Modelling Execution Tracing Quality by Means of Type-1 Fuzzy Logic

Tamás Galli, Francisco Chiclana, Jenny Carter, Helge Janicke

Centre for Computational Intelligence, Faculty of Technology, De Montfort University, The Gateway, Leicester, LE1 9BH, United Kingdom

p10553741@myemail.dmu.ac.uk, chiclana@dmu.ac.uk, jennyc@dmu.ac.uk, heljanic@dmu.ac.uk

Abstract: Execution tracing quality is a crucial characteristic which contributes to the overall software product quality though the present quality frameworks neglect this property. In the scope of this pilot study the authors introduce a process to create a model for describing execution tracing as a quality property; moreover, the performance of four different models created is compared. The process and the models presented are capable of capturing subjective uncertainty which is an intrinsic part of the quality measurement process. In addition, the possibility of linking the presented models to software product quality frameworks is also illustrated.

Keywords: software product quality models; execution tracing quality; fuzzy logic; uncertainty

1 Introduction

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 to check the state of software systems. In the scope of this publication we also use the two phrases 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 belongs to dynamic analysis techniques i.e. testing, and 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. Static and dynamic analysis techniques possess two significant common attributes: (1) they are applied to achieve the same goal to diagnose errors; (2) they generalize from a subset of all possible executions. Each technique has its own particular advantage.

Static analysis can produce sound results however with general properties, which are not precise but these results are accurate and have validity over all possible inputs. Dynamic analysis examines the concrete execution of the program by observing its behaviour, which is precise but the results are not valid for all possible inputs. The literature promotes the synergic use of these techniques [37], [6].

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 [2].

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 [2], [35]. In the case of distributed, multithreaded applications execution tracing is the only adequate instrument to help with the error analysis as states Laddad in [20]. 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 to what the application is doing [34].

Moreover, execution tracing significantly influences program comprehension, the importance of which arises if the program documentation is deficient or of poor quality. In a study by Fjeldstad and Hamlen [7] it is estimated that the comprehension of existing software systems consumes between 47% and 62% of maintenance resources [25], [31]. An experiment conducted by Karahasanovic and Thomas introduced in [19] categorized 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 which can greatly be enhanced by execution tracing, offering a basis for trace visualization and program comprehension [31].

Tracing, logging or constraint checks represent a significant part of the source code of applications. Spinczyk, Lehmann and Urban in [33] 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 as execution tracing which in itself is an important quality factor.

In conclusion, the above indicate that execution tracing has significant impact on the analysability of software systems. Moreover, measuring quality is difficult, some properties are easier to measure than others even if they are well defined [27]. Quality frameworks include the description of qualitative properties in quantitative manner and quality measure elements which cannot be measured directly but only derived. Consequently, the measurement process implicates subjective uncertainty which has also been admitted by the standard ISO/IEC 25021:2007 involved in software product quality by defining the subjective measurement method. In the scope of this article the authors introduce a pilot study to describe execution tracing quality by means of a model which can 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.

The remainder of this article is structured as follows: Section 2 describes how the quality model pilot was built including identification of inputs, outputs and construction of the knowledge base. Section 3 introduces the validation of the quality model. Section 4 describes the limitations of the pilot study and gives an outlook to the final model while Section 5 introduces related works. We summarise the contributions of our work in Section "Conclusions" and outline the future work in this area.

2 Constructing the Model

The model reflects the results of an empirical research which 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 results from a brainstorming session and further processing of the output of this session. Brainstorming served as a method of data collection, developed by A. Osborn and made more sophisticated by H. C. Clark as a technique to create, collect, express ideas to a topic [34]. 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 a critic at first, then (2) the ideas can be developed further by other group members. Consequently, synergistic effects can lead to the triggering of ideas by those already present [33]. Before and after the idea generation phase an ideation phase must take place. Before ideation the participants think over the brainstorming question individually as preparation for the brainstorming [13]. The idea generation is followed by an ideation phase again where evaluation of the collected ideas takes place [34]. The critics towards this method mainly focus on the idea generation phase regardless of ideation that takes place before and after; however, Osborn did not propose brainstorming instead of the ideation but as a supplement to it [13]. In this method the quantity of ideas is not limited. The more ideas that are collected the more probable it is to have qualitative ideas among them. The latter has been questioned in [8], which contradicts the views held in [13] in some respects.

