Expert Systems With Applications 86 (2017) 350–366

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

QATCH - An adaptive framework for software product quality assessment[☆]

Miltiadis G. Siavvas, Kyriakos C. Chatzidimitriou[*], Andreas L. Symeonidis

Electrical and Computer Engineering Department, Aristotle University of Thessaloniki, GR54124 Thessaloniki, Greece

ARTICLE INFO

Article history:
Received 1 February 2017
Revised 24 May 2017
Accepted 25 May 2017
Available online 25 May 2017

Keywords:
Software quality assessment
Software engineering
Multi-criteria decision making
Fuzzy analytic hierarchy process
Software static analysis
Quality metrics

ABSTRACT

The subjectivity that underlies the notion of quality does not allow the design and development of a universally accepted mechanism for software quality assessment. This is why contemporary research is now focused on seeking mechanisms able to produce software quality models that can be easily adjusted to custom user needs. In this context, we introduce QATCH, an integrated framework that applies static analysis to benchmark repositories in order to generate software quality models tailored to stakeholder specifications. Fuzzy multi-criteria decision-making is employed in order to model the uncertainty imposed by experts' judgments. These judgments can be expressed into linguistic values, which makes the process more intuitive. Furthermore, a robust software quality model, the base model, is generated by the system, which is used in the experiments for QATCH system verification. The paper provides an extensive analysis of QATCH and thoroughly discusses its validity and added value in the field of software quality through a number of individual experiments.

1. Introduction

Our era is characterized by major technological advancements and constant digitization of information. Software is practically everywhere; web, mobile, desktop, embedded, or distributed software is developed in order to assist people in achieving their goals easier, faster and more efficiently. This inevitably raises the issue of software quality as a major concern for both end users that want to do their job as best as possible, as well as software development companies that aspire to offer their customers high quality services, and maintain (or optimally increase) their market pool.

A great number of software quality models have been proposed the last decades, however none of them has managed to become widely accepted and applied across application domains. The main reason that universal software quality models fail is the subjectivity that underlies the notion of quality. Furthermore, the application domain plays an important role on the definition of software quality. For instance, an enterprise that produces online services may pay more attention on the reliability and the security of its products, while an enterprise that develops software for embedded systems may care chiefly for performance efficiency. As a result, the different stakeholders define quality from different perspectives.

For this reason, a lot of research has been carried out in recent years targeted to develop a mechanism that will allow the derivation of custom quality models. The main question that these research efforts try to answer may be summarized as follows:

Is it possible to develop a system that may allow the derivation of quality models totally adjusted to custom needs and may be able to directly assess the quality of software products?

In this context, we introduce QATCH[1], a complete tool chain that allows (i) the generation of software quality models that reflect stakeholder specifications by employing static analysis to a desired benchmark repository, and (ii) the quality evaluation of software products by using previously generated models. The pivotal characteristics of the proposed framework are:

- The adoption of benchmarking and static analysis for thresholds derivation, thus automating and adding objectivity to the quality assessment process.
- The simplicity of the derived quality models, which increases confidence to the QATCH framework.
- The adoption of an enhanced fuzzy multi-criteria decision-making technique for weights elicitation, in order to model the uncertainty imposed by human judgments.

[☆] QATCH home page: http://issel.ee.auth.gr/qatch.

[*] Corresponding author.

E-mail addresses: siavvasm@ece.auth.gr (M.G. Siavvas), kyrcha@issel.ee.auth.gr, kyrcha@gmail.com (K.C. Chatzidimitriou), asymeon@eng.auth.gr (A.L. Symeonidis).

[1] QATCH, or better Q.A.T.CH., stands for Quality Assessment Tool Chain.

http://dx.doi.org/10.1016/j.eswa.2017.05.060
0957-4174/© 2017 Elsevier Ltd. All rights reserved.

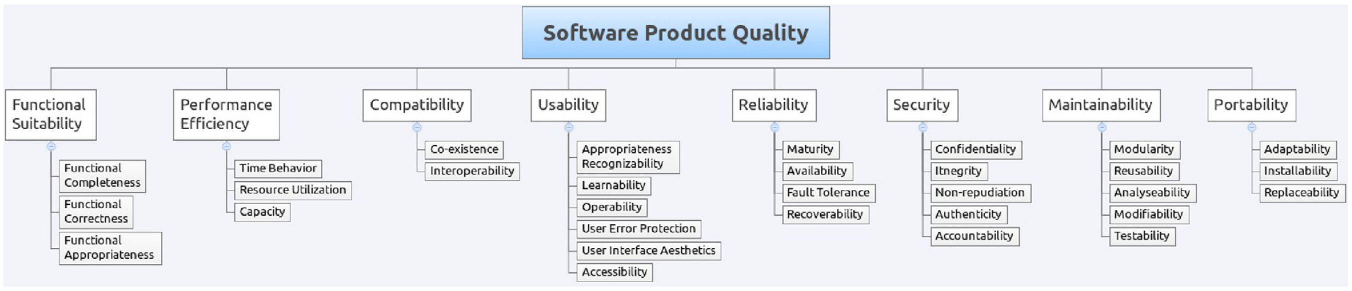


Fig. 1. The ISO 25010 software quality standard from a product standpoint.

- The ability of the experts to express their judgments in the form of linguistic values, thus making the QATCH process more intuitive.

The rest of the paper is structured as follows: Section 2 discusses the related work in the field of software quality by presenting various state-of-the-art frameworks, tools and models. Section 3 provides a description of the methodologies that QATCH follows for the derivation of quality models and the evaluation of software products. The main structure of the derived models is presented as well. Section 4 gives a high-level overview of QATCH, along with a brief description of the implemented modules, while Section 5 examines both the validity of the system and its added value in the field of software quality through a set of experiments. Finally, Section 6 summarizes the work conducted, discusses ideas for future improvements and concludes the paper.

2. Related work

2.1. Quality assessment models

Quality assessment models, as their name indicates, assess the quality of the software under investigation and are based on quality definition models (i.e. ISO 25010 (ISO/IEC, 2011), the most prominent definition model and the one we will employ). Such models strive to gather metrics related to software quality, map these metrics to quality characteristics as these are defined by a quality definition model and, through normalization and aggregation, assess the overall software quality. The first quality models were published by the teams of Boehm et al. (1976) and McCall, Richards, and Walters (1977). These models used hierarchical decomposition in order to decompose quality into simpler characteristics and sub-characteristics that were easier to manage. These models were the predecessors of ISO 9126 (ISO/IEC, 2001), which in turn was the predecessor of ISO 25010. The latest set of quality characteristics proposed by ISO 25010 are: Functional suitability, Performance efficiency, Compatibility, Usability, Reliability, Security, Maintainability, and Portability, which are further split into 31 sub-characteristics (Fig. 1).

The main problem with ISO 25010 (ISO/IEC, 2011) is that, even though it has a nice hierarchy of characteristics and sub-characteristics, it does not define how they are assessed and measured. Software quality measurement is needed, quantifying to what extent a software application or system rates along each of these eight dimensions. That is why many projects, teams and authors have used the ISO 9126 and the ISO 25010 as a theoretical model and then resolved to their own interpretation of how each characteristic is measured. Aggregated measures of software quality are computed ad-hoc through qualitative, quantitative or mixed scoring schemes, and then a weighting system reflects their priorities.

Preliminary approaches include work by Dromey (1995), that hierarchically decomposes software into its constituents (programs, modules, objects, variables etc.) and gives quality properties to each one of them. They also include the maintainability index (Coleman, 1992) that quantifies the characteristic of maintainability through the following equation:

$$MI = 171 - 5.2\ln(aveVol) - 0.23aveV(g) - 16.2\ln(aveLOC) + 50\sin(\sqrt{2.46perCM}) \quad (1)$$

where *aveVol*, *aveV*(*g*), *aveLOC* and *perCM* are the average Halstead's volume metric, the extended cyclomatic complexity, the average lines of code and the percentage of comments of a software product respectively. On top of that, the QMOOD approach (Bansiya & Davis, 2002) provides a quality model for object-oriented systems based on Dromeys model.

Other approaches may use software complexity metrics, i.e. lines of code with and without comments, number of characters, number of comments, McCabe's cyclomatic complexity (McCabe, 1976), Belady's bandwidth metric etc., to derive models that predict the number (Chiu, 2009) or the existence of faults (Chiu, 2011), or use metrics like those in Bansiya and Davis (2002) and in Chidamber and Kemerer (1994) for mining generalized thresholds that predict the fault-proneness of a software product (Arar & Ayan, 2016). Such models, in the view of the ISO standard, can be thought of as estimators of a sub-characteristic of the reliability component and thus cannot be used to classify or rank the software product quality holistically.

Contemporary work in software quality models includes work presented by SIG (Heitlager, Kuipers, & Visser, 2007), (Baggen, Correia, Schill, & Visser, 2012), which is specifically focused on maintainability. Instead of using the ISO characteristics that are not easily observable and measurable, SIG employs its own -tangible-quality properties to assess maintainability, i.e. Volume, Duplication, Unit Size, Unit Complexity, Unit Interfacing, Module Coupling, Component Balance and Component Independence. It is worth noticing that the SIG model incorporates a benchmark repository in order to calibrate the system, which practically contains their quality system knowledge. For assessing software maintainability, the model takes the software system as input, calculates its metrics and performs two levels of aggregation: at the first level, all quality properties are transferred to the system level by calculating their quality profiles, while at the second level SIG applies the quality profiles in order to extract cumulative quality profiles for each property. At every level, the profiles are compared with thresholds calculated from the benchmark repository and each quality property is given a quality score both in continuous and discrete (5-star) form. The individual quality scores of the properties are aggregated to first calculate the quality scores of the model's sub-characteristics, and in turn to produce the overall quality level of the product under evaluation.

Although of particular interest, the main disadvantage of software quality models is that, since there is no clear definition of which metrics should be mapped into which characteristics, they require quality experts to map the metrics to quality characteristics, assess their contribution and decide how to aggregate all the characteristics and sub-characteristics to produce a single value of the software quality. Therefore, the model generation process inevitably includes subjectivity and is not fully-automated as it is based on experts' judgments.

