controllers for Autonomous Robots, [Online], 2006, [Accessed: 25.06.2012], Available from: https://www.dora.dmu.ac.uk/bitstream/handle/2086/184/bj12006.pdf?sequence=3".

[97] R. John and J. Mendel, "Type-2 Fuzzy Sets Made Simple," *IEEE Transactions on Fuzzy Systems,* pp. pp. 117-127., 2002..

[98] R. John and S. Coupland, "Extensions to Type-1 Fuzzy Logic: Type-2 Fuzzy Logic and Uncertainity," in *Computational Intelligence: Principles and Practice*, 2006., pp. 89-102..

[99] J.-S. R. Jang, C.-T. Sun and E. Mizutani, Neuro-Fuzzy and Soft Computing, Prentice Hall, 1997..

[100] O. Castillo and P. Melin, Contributions to Fuzzy and Rough Set Theories and Their Applications, Type-2 Fuzzy Logic: Theory and Applications, Studies in Fuzzyness and Soft Computing, vol. 223., Springer, 2010..

[101] L. Zadeh, "The Concept of a Linguistic Variable and its Application to Approximate Reasoning-II.," *Information Sciences,* pp. 301-357, 1975.

[102] E. Herrera-Viedma, F. Herrera, F. Chiclana and M. Luque, "Some Issues on Consistency of Fuzzy Preference Relations," *European Journal of Operational Research,* vol. 154, no. 1, pp. 98-109, 2004.

[103] F. Herrera, E. Herrera-Viedma, S. Alonso and F. Chiclana, “Computing with Words in Decision Making,” *Foundations, Trends and Prospects Fuzzy Optimization and Decision Making,* vol. 8, pp. 337-364, 2009.

[104] F. Chiclana, E. Herrera-Viedma, S. Alonso and F. Herrera, “Cardinal Consistency of Reciprocal Preference Relations: A Characterization of Multiplicative Transitivity,” 2009.

[105] S. -M. Zhou, J. M. Garibaldi, R. I. John and F. Chiclana, “On Constructing Parsimonious Type-2 Fuzzy Logic Systems Via Influential Rule Selection,” *IEEE Transaction on Fuzzy Systems,* vol. 17(3), pp. 654-667, 2009.

[106] T. Ross, Fuzzy Logic with Engineering Application, Wiley, 2010..

[107] T. Runkler, “Selection of Appropriate Defuzzification Methods Using Applicationspecific Properties,” *IEEE Transactions on Fuzzy Systems,* vol. 5., no. 1., 1997..

[108] J. Jassbi, P. J. A. Serra, R. A. Ribeiro and A. Donati, “A Comparison of Mandani and Sugeno Inference Systems for a Space Fault Detection Application,” *Proceedings of World Automation Congress WAC06,* pp. 1-8., 2006..

[109] R. Kumar, Research Methodology, A Step-by-step Guide for Beginners, Sage, 2011..

[110] M. L. Nelson, “A Survey of Reverse Engineering and Program Comprehension,” *ODU CS 551 - Software Engineering Survey,* 1996..

[111] T. Galli, F. Chiclana, J. Carter and H. Janicke, “Towards Introducing Execution Tracing to Software Product Quality Frameworks,” *[Unpublished],* 2013..

[112] N. H. Malhotra, Marketingkutatas (Translated title: Marketing Research), Akademia Kiado, 2009..

[113] L. Hunyadi and L. Vita, Statisztika II. (Translated Title: Statistics II.), Budapest: Aula, 2008.

[114] T. Saaty, “An Essay on How Judgment and Measurement Are Different in Science and in Decision Making,” *International Journal of the Analytic Hierarchy Process,* vol. 1, no. 1, 2009.

[115] B. Roy, “The Outranking Approach and the Foundations of ELECTRE Methods,” *Theory and Decision,* pp. 49-73, 1991.

[116] M. Morisio, I. Stamelos and A. Tsoukias, “Software Product and Process Assessment Through Profile-Based Evaluation,” *International Journal of Software Engineering & Knowledge Engineering,* pp. 495-512, 2003.

[117] L. Zadeh, “Fuzzy Sets,” *Information and Control,* pp. 338-353, 1965.

[118] K. Tanaka, An Introduction to Fuzzy Logic for Practical Applications, Springer, 1996.

[119] H. Hamrawi, “Type-2 Fuzzy Alpha-cuts, [Online], 2011, [Accessed: 14.02.2013], Available from: https://www.dora.dmu.ac.uk/handle/2086/5137,” De Montfort University, Leicester, 2011.