The output of the brainstorming is a list of raw ideas considered to be feasible by the group [34]. This list forms the possible input candidates of the model, which need to undergo further analysis.

The quantitative part of the research formalizes the relationships of the inputs in the output. For collecting this information, experiences of one software developer involved also in software maintenance for several years were scrutinized. The quantitative part of the research needs to use methods to deal with subjective uncertainty. Consequently, fuzzy logic is used to describe the input-output relationships, which also offers tolerance towards imprecision [37].

Fuzzy logic offers basically two 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 larger amounts of uncertainty need to be considered. In contrast, type-2 fuzzy logic performs well in both situations but the operations and inference are more complex and computationally more expensive than the operations and inference of type-1 fuzzy logic. In the pilot study described in this paper type-1 fuzzy logic is used [23], [5], [17], [18].

Fuzzy modelling makes it possible to incorporate human expertise in the model directly [14], [4]. Castillo and Melin recommend the following modelling steps [4]:

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 to be able to compare the performance of the different models with the same membership functions.

2.1 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, the possible inputs, i.e. quality properties on which execution tracing quality depends, were identified by brainstorming.

The brainstorming group was constructed of software developers and 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 [22].

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, developers or 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.

2.2 Linguistic Variables

The notion of linguistic variables was introduced by L. Zadeh [38]. These variables are able to handle imprecision and offer a basis also for natural language computation. The formalism implemented by these variables and the if-then rules establishes an effective modelling language [37].

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 but it also introduces complexity as the number of necessary fuzzy rules needs to be increased. Moreover, incorporating human expertise with a high number of linguistic variables exposes difficulties because contradictions can be introduced in the model in an easy manner. 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*}

2.3 Membership Functions

Linguistic variables were depicted 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 1.

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.

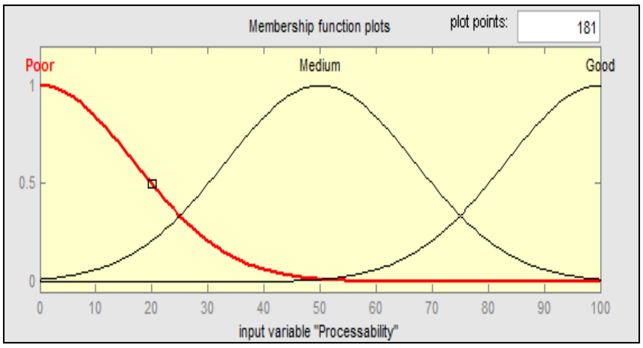

Figure 1
Membership Functions of the Input: Processability

2.4 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 [37]. The knowledge base is summarized in Table 1. This is not a complete rule set i.e. it does not contain each variation of all linguistic variables of all inputs but a complete rule set is not necessary to achieve appropriate performance. The model was assessed as described in Section 3.

Table 1

Antecedent and Consequent Parts of the Fuzzy Rules

	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*

2.5 Type-1 Fuzzy Inference Techniques

The two most widespread fuzzy methods for inference have been considered: (1) Mamdani’s approach and (2) the approach of Takagi-Sugeno-Kang. The Tsukamoto method [28], [14] has been excluded as it requires monotonic consequent membership functions.

2.6 Comparison of the Created Models

For the purpose of comparison, four models were created with the same inputs and output: (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 only for the best performing method which in this context was implemented by the inference mechanism of Takagi-Sugeno-Kang with Gaussian membership functions. 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 than 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.

2.6.1 Mamdani's Approach

Inference was performed with the min-max method [28]. 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.

The defuzzification methods applied indicated considerable deviations when the inputs reached the limits 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 [29].

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.

2.6.2 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 computationally expensive 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.

The model built with the approach of Takagi-Sugeno-Kang with Gaussian membership functions provided the best performance compared to the other models on the 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 [14]. Further investigation of Jassbi et al. confirms that Takagi-Sugeno-Kang method shows more tolerance towards input noise than Mamdani's method [15], which is an advantageous property in the problem domain of the current research.

3 Validation

As the best results were produced by the Takagi-Sugeno-Kang approach with Gaussian membership functions, the validation of this model is presented in this section. 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 [20] 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.