2.2. Software quality tools and frameworks

The SQALE[2] (Software Quality Assessment based on Lifecycle Expectations) method encompasses two models: a ranked uniform quality model and an analysis model based on remediation costs (Letouzey & Coq, 2010). The former model defines the quality characteristics and sub-characteristics, taking into account the lifecycle of source code (coding, testing, evolving, delivering, maintaining and reusing). It is structured in ranked layers - bottom up: testability, reliability, changeability, efficiency, maintainability and reusability - that are taken from previous models of Boehm et al. (1976), McCall, Richards, and Walters (1977) and ISO 9126 (ISO/IEC, 2001) and directly map to software lifecycle steps. The SQALE model constitutes a requirement document. To detect a non-conformance, both the developers and the users point of view is taken into account. If a non-conformance is detected this implies the absence of quality and induces technical debt (i.e. the effort required for the resolution of all the issues that a software product comprises) (Sterling, 2010). The latter model (analysis model) defines the rules to characterize or rate products based on the measurements obtained on the control points.

The SQUALE[3] project (Mordal-Manet et al., 2009) defines a qualimetry platform that allows to analyze multi-language software applications in order to give a sharp and comprehensive perception of their quality. It is inspired by existing standards (e.g. ISO 9126) and approaches (e.g. McCall, Richards, & Walters (1977)), and it is open-source. SQUALE aspires to help developers assess software and improve quality with respect to high level Architecture, Conformity, Evolutionary, Maintainability, Reliability and Reuse capacity. In order to do so, SQUALE aggregates raw information (metrics) from third party tools (commercial or open-source) into its quality models.

The Quamoco[4] project (Wagner et al., 2012) (Quality Modelling and Assessment Approach), aims at bridging the gap between abstract quality attributes of software quality models and concrete quality assessments. For this reason, the Quamoco team has developed a meta-quality model specifying general concepts, a quality base model covering the most important quality factors, and a quality assessment approach. Again, the base model uses the ISO 25010 quality attributes, which comprises 200 factors and 600 measures for Java and C#. Especially for the maintainability aspect, empirical analysis has shown that it has high correlation against human expert rating. Also, its modular approach gives the user the ability to match the needs of each respective software product.

Apart from small-medium scale projects, several tools have been developed for measuring and subsequently improving the quality of source code components. CKJM[5] and the Metrics Eclipse plugin[6], focus on computing static analysis metrics for source code, provide metrics values along with recommended thresholds. More complex tools employ static analysis to measure certain aspects of the source code. PMD[7] is a popular source code analyzer that detects bad practices in source code, including e.g. naming conventions, unused variables, duplicate code fragments etc. The rulesets used to define these violations are written in XML, and the user can add custom rules or modify existing ones. FindBugs[8] (Hovemeyer & Pugh, 2004) is another tool that also checks for bad coding practices, however its main focus is on bug detection using static analysis. The tool performs syntactic checks and dataflow analysis in Java bytecode in order to detect defect-prone source code. Maintaining a consistent code style throughout a project and complying with certain coding standards is also important, a function performed by Checkstyle.[9] The latest versions of the tool additionally support detecting bad design decisions.

Several frameworks combine information from multiple tools and present it to the user in a unified manner. Coverity Code Advisor[10] incorporates both its in-house quality advisor and the FindBugs tool to provide an in-depth analysis of code issues. Although it is a proprietary commercial solution, Coverity is quite promising, since it offers an intuitive web interface where issues can be assigned to developers, marked according to their priority, etc.

Finally, the SonarQube[11] platform is an open source platform for continuous inspection of code quality. It supports languages like Java, C/C++, Objective-C, C#, PHP, Flex, Groovy, JavaScript, Python, PL/SQL, COBOL, etc. and it offers reports on duplicated code, coding standards, unit tests, code coverage, code complexity, potential bugs, comments and design and architecture. It aggregates the results produced by static analysis tools to provide an estimation of software product quality expressed in terms of technical debt. For technical debt (Sterling, 2010) evaluation the SQALE methodology is used. SonarQube allows integration with several tools, including PMD, FindBugs (Hovemeyer & Pugh, 2004), and Checkstyle. The platform offers a dashboard in the form of a web interface and allows several configurations, which refer to all the available tools-plugins that are included. A similar platform is the SQUORE platform (Baldassari, 2013).

2.3. Contributions

From the above discussion on the state-of-practice, it becomes eminent that the existing projects, tools and frameworks are quite capable of providing measurable information for the quality characteristics of a software project. However, their offerings are often overwhelming to the developers; metrics values, either low-level (e.g. McCabe complexity (McCabe, 1976)) or high-level (e.g. Technical Debt (Sterling, 2010)), are not always comprehensive, especially if they are not accompanied by quality thresholds. Defining these thresholds is a non-trivial task, thus most organizations need to resort to the help of quality experts. Moreover, even when expert help is available, it is highly improbable that the designed configurations are flexible enough to effectively describe diverse classes of software components. Only few approaches have focused on training quality models that can provide such adaptable quality estimations, however such systems are still limited by the definition of certain quality thresholds, imposed by ground truth (possibly expert-defined) values used for training, while the resulting models are usually too complex.

This lack of the transparency of the existing quality approaches, along with the absence of sophisticated techniques for objective weights elicitation, are the major drawbacks that the current research tries to tackle. We introduce QATCH, an integrated framework that assists the user throughout the whole quality assessment process, from model derivation to quality evaluation. QATCH is a holistic approach that combines the pivotal characteristics of

[2] http://www.sqale.org/.
[3] http://www.squale.org.
[4] http://www.quamoco.de/.
[5] http://www.spinellis.gr/sw/ckjm/.
[6] http://metrics.sourceforge.net/.
[7] https://pmd.github.io/.

[8] http://findbugs.sourceforge.net/.
[9] http://checkstyle.sourceforge.net/.
[10] http://www.coverity.com/products/code-advisor/.
[11] https://www.sonarqube.org/.

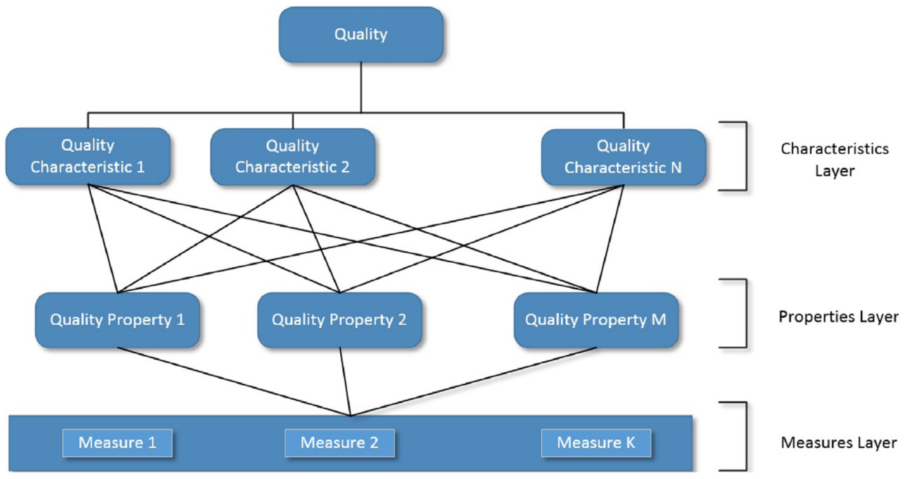


Fig. 2. The general structure of a quality model derived by the proposed framework.

the existing efforts in order to allow the derivation of simple-three-layer hierarchical models. The produced models employ a weighted average aggregation scheme to calculate the overall quality score of the product under evaluation, the weights of which are defined based on experts judgments.

The following subsections discuss the advances of QATCH with respect to the drawbacks identified in the related state-of-the-art research.

2.3.1. *Transparency*

The internal structure of the existing quality models is not always clear, due to the fact that they are either proprietary, and therefore closed source, or they are open source, but they have become obsolete with lack of support (e.g. for deriving aggregation weights). In addition, many of the quality models proposed in the literature utilize Machine Learning (ML) techniques in order to calculate the quality scores of the software products under assessment (Al-Jamimi & Ahmed, 2013). However, these models do not provide any insight on how a specific quality score is calculated, and therefore they are viewed as black box techniques, not allowing experts to incorporate their knowledge and comment on the validity of the assessment approach under question.

Opposed to related work, QATCH constitutes an open source[12] and fully transparent quality modeling and assessment mechanism, which is freely available for use or further testing and experimentation. The quality models produced by QATCH are mathematical and statistical models and not based on black box classification and regression techniques (e.g. neural network architectures). Thus, each model decision during the assessment procedure can be traced and justified. As a result, users can understand how a specific quality score was derived and validate the outcome. This leads to an increased confidence of the produced models, as the users are more likely to trust a transparent approach.

2.3.2. *Weights elicitation*

Contrary to the derivation of thresholds that is based on benchmark data (Deissenboeck, Heinemann, Herrmannsdoerfer, Lochmann, & Wagner, 2011), (Alves, Ypma, & Visser, 2010), weights elicitation cannot be data-driven. It is based exclusively on the opinion of experts in the field of software quality, and therefore the produced weights are inevitably characterized by subjectivity. Additionally, existing quality models and frameworks adopt rather naive techniques for weights elicitation.

[12] https://github.com/AuthEceSoftEng/qatch.

Towards this end, QATCH facilitates the weights elicitation procedure by the adoption of two multi-criteria decision making techniques, Analytic Hierarchy Process (AHP) (Saaty, 2008) and its fuzzy alternative. The fuzzy AHP approach allows quality experts to express uncertainties in the significance of specific evaluation criteria, while the subjectivity that their judgments inevitably encompass is also modeled. This information is automatically taken into consideration during the weights elicitation process, and therefore the expert opinions are effectively reflected in the produced weights.

Another novelty of the proposed technique compared to the related work that adopts fuzzy logic for the assessment of software products (e.g. (Yuen & Lau, 2011), (Chang, Wu, & Lin, 2008) and (Büyüközkan, Kahraman, & Ruan, 2004)), is that the assessment procedure is fully automated due to the adoption of static analysis, while it also allows experts to express their judgments, along with their associated uncertainty, into linguistic values instead of fuzzy numbers. This makes the process more intuitive and easier to use. The weights elicitation techniques are thoroughly described in Section 3, as they form the major contribution of this research effort.

3. Methodologies

3.1. *Quality model structure*

Fig. 2 illustrates the structure of the quality models derived by the proposed framework. It is obvious that the quality models adopt the classic hierarchical structure previously discussed in Section 2. More specifically, they have a simple three-layer form, similar to the one proposed by SIG (Heitlager, Kuipers, & Visser, 2007).