[120] L. Zadeh, “Is There A Need for Fuzzy Logic?,” Annual Meeting of the North American Fuzzy Information Processing Society, 2008.

[121] S. Chopra, R. Mitra and V. Kumar, “Identification of Rules Using Subtractive Clustering with Application to Fuzzy Controllers,” in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, 2004.

[122] C.-C. Wong and C.-C. Chen, “An SVD-QR-based Approach to Fuzzy Modeling,”

in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, 2000.

[123] M. Nie and W. W. Tan, “Towards an Efficient Type-Reduction Method for Interval Type-2 Fuzzy Logic Systems,” in *Proceedings of IEEE World Congress on Computational Intelligence*, 2008.

[124] MathWorks Inc., Fuzzy Inference Process, MatLab 2012b Fuzzy Logic Toolbox Documentation, [Online], [Accessed: 09.11.2012], Available from: http://www.mathworks.com/help/fuzzy/fuzzy-inference-process.html.

[125] MathWorks Inc., Comparison of Sugeno and Mamdani Systems, MatLab 2012b Fuzzy Logic Toolbox Documentation, [Online], [Accessed: 12.11.2012], Available from: http://www.mathworks.com/help/fuzzy/comparison-of-sugeno-and-mamdani-systems.html.

[126] MathWorks Inc., What Is Sugeno-Type Fuzzy Inference?, MatLab 2012b Fuzzy Logic Toolbox Documentation, [Online], [Accessed: 12.11.2012], Available from:http://www.mathworks.com/help/fuzzy/what-is-sugeno-type-fuzzy-inference.html.

[127] G. J. Klir and B. Youan, Fuzzy Sets and Fuzzy Logic, Theory and Applications, Prentice Hall, 1995.

[128] N. N. Karnik and J. M. Mendel, “Centroid of A Type-2 Fuzzy Set,” *Information Sciences ,* pp. 195-220, 2001.

[129] S. Greenfield, F. Chiclana, S. Coupland and R. I. John, “The Collapsing Method of Defuzzification for Discretised Interval Type-2 Fuzzy Sets,” *Information Sciences,* pp. 2055-2069, 2009.

[130] M. Nie and W. W. Tan, “Towards an Efficient Type-Reduction Method for Interval Type-2 Fuzzy Logic Systems,” in *Proceedings of IEEE World Congress on Computational Intelligence*, 2008.

[131] S. Greenfield, F. Chiclana and S. C. R. John, “The Sampling Method of Defuzzification for Type-2 Fuzzy Sets: Experimental Evaluation,” *Information Sciences,* pp. 77-92, 2012.

[132] S. C. Coupland and R. I. John, “Geometric Type-1 and Type-2 Fuzzy Logic Systems,” in *IEEE Transactions on Fuzzy Systems*, 2007.

[133] J. M. Mendel and D. Z. F. Liu, “Alpha-Plane Representation for Type-2 Fuzzy Sets: Theory and Applications,” in *IEEE Transactions on Fuzzy Systems*, 2009.

[134] D. Wu and J. Mendel, “Enhanced Karnik-Mendel Algorithms for Interval Type-2 Fuzzy Sets and Systems,” in *Annual Meeting of the North American Fuzzy Information Processing Society*, 2007.

[135] S. Greenfield, F. Chiclana, S. Coupland and J. Robert, “Type-2 Defuzzification: Two Contrasting Approaches,” in *6th IEEE World Congress on Computational Intelligence*, 2010.

[136] R. John, “Perception Modelling Using Type-2 Fuzzy Sets,” De Montfort University, Leicester, 2000.

[137] J.-S. R. Jang, “ANFIS: Adaptive-Network-Based Fuzzy Inference System,” in *IEEE Transactions on Systems, Man and Cybernetics*, 1993.

[138] O. P. Sahu and S. Kumar, “A New Channel Equalizer Using Adaptive Neuro Fuzzy Inference System,” *IETE Journal of Research,* pp. 201-206, 2011.

[139] A. Aldobhani, “Maximum Power Point Tracking of PV System Using ANFIS

Prediction and Fuzzy Logic Tracking," De Montfort University, Leicester, 2008.

[140] S. Holmes, Introductory Statistics, [Online], [Accessed: 21.02.2013], Available from: http://www-stat.stanford.edu/~susan/courses/s60/split/node106.html, Stanford University, 2000.