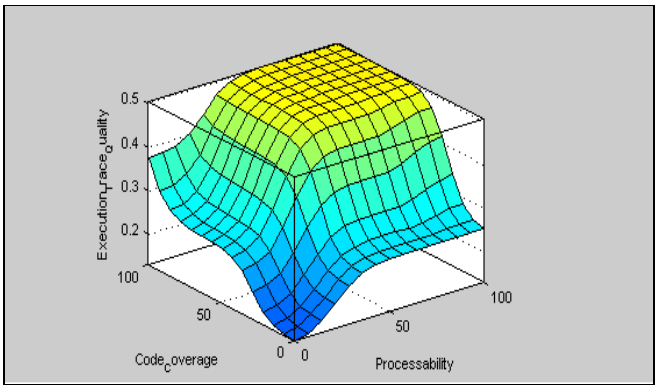

Figure 2
Code Coverage and Processability vs. Execution Trace Quality

Figure 2 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 also reflects 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.

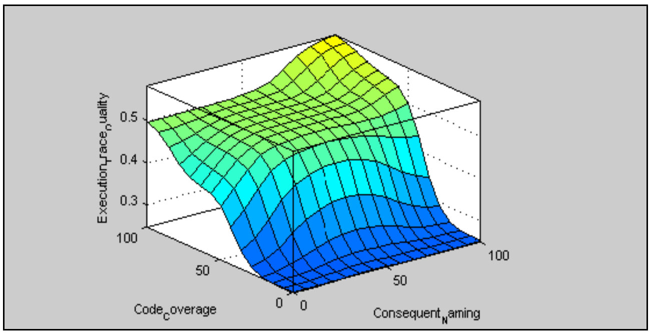

Figure 3
Code Coverage and Consequent Naming vs. Execution Trace Quality

Figure 3 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.

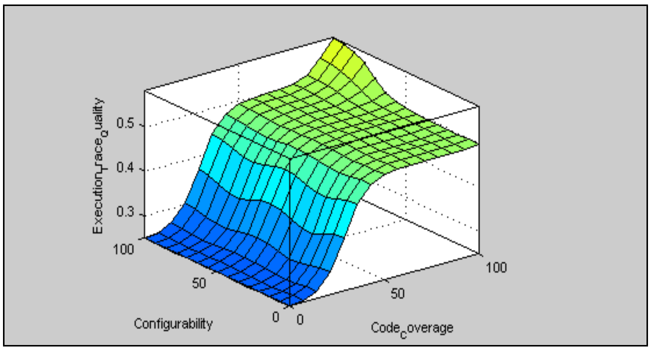

Figure 4

Configurability and Code Coverage vs. Execution Trace Quality

Figure 4 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.

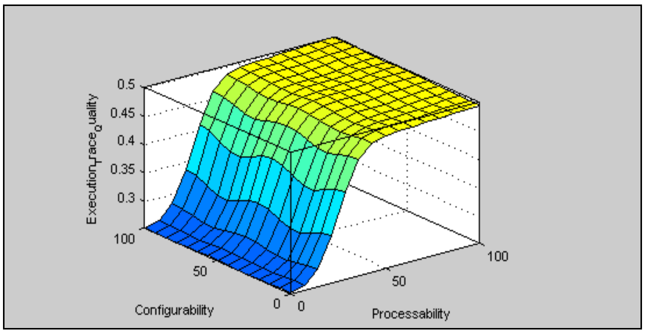

Figure 5

Configurability and Processability vs. Execution Trace Quality

Figure 5 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, *"Configurability"* has little more than zero influence on the output in comparison to *"Processability"*, which has to be reflected by the model.

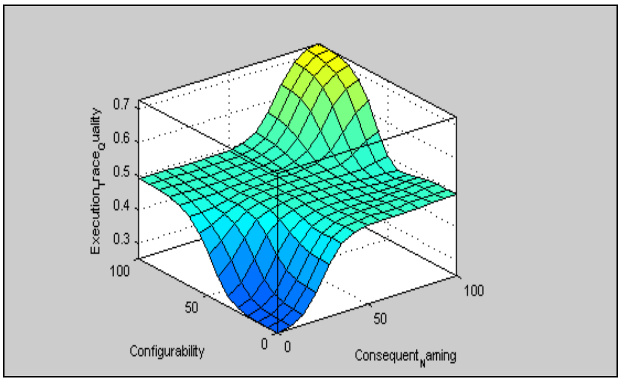

Figure 6

Configurability and Consequent Naming vs. Execution Trace Quality

Figure 6 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 most 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.