A quality model produced by QATCH comprises three layers: (i) the layer of characteristics, (ii) the layer of properties and (iii) the layer of measures. The first two layers correspond to the notions of characteristics and properties described in Section 2. In brief, the quality properties are evaluated directly through a set of well-chosen measures, while the quality characteristics are indirectly evaluated by exploiting the impact that properties have on them. As far as the layer of measures is concerned, the system distinguishes two individual types of measures as in (Lochmann & Heinemann, 2011), which are: (i) metrics and (ii) findings. Metrics are well-established source-code metrics proposed in the literature, such as cyclomatic complexity (McCabe, 1976); findings, also called violations, are the results produced by different types of static analysis tools such as bug pattern detectors and rule-based tools. QATCH employs CKJM Extended[13] and PMD[14] for the quantification of the metrics and findings, respectively. This way QATCH supports almost all the state-of-the-art metrics proposed over the years in the literature, while the derived quality models are highly customizable due to the fact that PMD also allows the incorporation of custom rulesets.

For simplicity purposes, each quality property is quantified by a single measure, while only one level of hierarchy is allowed for each layer. This practice, which is widely used in the literature (e.g. SIG Maintainability Model (Heitlager, Kuipers, & Visser, 2007)), ensures the hierarchical models:

- are more intuitive and comprehensible, even by people with little or no technical knowledge (Wagner et al., 2012).
- are easily extensible as their extension needs only the addition of new nodes, and easily reducible via appropriate cropping.
- can be effectively and easily represented by markup languages, such as XML, given their tree structure. Thus, they can be easily exploited by third party systems through the implementation of simple parsers.

From all the above, it is evident that the QATCH quality models are highly customizable. They are not restricted to a specific facet of software quality, nor to a particular standard or definition model. Furthermore, the wide range of metrics available to the stakeholders, along with the ability to define completely custom rulesets for the findings, make the quality model derivation process flexible and the derived models highly adaptable to custom needs. Last but not least, the simplicity of the model structure builds confidence to the model and enables root-cause analysis as will be discussed in the following sections.

3.2. Quality assessment

Quality assessment is the process of assigning a value to the software product under evaluation that indicates its quality according to a specific quality model. This value is called *quality score* or *rating*. The convention we follow for the quality score is that it lies in the interval between 0 and 1, where 0 corresponds to the worst possible quality, while 1 to excellent quality. Thus, the higher the rating, the better the quality of the software product against a specific quality model.

The overall rating of a software product is obtained from the quality scores of the properties and the characteristics of the quality model applied. QATCH divides the quality assessment process into two sequential levels: the first level of assessment is responsible for the evaluation of the properties, while the second for the evaluation of the characteristics and, in turn, of the overall quality of the software product. With the term “evaluation” we refer to the assignment of a quality score to each property or characteristic of the model. The two levels are thoroughly discussed in the following subsections.

3.2.1. Properties assessment level

The first level of assessment is responsible for the evaluation of the properties of the quality model, from the normalized values of the measures used for their quantification. Initially, the system receives as input the software product that should be evaluated and the QATCH quality model it should be evaluated against. QATCH identifies the different measures that should be calculated and invokes the appropriate static analysis tools with the appropriate configuration.

It should be mentioned that the absolute values of the measures (as calculated by the static analysis tools) are not suitable for the evaluation of the properties, since they are calculated at the source-code level (i.e. class level) and are highly dependent on the size of the product they refer to. Thus, aggregation and normalization is applied in order to bring them to the system level and make them size-independent. The normalized value M of a specific metric is calculated by applying the following formula:

$$M = \frac{x_1 LOC_1 + x_2 LOC_2 + \ldots + x_N LOC_N}{TLOC} = \frac{\sum_{i=1}^{N} x_i LOC_i}{TLOC} \qquad (2)$$

where:

x_i : the absolute value of the metric for the i-th class of the software product
LOC_i : the lines of code of the i-th class of the software product
$TLOC$: the total lines of code of the software product
N : the total number of classes that the software product comprises

The numerator of the aforementioned formula is the aggregation of the values of the metric for each class of the system under evaluation. These values are weighted by the lines of code of their corresponding class. This sum is divided by the total lines of code of the system in order to make the value size-independent. For the aggregation and normalization of findings the following formula is used:

$$F = \frac{v_1 w_1 + v_2 w_2 + \ldots + v_5 w_5}{TLOC} = \frac{\sum_{i=1}^{5} v_i w_i}{TLOC} \qquad (3)$$

where:

F : the normalized value of a specific finding
v_i : the number of violations that belong to the i-th severity category
w_i : the weight that reflects the importance of the i-th severity category
$TLOC$: the total lines of code of the software product

In brief, for a specific finding (i.e. PMD ruleset) the system calculates the number of violations that fall into each of the five severity categories as defined by the PMD. Subsequently, it calculates the sum of the violations weighted by a set of weights that reflect the importance of each severity category. In order to make the value independent of the product size, the sum is divided by the total lines of code (TLOC) of the system. The weights for each severity category can be defined by the user.

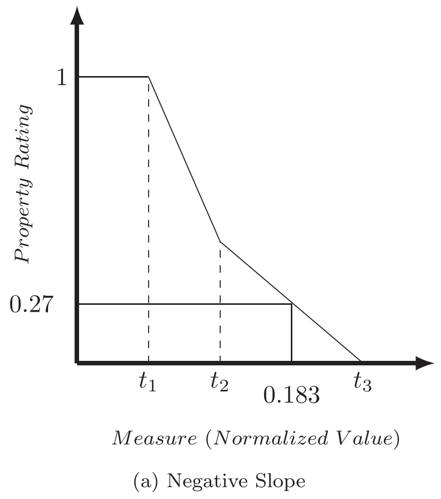
Next, the quality score of each property is calculated. For this purpose, we adopt the concept of utility functions initially proposed in (Wagner et al., 2012). Two examples of such functions are illustrated in Fig. 3.

The utility functions adopted have a partially linear form and the score of the property is calculated through linear interpolation applied between a set of measure-specific thresholds (i.e. t_1, t_2 and t_3). The slope of the function indicates the impact that the corresponding property has on the overall quality. If the function has an ascending slope, then the property has a positive impact on the overall quality. Conversely, if it has a descending slope, then the corresponding property has a negative influence on the overall quality.

The slope parameters and the thresholds of the utility functions are specified during the quality model derivation process which is described in Section 3.3. In particular, the slope is determined by software quality experts, while the thresholds via benchmarking.

The main difference between the utility functions used by QATCH and those proposed in (Wagner et al., 2012) is that in our case the utility functions correspond directly to the properties rather than to the measures. In other words, due to the fact that a single measure is used for the evaluation of each property, the utility function assigns a quality score directly to the corresponding

[13] http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/.
[14] http://pmd.sourceforge.net.



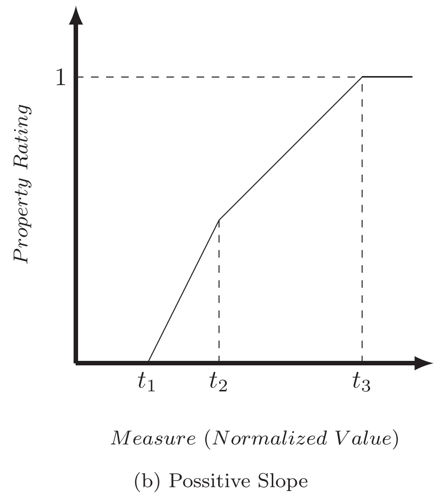


Fig. 3. Examples of utility functions. The values in the left graph are depicted for demonstration purposes based on an example in the end of this section.

property. Thus, the weighted average used in (Wagner et al., 2012) is not needed for the evaluation of the properties. This makes the quality models derived by our framework simpler to understand and to produce. The quality assessment process is also straightforward as the quality score assigned to a certain property indicates the quality level of its corresponding measure.

After the first level of assessment, a quality score is assigned to each property of the quality model. These scores are exploited by the second level of assessment in order to evaluate the quality model's characteristics and, in turn, the overall quality of the software product under evaluation.

For better understanding of the quality evaluation process for a specific property, a simple example is provided. Suppose that we want to evaluate the property of Complexity, which is quantified by the normalized value of the metric Cyclomatic Complexity. As the Complexity property has a negative impact on the overall quality, the first type of utility functions (lets assume that the utility function is that of Fig. 3a) is used for its evaluation. Let the thresholds t_1, t_2 and t_3 be 0.13, 0.16 and 0.21 respectively (as derived by the quality model derivation process described in Section 3.3) and the normalized value of the Cyclomatic Complexity be 0.183, which lies in the interval between the second and the third threshold. Thus, the utility function applies linear interpolation and assigns the appropriate quality score to the property of Complexity, which is 0.27.

3.2.2. *Aggregate assessment level*

The second level of assessment is responsible for the calculation of the overall quality score of software products. This is achieved by aggregating the quality scores of the quality model's properties and characteristics.

The aggregation function used by QATCH is the weighted average, an approach that is highly intuitive and comprehensible. The weights are a quantitative expression of the impact that the properties have on the characteristics and that the characteristics have on the overall quality. The overall quality Q of the software product under evaluation is calculated by the following formula:

$$Q = \sum_{i=1}^{N} wc_i qc_i = wc_1 qc_1 + wc_2 qc_2 + \ldots + wc_N q_N \tag{4}$$

where:

qc_i : the quality score of the i-th characteristic
wc_i : the weight that corresponds to the i-th characteristic, ranging in [0,1]

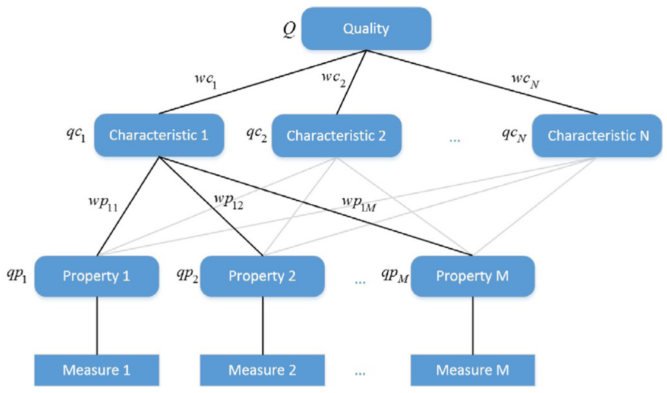


Fig. 4. The overall aggregation process.

N : the number of the quality model's characteristics

Similarly, the quality score of each characteristic is obtained by averaging the quality scores of the models properties. The formula used for the calculation of the quality score qc_i of the i-th characteristic is presented below:

$$\begin{aligned} qc_i &= \sum_{j=1}^{M} wp_{ij} qp_j \\ &= wp_{i1} qp_1 + wp_{i2} qp_2 + \ldots + wp_{iM} qp_M, \quad \forall i \in [1, M] \end{aligned} \tag{5}$$

where:

qp_j : the quality score of the j-th property
wp_{ij} : the weight that corresponds to the impact of the j-th property on the i-th characteristic, ranging in [0,1]
M : the number of the quality model's properties

The overall process is depicted in Fig. 4. It is quite obvious that the quality assessment procedure follows a bottom up approach.

The outcome of the quality assessment process, as stated previously, is the assignment of a software quality score that lies in the [0,1] interval. However, for better comprehensibility, a discrete rating is provided as well. QATCH rating has a 5-star mapping, based on the intervals presented in Table 1.

At this point we should state that, by default, each property is considered to have impact on more than one characteristics of the given quality model. This is why each property is linked to all the characteristics of the model as depicted in Fig. 2. If a property does

Table 1
Quality Score Mapping.

Quality Score	Stars
(0.8, 1.0]	⋆ ⋆ ⋆ ⋆ ⋆
(0.6, 0.8]	⋆ ⋆ ⋆ ⋆
(0.4, 0.6]	⋆ ⋆ ⋆
(0.2, 0.4]	⋆ ⋆
[0, 0.2]	⋆

not influence a specific characteristic, this is represented quantitatively with the assignment of a zero or of a really small weight. Thus, the quality score of this property is not taken into consideration during the evaluation of the specific characteristic (Fig. 4).

The exact wc_i and wp_{ij} values are calculated based on the opinions of experts in the field of software quality by applying both deterministic and fuzzy multi-criteria decision-making techniques. These techniques are discussed in the next subsection.

3.3. QATCH quality models

From the above discussion, it is quite obvious that the basic design parameters of a quality model are the thresholds and the weights used by the properties and aggregate assessment levels. The process of specifying these design parameters is called calibration. Following the same approach as with quality assessment, QATCH defines two levels of calibration. The first level is responsible for deriving the thresholds of the model (Section 3.3.1), while the second for eliciting the weights of the model (Section 3.3.2). The QATCH calibration approach follows an approach similar to (Alves, Correia, & Visser, 2011), (Alves, Ypma, & Visser, 2010) and (Baggen, Correia, Schill, & Visser, 2012); however, our approach differentiates at the second level of calibration, where we calculate the weights of the model and not metric-specific thresholds.

3.3.1. Threshold calibration

QATCH calculates the thresholds by applying statistical analysis techniques to a manually constructed benchmark set of software products (i.e. a benchmark repository). The process is as follows.

Initially, a benchmark repository of quality software products is carefully constructed by the user. For each software product found in the repository, the system performs properties assessment. As a result, the normalized values of the quality models measures are calculated for each benchmark product.

Subsequently, for each measure, the system selects three thresholds based on the distribution of their normalized values observed between the different products. In particular, the framework adopts the formulas introduced in Lochmann (2012):

$$\begin{aligned} t_1 &= \min(\{x : x \geq Q_{25\%}(x_1, \ldots, x_n) - 1.5 \cdot IRQ(x_1, \ldots, x_n)\}) \\ t_2 &= \text{median}(x_1, \ldots, x_n) \\ t_3 &= \max(\{x : x \leq Q_{75\%}(x_1, \ldots, x_n) + 1.5 \cdot IRQ(x_1, \ldots, x_n)\}) \end{aligned} \tag{6}$$

where:

x_i denotes the normalized value of the measure x of the i-th benchmark product
Q_p denotes the p-percentile
$IRQ(x_1, \ldots, x_n)$ denotes the inter-quartile-range:

$$IRQ(x_1, \ldots, x_n) = Q_{75\%}(x_1, \ldots, x_n) - Q_{25\%}(x_1, \ldots, x_n) \tag{7}$$

In simple words, the framework selects as thresholds of each measure the minimum, the median and the maximum observations after removing outliers. Thus, the selected thresholds bring out the maximum variability of the measures observed between the systems of the benchmark repository.

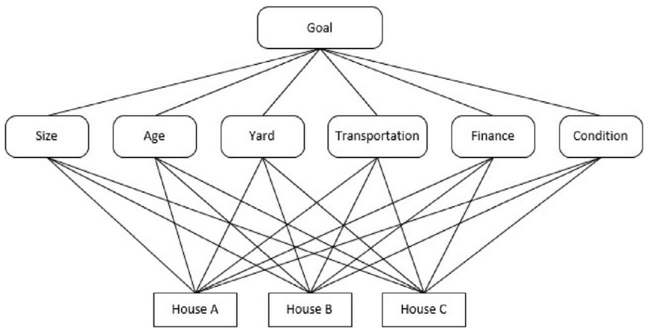


Fig. 5. An example of a hierarchical multi-criteria decision-making problem - The selection of the optimal house.

The reasoning behind the adoption of benchmarking for the thresholds derivation is that:

- The threshold derivation process is data-driven, and therefore thresholds are based on data and not on experts opinions. Thus, the quality assessment process is highly objective, while the validity of the model is enhanced.
- The derivation of thresholds is fully automated.
- Comparison of two software products, with respect to their quality is now feasible, since they are evaluated against the same reference system (i.e. benchmark repository).
- It is a technique widely used in the literature for thresholds derivation of source-code measures (e.g. Quamoco (Wagner et al., 2012) and SIG Maintainability Model (Heitlager, Kuipers, & Visser, 2007)).

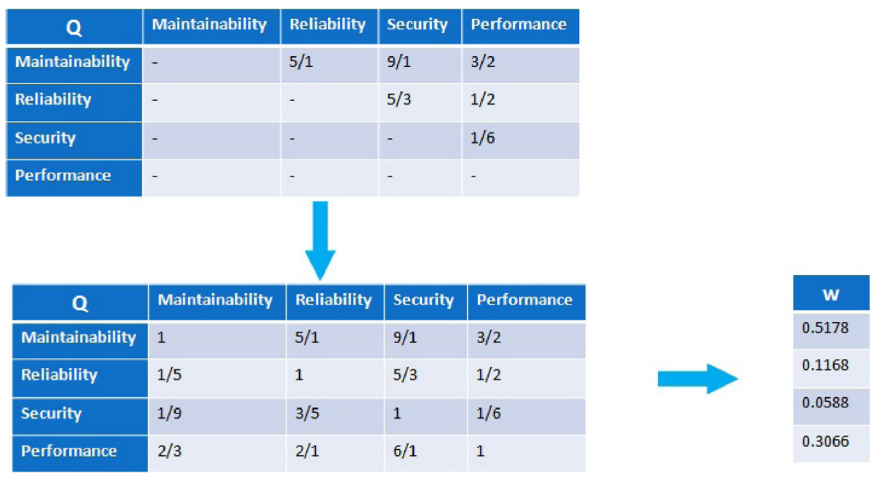
The basic disadvantage of such a benchmarking process is that it is an extremely time-consuming process. Both static and statistical analysis techniques have to applied to a wide range of software products in order to ensure the validity of the derived models. This problem can be solved via parallelization and, towards this end, we have implemented a simple approach that adopts multi-threading in order to accelerate the quality model derivation process.

3.3.2. Weight calibration

As described in Section 2, the major shortcoming of the existing quality assessment models and frameworks is the lack of a sophisticated method for the elicitation of the quality models weights. Contrary to the thresholds derivation, the weights elicitation process cannot be data-driven. It is based on the opinions of experts in the field of software quality. The main reason for this is that the notion of quality is highly subjective and the extend to which a property or a characteristic influences the overall quality depends on how the individual stakeholders perceive the notion of quality. As a result, the weights are also characterized by subjectivity, and therefore they should be elicited with care.

Towards this end, QATCH provides two approaches for weights elicitation: a) it employs the Analytic Hierarchy Process (AHP) (Saaty, 2008) and b) it introduces a fuzzy alternative in order to model the experts judgments uncertainties. The latter method is enhanced in order to support judgments expressed in natural language, making the whole process easier and more intuitive.

3.3.2.1. Analytic Hierarchy Process. The Analytic Hierarchy Process (AHP) (Saaty, 2008) is an approach followed for decision making that reduces decisions complexity when it comes to pairwise comparisons. It can be applied to hierarchical multi-criteria decision-making problems that consist of an overall goal, a set of criteria that influence the final choice and a set of alternatives. In brief, the AHP is employed to assist decision makers in choosing the best among a set of alternatives. An example of the general form of the problems solved by the AHP approach is depicted in Fig. 5.



Q	Maintainability	Reliability	Security	Performance
Maintainability	-	5/1	9/1	3/2
Reliability	-	-	5/3	1/2
Security	-	-	-	1/6
Performance	-	-	-	-

Q	Maintainability	Reliability	Security	Performance
Maintainability	1	5/1	9/1	3/2
Reliability	1/5	1	5/3	1/2
Security	1/9	3/5	1	1/6
Performance	2/3	2/1	6/1	1

W
0.5178
0.1168
0.0588
0.3066

Fig. 6. An example of applying the AHP approach for weights elicitation.

We argue that, given its hierarchical structure, the problem of weights elicitation of a quality model can be treated as a multi-criteria decision making problem; thus, AHP can be applied in order to elicit the quality models weights based on the opinions of software quality experts. Inspired chiefly by the idea proposed in (Kanellopoulos et al., 2010), we equipped QATCH with the AHP technique for eliciting the quality models weights. In particular, the AHP approach is applied to both levels of assessment in order to reflect the importance of properties and characteristics in the overall quality score.