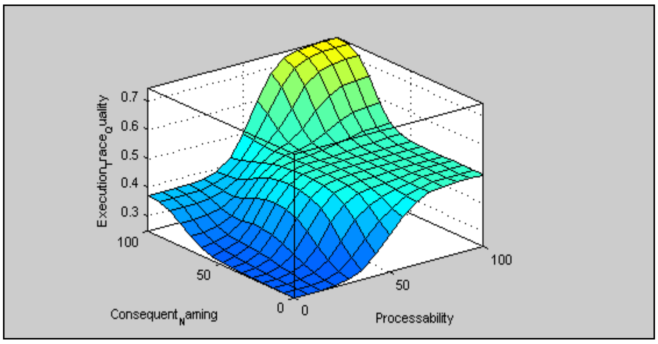

Figure 7

Consequent Naming and Processability vs. Execution Trace Quality

Figure 7 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, depicted on the chart.

Table 2
Summary of the Validation Charts

Summary of the validation charts		
ID	Diagram	Conclusion
1.	From Figure 2. to Figure 7.	Changes of the inputs produce appropriate responses in the output.
2.	Figure 2.	The inputs Code Coverage and Processability have a significant impact on Execution Trace Quality.
3.	Figure 3.	Code Coverage influences Execution Trace Quality to a bigger extent than Consequent Naming.
4.	Figure 4.	Code Coverage influences Execution Trace Quality to a bigger extent than Configurability.
5.	Figure 5.	Processability influences Execution Trace Quality to a bigger extent than Configurability.
6.	Figure 6.	The inputs Consequent Naming and Configurability have approximately the same impact to Execution Trace Quality.
7.	Figure 7.	Processability has a bigger impact on Execution Trace Quality than Consequent Naming.
8.	Figure 7.	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.

4 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 [1], [3], [24] or reusability [26], [32]. 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 [10]. In addition, the maintainability models investigated do not handle execution tracing quality.

Canfora, Cerulo, Troiano in [3] 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 which 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 et al. discussed in [3] 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 [26] proposed a model based on type-1 fuzzy logic for reusability of aspect-oriented systems. Singh, Bhatia, Sangwan in [32] 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.

5 Limitations of the Pilot Study and Outlook to the Final Model

We need to make a distinction between the research methods applied for the pilot model described by this paper and the 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.

The reliability of the model strongly depends on the reliability of the data collected. The data of the pilot originate from the output of one brainstorming session processed by two experts in the field; meanwhile, the knowledge base formalises the knowledge of one expert. In contrast, the data of the final model will be based on a well-defined study population: software developers and maintainers will be selected from companies which have at least 50 employees in Hungary. The study population is distributed among 37[1] companies and its size amounts to 6010[2] individuals. Participants of the brainstorming sessions will be selected from this study population with judgmental sampling [22] for the qualitative research. Several brainstorming sessions will take place until a saturation point is reached or appropriately approached [20]. To implement this, two coders will look for synonyms in the outputs of the brainstorming sessions.

[1] Online database of HBI Online, [Online], 2012, [Accessed: 23.05.2012], Available from: www.hbi.hu, Search criteria: TEAOR'08=6201 and number of employees greater or equal 50

[2] Source: Hungarian Central Statistical Office, Social Statistics, Labour Market, 2012, [Online], [Accessed: 14.09.2012], Available from: http://statinfo.ksh.hu/Statinfo/themeSelector.jsp?&lang=en

Moreover; the data collected will undergo first and second cycle coding to establish the quality properties [30]. Coding also assumes calculating intercoder reliability for the coding process between the coders.

Regarding the quantitative stage, the same study population will be sampled with random multistage sampling to ensure a $p<.05$ statistical significance [22], [9]. The knowledge base, i.e. the rule set, of the model will be constructed from the knowledge gained from the sample by on-line surveying.

Conclusions

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 measurements 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. Different algorithms for parameter tuning will also be considered [16].