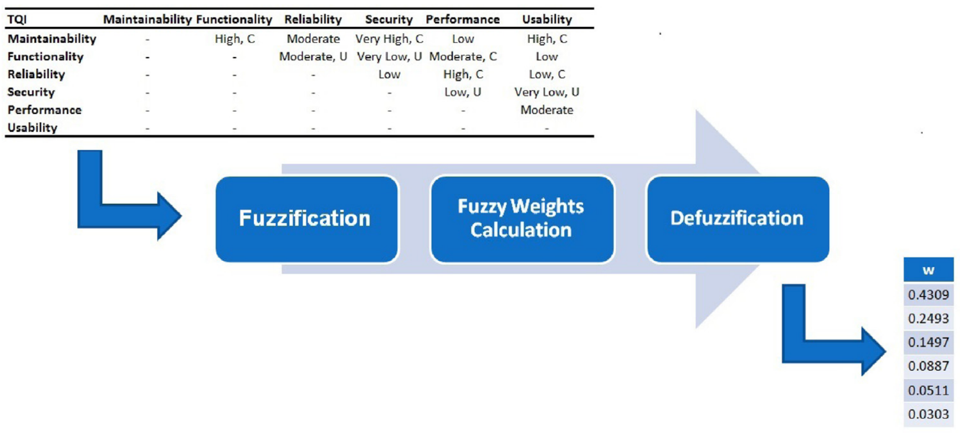
For better understanding of AHP, an example is provided and presented in Fig. 6. Suppose that we have a quality model that consists of four quality attributes, namely maintainability, reliability, performance efficiency and security. In order to elicit the weights used for the calculation of the overall quality score (Q) from the quality scores of the characteristics, a matrix, normally known as pair-wise comparison matrix, is conducted. An expert in the field of software quality completes the upper triangular part of the matrix with his/her judgments, stating how one quality attribute affects (relates to) another. The judgments are values of the form w_{row}/w_{col}, where the w_{row} and w_{col} receive an integer value between 1 and 9. For instance, the entry 5/1 of the example (cell [1,2]) denotes that maintainability is five times more important than reliability with respect to the overall quality of the system, as perceived by the software experts for this particular software product. Subsequently, the framework completes the main diagonal of the matrix with ones and the lower triangular part with the reciprocal values of the transpose cells. Eventually, the principal eigenvector (i.e. the eigenvector that correspond to the maximum eigenvalue) normalized by the sum of its entries is selected to be the desired weights vector of the corresponding pair-wise comparison matrix. Normalization is important, since the weights should have a sum of 1. The approach guarantees that the derived weights reflect the experts judgments and therefore the impacts that the characteristics have on the overall quality.

The same approach is applied for the elicitation of the weights used for the evaluation of the models characteristics from the ratings of the models properties. For each characteristic a pair-wise comparison matrix is constructed. The difference is thattable rows and columns are now the properties of the quality model and the comparison matrix refers to a specific characteristic and not to overall software quality.

3.3.2.2. Fuzzy AHP. Although the AHP approach constitutes a reliable and mathematically justified approach for weights elicitation, the derived weights are still based on human judgments. As a result, AHP, and therefore the whole assessment, will be characterized by subjectivity, which is an inherent characteristic of AHP as stated in (Kanellopoulos et al., 2010). Furthermore, AHP does not take into consideration the uncertainty that inevitably underlies in the experts judgments. For instance, if an expert believes that a property is **almost** five times more important than another, he/she cannot express this uncertainty to the system. Thus, an extension of this approach is necessary in order to allow the modeling of such uncertainties.

To this end, fuzzy logic is applied. Numerous fuzzy extensions to AHP have been proposed ((Van Laarhoven & Pedrycz, 1983), (Buckley, 1985), (Boender, de Graan, & Lootsma, 1989), (Wang & Chin, 2011)), all arguing that they lead to more realistic results compared to the original non-fuzzy approach. However, in these cases, experts have to provide much more information which makes the whole process time-consuming, while relevant mathematical background is necessary in order to understand the process. Thus, a more intuitive approach is needed in order to facilitate the experts in their decision-making tasks. This can be achieved by allowing the experts to express their opinions into linguistic values, rather than by obliging them to directly define the parameters of the fuzzy numbers. The linguistic values have the advantage of being more comprehensive and acceptable by the decision-makers, as the human brain is more efficient in manipulating qualitative than quantitative values.

In Yuen and Lau (2011), which is an improvement of Büyüközkan, Kahraman, and Ruan (2004) and Chang, Wu, and Lin (2008), a fuzzy group AHP method is proposed in order to assess the quality of software products based on experts judgments. However, the quality score provided by the model is a fuzzy triangular number, and not a numeric score, which constitutes the communication of its meaning to stakeholders with no relevant mathematical background difficult. Other drawbacks of this model are that it is based exclusively on ISO/IEC 9126, not allowing much tailoring, while the quality assessment process is not automated, since the fuzzy AHP (FAHP) approach has to be applied both for the weights elicitation and for the evaluation of characteristics through measures. This means that the presence of experts is necessary even during the quality evaluation process. Finally, the model allows the experts to choose one of only nine (9) available predefined and labeled fuzzy numbers in order to express each one of their judgments, restricting in this way their options, and in turn, the calculation capabilities of the overall approach. QATCH tackles these shortcomings by (i) providing a crisp expression of the final calculated quality score, (ii) evaluating the model's properties via static



TQI	Maintainability	Functionality	Reliability	Security	Performance	Usability
Maintainability	-	High, C	Moderate	Very High, C	Low	High, C
Functionality	-	-	Moderate, U	Very Low, U	Moderate, C	Low
Reliability	-	-	-	Low	High, C	Low, C
Security	-	-	-	-	Low, U	Very Low, U
Performance	-	-	-	-	-	Moderate
Usability	-	-	-	-	-	-

w
0.4309
0.2493
0.1497
0.0887
0.0511
0.0303

Fig. 7. The fuzzy AHP weights elicitation process.

analysis in order to automate the assessment process, (iii) allowing the production of custom quality models, in order to satisfy specific needs imposed by the stakeholders and, (iv) by allowing the experts to express both their judgments and their associated uncertainty into linguistic values, information that is exploited for the definition of the parameters of the corresponding fuzzy numbers, making in this way the weights elicitation process more robust.

The QATCH proposed approach adopts the widely used Fuzzy Logarithmic Least Squares method (described in Section 3.3.2.2) in order to derive a set of fuzzy weights from pair-wise comparison matrices completed with judgments expressed in fuzzy triangular numbers. It is enhanced in order to (i) allow the experts express their opinions into linguistic variables and (ii) produce crisp weights that can be directly used by the quality models derived by the framework. In QATCH, the fuzzy AHP is employed only once for the elicitation of the quality models weights; as a result, quality assessment is fully automated.

The proposed technique comprises three steps as depicted in Fig. 7. These steps are (i) fuzzification, (ii) fuzzy weights calculation, and (iii) defuzzification. In brief, the expert is asked to complete a set of pair-wise comparison matrices with linguistic variables expressing both his/her opinion and his/her uncertainty. During the fuzzification step, a fuzzifier receives each matrix as input and constructs the corresponding fuzzy pair-wise comparison matrix containing the appropriate fuzzy triangular numbers (see definition below). The Fuzzy Logarithmic Least Squares method is applied and a vector of fuzzy weights is exported for each matrix. These weights are inserted into a defuzzifier and the corresponding crisp weights are produced. These steps are thoroughly described in the following subsections.

Background of fuzzy triangular numbers. At this point, it should be stated that we consider triangular fuzzy numbers in the form of $F_i = (l_i, m_i, u_i)$, following the definition given by DuBois (1997). For better understanding, this definition is given below:

Definition 1. We define a fuzzy number M on $R = (-\infty, +\infty)$ to be a triangular fuzzy number if its membership function μ_M: $R \rightarrow [0, 1]$ is equal to:

$$\begin{cases} \frac{1}{m-l}x - \frac{l}{m-l}, & x \in [l, m] \\ \frac{1}{m-u}x - \frac{u}{m-l}, & x \in [m, u] \\ 0, & othrerwise \end{cases} \tag{8}$$

with $l \leq m \leq u$, where l, m and u stand for the lower, modal and upper value of the triangular number, respectively. The support of M is the set of elements $\{x \in R | l < x < u\}$. The triangular number M is depicted in Fig. 8 and we refer to it as $M = (l, m, u)$.

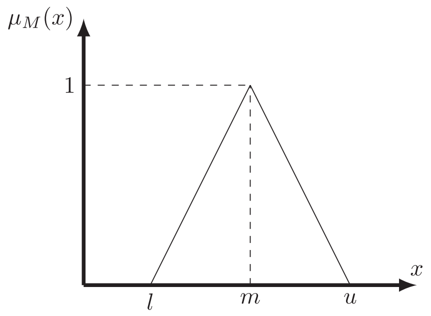


Fig. 8. The membership function of a fuzzy triangular number.

Additionally, we extend the algebraic operations to triangular fuzzy numbers according to the principle proposed by Zadeh (1965) and the characteristics of fuzzy triangular numbers presented by Liang and Wang (1991). In particular, considering two triangular fuzzy numbers $F_1 = (l_1, m_1, u_1)$ and $F_2 = (l_2, m_2, u_2)$, the extension of algebraic operations to triangular fuzzy numbers can be expressed as follows:

$$\begin{aligned} &F_1 + F_2 = (l_1 + l_2, m_1 + m_2, u_1 + u_2) \\ &F_1 \times F_2 = (l_1 \times l_2, m_1 \times m_2, u_1 \times u_2) \\ &F_1 \div F_2 = (l_1/u_2, m_1/m_2, u_1/l_2) \\ &\tfrac{1}{F_1} = \left(\tfrac{1}{u_1}, \tfrac{1}{m_1}, \tfrac{1}{l_1}\right) \end{aligned} \tag{9}$$

After the definition of these operations, the extension of the AHP in the field of fuzzy numbers is a straightforward process. These operations are used for the calculation of the fuzzy weights from the fuzzy pair-wise comparison matrices by applying the Fuzzy Logarithmic Least Squares method described in Section 3.3.2.2

Weights Elicitation Process. The first step of the weights elicitation process, as depicted in Fig. 7, is the derivation of the fuzzy pair-wise comparison matrices that are required for the application of the fuzzy AHP approach from the corresponding matrices containing the linguistic values provided by the expert. For this purpose, the software quality expert is asked to complete a set of pair-wise comparison matrices. In particular, the expert has to complete each cell of the matrix with a linguistic value, selecting one of *Very Low, Low, Moderate, High and Very High*, which expresses his/her judgment. For instance, if the expert selects a *Very High* value, he/she states that the property (or characteristic) of the corresponding row is far more important (quality-wise) compared to the property (or characteristic) of the corresponding column of the matrix. QATCH also allows experts to express the certainty of their decision. *U, D* and *C* stand for *Uncertain, Default* and *Certain*, respectively. Thus, each pair-wise comparison matrix consists of pairs of (*judgment, certainty*) linguistic variables.

In order to construct the fuzzy pair-wise comparison matrices from the pair-wise comparison matrices containing linguistic values, a *Fuzzification* module is employed (see Fig. 7). For each pair of linguistic values (e.g. $<$ *High, C* $>$) the fuzzifier assigns an appropriate fuzzy triangular number of the form $r = (l, m, u)$, where *l, m* and *u* correspond to the low-, modal- and upper-value respectively. Suppose σ a constant that defines the distance of the lower and upper border of a fuzzy triangular number from its modal value. As this constant defines the range of the triangular set, it is an indicator of the judgments uncertainty. Thus, the fuzzy numbers derived from the fuzzifier have the form $r = (m - \sigma, m, m + \sigma)$.

Consequently, the fuzzifier has to determine the modal (m) and the uncertainty (σ) values for each triangular fuzzy number. The linguistic variable values provided by the experts concerning their judgments are used for determining the center of the fuzzy triangular numbers, while the linguistic variable values concerning the certainty of each judgment are used for determining the σ of each fuzzy number. In particular, each one of the allowed linguistic values used for expressing the expert judgments (i.e. *Very Low, Low, Moderate, High* and *Very High*), leads to a fuzzy triangular number with modal m that receives an integer value among 1 and 5 respectively (i.e. the integer values are assigned based on the order of the linguistic values). Similarly, a completely uncertain judgment (i.e. a judgment with $certainty = U$) corresponds to a triangular fuzzy number with $\sigma = 0.9$. Accordingly, judgments with *certainty* values of *D* and *C*, correspond to fuzzy triangular numbers with a σ value of 0.5 and 0.1, respectively.

The system iterates through all the pairs of the linguistic variable values and calculates the corresponding fuzzy number according to the aforementioned process. The outcome is a fuzzy pair-wise comparison matrix that contains the corresponding fuzzy judgments. QATCH applies this process for each pair-wise comparison matrix created by the system for the elicitation of all the weights needed by the quality model.

The fuzzy pair-wise comparison matrices produced by the previous step are provided as inputs to the *Fuzzy Weights Calculation* module (see Fig. 7). This module is responsible for calculating a set of fuzzy weights for each one of the received matrices, by applying the Fuzzy Logarithmic Least Square method (see the definition below), which is commonly used in the literature for similar purposes (e.g. Yuen & Lau (2011)). For comprehension reasons, a formal mathematical description of the approach is provided below.

Suppose a fuzzy pair-wise comparison matrix *R*, which was generated during the fuzzification phase (see Fig. 7), and r_{ij} is an entry of this matrix (a fuzzy triangular number of the form $r_{ij} = (l_{ij}, m_{ij}, u_{ij})$) that corresponds to the expert's judgment concerning the properties P_i and P_j (or the characteristics C_i and C_j). The fuzzy logarithmic least squares method is employed for the calculation of the desired fuzzy weights. In particular, according to Boender, de Graan, and Lootsma (1989) the challenge is to calculate the fuzzy weights $a_1, a_2, \ldots, a_n$ that minimize the fuzzy version of the logarithmic regression function:

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\left\{\ln(r_{ijl}) - \ln(a_{il}) + \ln(a_{ju})\right\}^2 + \left\{\ln(r_{ijm}) - \ln(a_{im}) + \ln(a_{jm})\right\}^2 + \left\{\ln(r_{iju}) - \ln(a_{iu}) + \ln(a_{jl})\right\}^2 \tag{10}$$

According to Boender, de Graan, and Lootsma (1989) and Buckley (1985), the solution of the optimization problem is expected to converge to the geometric mean. Thus, the fuzzy weights obtained by this method have the following lower, modal and upper values:

$$(a_{il}, a_{im}, a_{iu}) = \left(\frac{\prod_{j=1}^{n}(r_{ijl})^{1/n}}{\sum_{i=1}^{n}\prod_{j=1}^{n}(r_{iju})^{1/n}}, \frac{\prod_{j=1}^{n}(r_{ijm})^{1/n}}{\sum_{i=1}^{n}\prod_{j=1}^{n}(r_{ijm})^{1/n}}, \frac{\prod_{j=1}^{n}(r_{iju})^{1/n}}{\sum_{i=1}^{n}\prod_{j=1}^{n}(r_{ijl})^{1/n}}\right) \tag{11}$$

The value n used in the formulas above corresponds to the number of the derived weights.

This approach is applied for each comparison matrix produced by the system in order to determine their corresponding fuzzy weights. Thus, the outcome of this module is a vector of fuzzy weights for each one of the received matrices.

Due to the fact that the weights derived by the previous process are fuzzy triangular numbers, they cannot be directly used by the derived quality model for the conduction of quality assessments. The produced weights should be in a crisp form in order to be used in the weighted average formulas. Thus, the fuzzy weights are passed through the *Deffuzzification* module (see Fig. 7), which is responsible for the calculation of the corresponding exact values.

For the defuzzification of the fuzzy weights, the center of mass approach is applied, which was proposed by Dimiter Driankov (1996). This method is expressed by the following formula:

$$F = \int x\mu(x)d_x \Big/ \int \mu(x)d_x \tag{12}$$

As mentioned previously, the proposed crisp weights constitute the parameters of the weighted average formulas of the produced quality model. This implies that they should sum to 1. However, the deffuzzification step may introduce a small error (i.e. computation error) to the resulting crisp weights, leading their sum to slightly diverge from the desired value. Therefore, the weights are divided by their sum, which guarantees that the sum of the final weights will be equal to 1.

Finally, with respect to the normalization procedure, suppose that $\bar{w}$ is the weights vector derived by the previously described process and w_i corresponds to its *i*-th weight. Also, suppose that the length of the vector is *n*. Each element (i.e. weight) w_i' of the normalized weights vector $\bar{w}'$ is calculated using the following equation:

$$w_i' = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{13}$$

4. QATCH architecture

The overall QATCH architecture is illustrated in Fig. 9, where four modules (i.e. subsystems) can be identified:

(a) The Quality Model Designer (QMD)
(b) The Single Project Evaluator (SPE)
(c) The Multi Project Evaluator (MPE)
(d) The Quality Certification App (QCA)

In brief, the QMD, based on the specifications provided by the user, exports the desired quality model in XML format. The XML file containing the quality model is used by the other three subsystems for the evaluation of both proprietary and open source software products. In the following sub-sections the main features and functionalities of the aforementioned modules are analyzed.

4.1. The quality model designer

The Quality Model Designer (QMD) constitutes the cornerstone of the framework, since all the other modules are highly depen-

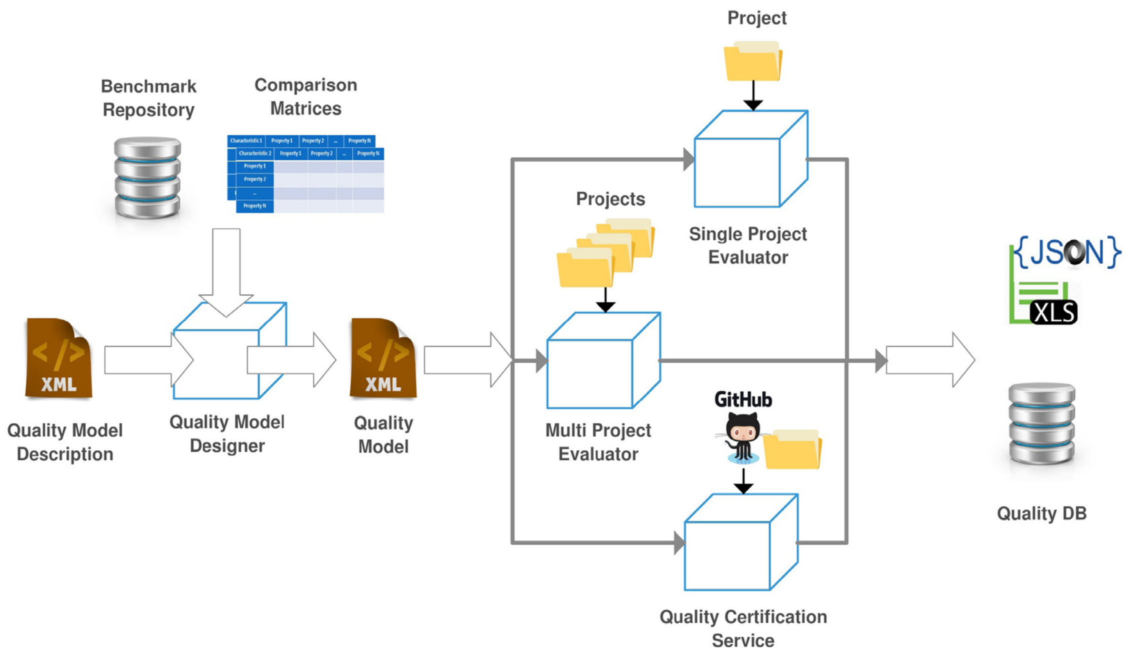


Fig. 9. The overall structure of QATCH.

dent on it. It is responsible for the derivation of a quality model based on user specifications. The QMD receives as an input: (1) the description of a quality model in XML format, (2) the desired benchmark repository needed for the thresholds derivation and (3) the appropriate pair-wise comparison matrices needed for the weights elicitation. QMD, based on the analysis performed as discussed in the previous section, exports an XML file containing the desired software quality model.

QATCH offers a QMD GUI wizard that guides the user throughout the whole model derivation process. Through the GUI the user is able, among others, to:

- Create/Store/Load/Update a description of the desired quality model.
- Define the benchmark repository to be used for thresholds generation.
- Define the weights elicitation technique (i.e. AHP or fuzzy AHP) and generate the appropriate pair-wise comparison matrices.
- Define the parallelization parameters for the static analysis process.

The aforementioned features constitute the basic configuration that the user should provide for the derivation of the desired quality model. Validation checks are performed whenever new data are provided, thus guaranteeing input quality.

4.2. *Offline quality assessment tools*

For the offline quality assessment process two individual subsystems were built, the Single Project Evaluator (SPE) and the Multi Project Evaluator (MPE). The former is responsible for the quality assessment of a locally stored software product according to a desired quality model, previously derived by QMD, while the latter performs evaluation on a group of locally stored software products. Instantiations of SPE and MPE have been developed for Java projects. However, the system is language agnostic and can be instantiated for other programming languages, if the appropriate language-specific rulesets are selected for the PMD tool or if other static analysis tools specialized in calculating source code metrics for common languages are added to the system.

SPE and MPE evaluate the software products by applying the methodologies described in Section 3. As depicted in Fig. 9, the evaluation results are stored in the QATCH database, while they can be exported in widely used formats such as JSON and XLS. Thus, the results can be easily transferred and analyzed, while they can be easily exploited by third-party applications via implementation of simple parsers or via querying the database.

4.3. *The QATCH certification application*

The QATCH Certification App (QCA) is a web application that allows the quality assessment of open source software Java products residing on GitHub.[15] The QCA landing page is available at: http://issel.ee.auth.gr/qatch.