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 while using overlapping Gaussian membership functions. In addition, Mamdani’s inference method with the COG or MOM defuzzification techniques could not be applied as 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 methods 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. According to the plans its validity will be based on statistical evidence beside face and content validity [20]. Furthermore, the final model is planned to be constructed by using the adaptive neuro-fuzzy approach (ANFIS) which helps to keep the internal consistency by creating the model on the randomly selected one half of the data and checking it on the other half [14]. Application of ANFIS is also necessary due to automatic processing lager amount of data planned to be collected during the quantitative research. Reliability will also be embedded in the whole process of the research reaching from intercoder reliability to the reliability of the sampling and statistical inference. Moreover, in the qualitative part credibility, transferability, dependability and confirmability will also be considered [20]. Research methods for the final model are presented in more detail in the previous section.

The present model is a standalone model but it also offers the possibility to be linked to the analysability sub-characteristic of the characteristic maintainability of ISO/IEC 9126-1 [10] or ISO/IEC 25010 software product quality models [10]. Linking the developed model to the standards is possible after formal description of the inputs, required by ISO/IEC 25021 [12], and after applying decomposition according to the internal-external view of the software product quality expressed by the ISO/IEC software product quality models.

Acknowledgement

The authors would like to acknowledge gratefully to all colleagues who participated in the brainstorming session and in the qualification of the input candidates of the model or contributed in any other way to the research.

References

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

[2] Buch, I. Park and R. “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

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

[4] Castillo, O., 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

[5] Coupland, S., M. Gongora, R. John, and K. Wills. “A Comparative Study of Fuzzy Logic Controllers for Autonomous Robots.” *Proceedings of IPMU 2006 Conference, [Online], [Accessed: 07.12.2011], Available from: https://www.dora.dmu.ac.uk/handle/2086/184*, 2006

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

[7] Fjeldstad, R. K. and W. T. Hamlen, “Application Program Maintenance Study: Report to Our Respondents,” Proceedings GUIDE 48, Philadelphia, PA, 1983

[8] Goldenberg, O., and J. Wiley. “Quality, Conformity, and Conflict: Questioning the Assumptions of Osborn’s Brainstorming Technique.” *The Journal of Problem Solving* 3, No. 2 (2011)

[9] Hunyadi, L., and L Vita. *Statisztika II. (Translated Title: Statistics II.).* Aula Kiado, 2008

[10] 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

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

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

[13] Isaksen, S. G., 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* 49, No. 4 (2005)

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

[15] Jassbi, J., 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* (doi: 10.1109/WAC.2006.376033), 2006: 1-8

[16] Johanyák, Zsolt Csaba, Olga Papp. "A Hybrid Algorithm for Parameter Tuning in Fuzzy Model Identification." Acta Polytechnica Hungarica, Vol. 9, No. 6, pp. 153-166, 2012

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

[18] John, R., and S. Coupland. "Extensions to Type-1 Fuzzy Logic: Type-2 Fuzzy Logic and Uncertainity." In *Computational Intelligence: Principles and Practice*, by eds. G. Y. Yen ET AL, 89-102, 2006

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

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

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

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

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

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

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

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

[27] Research Triangle Institute. “RTI Project Number 7007.011, The Economic Impacts of Inadequate Infrastructure for Software Testing.” *[Online], [Accessed: 06.12.2012], Avaliable from: http://www.nist.gov/director/planning/upload/report02-3.pdf* (U.S Department of Commerce), 2002

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

[29] Runkler, T. “Selection of Appropriate Defuzzification Methods Using Applicationspecific Properties.” *IEEE Transactions on Fuzzy Systems* 5, No. 1 (1997)

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

[31] Shi, Z. “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

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

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

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

[35] V. Uzelac, A. Milenkovic, M. Burtscher, 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

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

[37] Zadeh, L. A. “Fuzzy logic = computing with words.” *IEEE Transactions on Fuzzy Systems* 4, No. 2 (1996): 103-111

[38] Zadeh, L. A. “The Concept of a Linguistic Variable and its Application to Approximate Reasoning-II.” *Information Sciences*, 1975: 301-357

[39] Zadeh, L.A. “Fuzzy Sets.” *Information and Control*, 1965: pp. 338-355