In order to perform software evaluation, a user has to provide the following input:

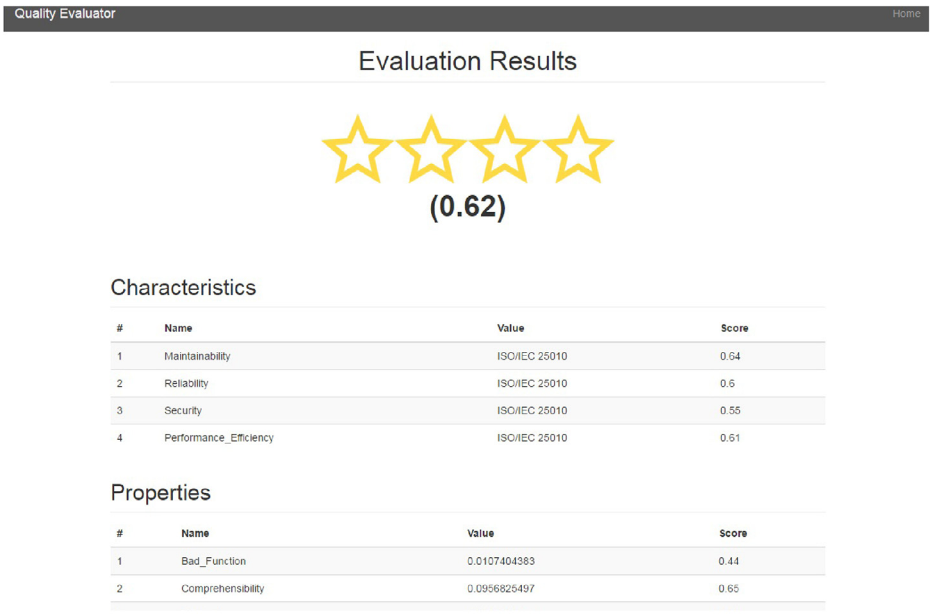
- The GitHub url of his/her Java project that he/she would like to evaluate.
- The desired quality model (one of the available) that should be used for the quality assessment of the desired product.

Upon the selection of the *Evaluate Project* button, the server clones the desired project locally, compiles it by using *Maven*[16] and performs quality assessment. After the successful execution of the evaluation, an HTML page is printed on the users screen, presenting the quality score of the product and the models properties and characteristics scores. An example of such a page is illustrated in Fig. 10.

5. Experiments and discussion

In order to prove the validity of the proposed framework and investigate its added value in the field of software quality, six individual experiments were performed. A carefully calibrated baseline quality model, the basic model, was derived in order to help

[15] https://github.com/.
[16] https://maven.apache.org/.



Evaluation Results

(0.62)

Characteristics

#	Name	Value	Score
1	Maintainability	ISO/IEC 25010	0.64
2	Reliability	ISO/IEC 25010	0.6
3	Security	ISO/IEC 25010	0.55
4	Performance_Efficiency	ISO/IEC 25010	0.61

Properties

#	Name	Value	Score
1	Bad_Function	0.0107404383	0.44
2	Comprehensibility	0.0956825497	0.65

Fig. 10. An example of the results page of the online service.

us conduct the experiments and reach useful conclusions. The experiments presented in this section also helped us to identify the strong points, but also the weaknesses of the proposed framework. Before proceeding to their description, a brief reference to the basic model is provided.

5.1. Basic model

The basic model consists of 4 characteristics retrieved from the ISO/IEC 25010 standard and 11 properties. The properties are evaluated both by state-of-the-art metrics (e.g. Cyclomatic Complexity (McCabe, 1976)) and by custom PMD rulesets constructed by us in order for their rules to reflect the corresponding properties. For the calibration of the model, a benchmark repository of 100 Java products was constructed. The software products were retrieved from the Maven Repository.[17] In particular, the top 100 Java libraries (based on their reputation) were selected for the benchmarking purposes. The benchmark repository comprises almost 6.5 million LOC. The pair-wise comparison matrices required for the weights elicitation of the quality model were completed by the authors to their best of their knowledge, as experts in the field of software quality. In particular, the pair-wise comparison matrices were completed with linguistic values (like in the first table presented in Fig. 11), and subsequently the fuzzy AHP approach was employed for the derivation of the model's weights.

The reasoning behind the selection of Java libraries as benchmark products for the derivation of the basic model is threefold. First of all, we chose Java as a proof of concept because it is a very popular object oriented programming language with official guides for developers[18] and best practices concerning how to enhance the quality of the product source code. Secondly, the selected libraries are widely used by millions of developers for the implementation of their own applications, and therefore, it is likely that their development and maintenance to be based on best practices and coding standards, which acts as an inherent indicator of quality. Last but not least, the Maven Repository provides both the sources and the binaries of the libraries, which are needed for the calibration of the quality model.

5.2. Minimum benchmark repository size

In order to ensure the validity of the basic model, we have to make sure that the benchmark repository used for its calibration is of the appropriate size. In other words, we had to check if the size of the benchmark repository influences significantly the derived thresholds. For this purpose, hypothesis testing was applied. Specifically, we tested whether the thresholds derived by using all the 100 products of the benchmark repository are statistically similar to those produced by using a subset of them. We defined the following null hypothesis:

$H_{0,\,n}$: *There is no statistical difference between the thresholds derived by the first n products and those derived by the whole benchmark repository.*

Thus, the question that we had to answer is the following:

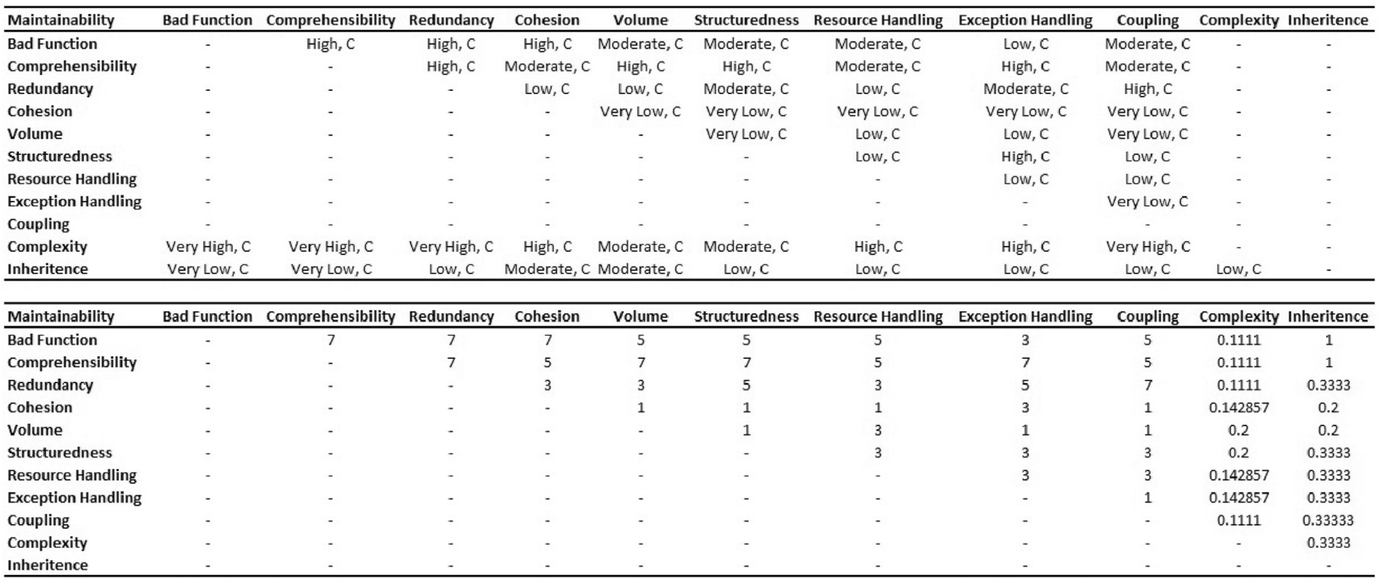
"For confidence level 95% can we reject the null hypothesis?"

We executed the hypothesis test repeatedly for benchmark repositories of various sizes. In particular, for each experiment we initially applied *F-test* in order to investigate the homogeneity of the thresholds variability. Based on the results of this test we applied *Student's t-test* if the variability was homogenous and *Welch's t-test* otherwise.

Table 2 depicts the *p*-values of the tests concerning the lower thresholds of the model. From this table one can easily notice that the *p*-value is inversely proportional to the benchmark repository size. We can also see that it falls below 0.05 for the first time when the benchmark repository comprises 25 software products, which corresponds to 1,347,993 LOC. For even smaller repositories the *p*-value declines even more. Thus, we can conclude that the minimum LOC that the benchmark repository should contain in order for the null hypothesis not to be rejected is almost 1.5 million in this case. The benchmark repository used for the calibration of the basic model comprises 6.5 million LOC, and therefore it highly satisfies the minimum requirements.

[17] https://mvnrepository.com/.

[18] http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-139411.html.



Maintainability	Bad Function	Comprehensibility	Redundancy	Cohesion	Volume	Structuredness	Resource Handling	Exception Handling	Coupling	Complexity	Inheritence
Bad Function	-	High, C	High, C	High, C	Moderate, C	Moderate, C	Moderate, C	Low, C	Moderate, C	-	-
Comprehensibility	-	-	High, C	Moderate, C	High, C	High, C	Moderate, C	High, C	Moderate, C	-	-
Redundancy	-	-	-	Low, C	Low, C	Moderate, C	Low, C	Moderate, C	High, C	-	-
Cohesion	-	-	-	-	Very Low, C	Very Low, C	Very Low, C	Very Low, C	Very Low, C	-	-
Volume	-	-	-	-	-	Very Low, C	Low, C	Low, C	Very Low, C	-	-
Structuredness	-	-	-	-	-	-	Low, C	High, C	Low, C	-	-
Resource Handling	-	-	-	-	-	-	-	Low, C	Low, C	-	-
Exception Handling	-	-	-	-	-	-	-	-	Very Low, C	-	-
Coupling	-	-	-	-	-	-	-	-	-	-	-
Complexity	Very High, C	Very High, C	Very High, C	High, C	Moderate, C	Moderate, C	High, C	High, C	Very High, C	-	-
Inheritence	Very Low, C	Very Low, C	Low, C	Moderate, C	Moderate, C	Low, C	Low, C	Low, C	Low, C	Low, C	-

Maintainability	Bad Function	Comprehensibility	Redundancy	Cohesion	Volume	Structuredness	Resource Handling	Exception Handling	Coupling	Complexity	Inheritence
Bad Function	-	7	7	7	5	5	5	3	5	0.1111	1
Comprehensibility	-	-	7	5	7	7	5	7	5	0.1111	1
Redundancy	-	-	-	3	3	5	3	5	7	0.1111	0.3333
Cohesion	-	-	-	-	1	1	1	3	1	0.142857	0.2
Volume	-	-	-	-	-	1	3	1	1	0.2	0.2
Structuredness	-	-	-	-	-	-	3	3	3	0.2	0.3333
Resource Handling	-	-	-	-	-	-	-	3	3	0.142857	0.3333
Exception Handling	-	-	-	-	-	-	-	-	1	0.142857	0.3333
Coupling	-	-	-	-	-	-	-	-	-	0.1111	0.33333
Complexity	-	-	-	-	-	-	-	-	-	-	0.3333
Inheritence	-	-	-	-	-	-	-	-	-	-	-

Fig. 11. Comparison matrices that are supposed to lead to same weights.

Table 2
The *p*-values against benchmark repository size in terms of number of products and lines of code.

Number of Products (n)	Lines of Code (*LOC*)	*p*-value
90	5,057,202	0.1704
80	4,792,348	0.1482
70	4,590,831	0.1124
60	3,738,618	0.1392
50	3,265,838	0.06411
40	2,346,764	0.05925
30	1,691,153	0.05687
25	**1,347,993**	**0.04859**
20	1,064,100	0.04719
15	959,179	0.04981
10	414,563	0.03738

5.3. Comparison of weights elicitation techniques

In order to ensure the validity of the modified fuzzy AHP technique, the produced weights should be closely related to those derived by AHP. A set of experiments were performed for this purpose. An example of such an experiment is presented in Fig. 11.

Suppose we have a deterministic pair-wise comparison matrix and three appropriate comparison matrices completed with linguistic values that they are supposed to lead to similar weights. More specifically, we have three additional pair-wise comparison matrices, where the second linguistic value of all their cells is *U*, *D* and *C* respectively. Weights elicitation is applied and the correlation between the derived weights vector of each of these three matrices with the weights produced by AHP is calculated. Table 3 contains the exact weights derived from each matrix by applying the appropriate weights elicitation technique and the corresponding correlation with the weights derived by AHP.

The results of this experiment indicate that the thresholds derived by fuzzy AHP are closely related to those calculated by AHP, even to the extreme case where all the judgments are uncertain (i.e. the second linguistic value of all the cells is *U*). The third column of Table 3 contains the weights derived from the corresponding pair-wise comparison matrix, where all the provided judgments are certain (i.e. the second linguistic value of all its cells is *C*), but the corresponding σ is set to 0.0001. In other words, the fuzzy triangular numbers have an extremely small range, and therefore they converge to singletons. In this extreme case, the derived weights are almost equivalent to those produced by the AHP approach, as the correlation is approximately equal to 1 and has its highest value compared to the other three cases, which is the desired behavior. However, in all the cases the correlation is close to 1, which indicates that the proposed fuzzy AHP technique constitutes a valid fuzzy alternative of the original AHP approach.

5.4. Quality assessment of software products

The basic model was employed to assess the quality of the software products stored in the benchmark repository. The products were ranked in a descending order according to their overall quality score provided by the basic model. Tables 4 and 5 show the first and last 10 products of the aforementioned ranking, respectively.

From the assessment results presented in these tables, one may easily identify that quality assessment is independent of product size. For instance, the products *Apache Commons* and *Javac* receive almost the same quality score even though the latter is approximately 6 times bigger than the former. Moreover, the *Annotations Plugin* receives almost the same rating with *Hamcrest* despite the fact that it consists of only one class and it comprises significantly fewer LOC. Thus, one may argue that QATCH can be used for the evaluation of individual classes, as well as for entire systems, avoiding the major threat to validity of quality assessment models, i.e. to be influenced by the size of the products under evaluation (both in terms of number of classes and of LOC).

5.5. Quality improvement

Another research question answered in our analysis is whether the quality models derived by QATCH can assist in monitoring, and therefore improving software product quality. For this purpose, two case studies were conducted.

5.5.1. Case study 1: auto-generated products evaluation

Within the context of the first case study, a set of software products containing exclusively auto-generated code were constructed. Snippets of code that violate specific (previously known) rules were added to these products. In other words, a logic similar to error seeding, which is widely applied for testing purposes, is adopted. As a result, quality assessment against the basic model

Table 3
Comparison of the weights derived through the application of the AHP and the fuzzy AHP techniques for a specific characteristic.

Property Name	W_{AHP}	$W_{FAHP,C,\sigma=0.0001}$	$W_{FAHP,C}$	$W_{FAHP,D}$	$W_{FAHP,U}$
Bad Function	0.1745	0.1712	0.1712	0.1758	0.1768
Comprehensibility	0.1232	0.1341	0.1341	0.1406	0.1383
Redundancy	0.0610	0.0702	0.0703	0.0672	0.0664
Structuredness	0.0217	0.0298	0.0298	0.0283	0.0282
Assignment	0.0298	0.0371	0.0358	0.0364	0.0388
Resource Handling	0.0390	0.0442	0.0442	0.0495	0.0517
Cohesion	0.0262	0.0317	0.0327	0.0331	0.0340
Coupling	0.0194	0.0217	0.0220	0.0212	0.0228
Complexity	0.0192	0.0244	0.0239	0.0263	0.0314
Messaging	0.3289	0.2852	0.2847	0.2807	0.2685
Encapsulation	0.1569	0.1505	0.1513	0.1408	0.1432
Correlation	-	0.9961299	0.9960122	0.9927116	0.9906083

Table 4
The top 10 benchmark products according to their quality.

Product Name	*LOC*	*Q*
Annotations Plugin	58	0.8149
Hamcrest	48,890	0.79
Apache Commons	24,907	0.7465
Javac	120,726	0.7425
Joda Convert	6542	0.7158
Findbugs	491	0.6954
Persistence	1422	0.6767
Jpa	3286	0.6705
Maven Model	35,684	0.6702
Google Protobuf	78,046	0.6527

Table 5
The last 10 benchmark products according to their quality.

Product Name	*LOC*	*Q*
HyperSQL	361,948	0.3934
Hamcrest Core	2557	0.3765
Bcel	72,470	0.3714
Jackson Data Mapper	69,508	0.3590
Jcommander	6231	0.3489
Osgi	18,326	0.3249
Compentium	2272	0.3220
Dom4j	38,022	0.3217
Jackson	22,035	0.3200
Slf4j	2983	0.2912

Table 6
The results of the quality assessment of different versions of Quality Model Designer according to basic model.

Version (*n*)	*LOC*	*Q*
v0.1	725	0.7541
v0.2	1647	0.6374
v0.3	10,853	0.6538
v0.4	10,853	0.6538
v0.5	10,742	0.6459
v0.6	14,298	0.6492
v0.7	17,354	0.6446
v1.0	18,365	0.6644
v1.1	19,519	0.6654
v1.3	12,930	0.6438
v1.4	12,930	0.7234
v1.5	13,159	0.8115

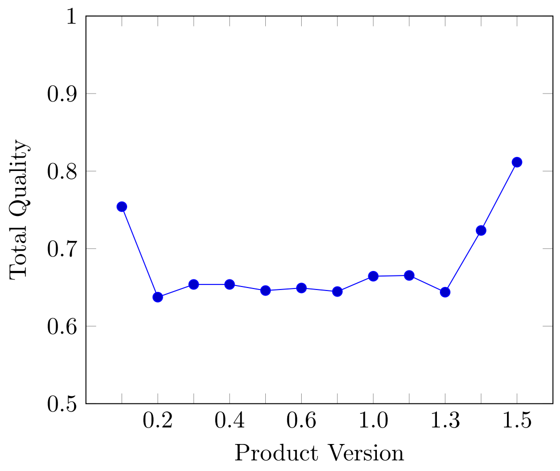


Fig. 12. The evolution of the Quality Model Designer rating throughout its development.

leads to relatively low quality scores. Through inspection of the evaluation results, we realized that the quality model identified all the deliberately added violations, along with other violations caused by the auto-generated code. After solving these issues and re-assessing software, the products received the highest possible rating, which is the value of 1.

The usage of the auto-generated codes has the advantage that we have apriori knowledge of the issues that will arise during the evaluation. This way, a totally controlled environment is created for evaluating the validity of the quality models and, in turn, of the framework as a whole.

From the above, we argue that the quality model derived by QATCH was able to identify all the potential violations that underlie in the software products under evaluation. The derived quality models also support root-cause analysis; one may identify the issues that cause a reduced quality score and take the appropriate corrective actions in order to improve the overall software quality.

5.5.2. Case study 2: real product evaluation

Within the context of the second case study, the basic model was employed for the quality assessment of different versions of the QMD itself. The evaluation results for each QMD version, along with their LOCs, are presented in Table 6. For better understanding the results are also depicted in Fig. 12 in a line graph form.

One may notice that initially the product had relatively high quality (i.e. 0.75). After the addition of the needed components, the quality fell until version v1.3. Version v1.3 is the final version of the product after the deletion of unnecessary classes that were used for testing purposes or as alternative solutions. No corrective actions for quality improvement were taken until this version.

Between v1.3 and v1.4, corrective actions were taken according to the evaluation results provided by the basic model. This led to an increase of the overall quality score to 0.71. Violations concern-

Table 7
The evaluation results produced by the basic model and the rankings provided by the basic model, the Quamoco framework and the experts of "Linzer Software-Verkostung".

Software Product	Product Version	QATCH Quality Score	QATCH Ranking	Quamoco 2015 Ranking	Quamoco 2011 Ranking	Experts Ranking
Checkstyle	5.3	0.57201	1	1	1	1
Log4j	1.2.15	0.47829	2	3	2	2
RSSOwl	2.0	0.47490	3	2	2	3
TV-Browser	2.2.6	0.47466	4	4	4	4
JabRef	2.4	0.45239	5	5	5	5

ing naming conventions and comments structure were fixed (the reason why the LOCs did not change between these two versions). Subsequently, more advanced quality improvement actions, such as refactoring, were performed, leading the overall quality to reach the value of 0.81 at the final version of the product (i.e. version v1.5).

The regular application of quality assessment may help in the identification of quality problems early enough in the software development lifecycle of a software product, when their correction is relatively easy and cheap. Thus, less refactoring will be required in the future. In other words, QATCH helps to the establishment of a Continuous Quality Control paradigm.

5.6. Relation between quality and reputation

The software products of the benchmark repository are ranked in a descending order according to the quality score provided by the basic model. This ranking is compared to the corresponding ranking of the aforementioned products according to their reputation provided by the Maven Repository. For this purpose we measure the Spearmans rank correlation coefficient (r) (Spearman, 1987) between these two rankings. A high positive correlation, which is the desired outcome of this experiment, means close relation between the two rankings.

In our case, the calculated Spearmans rho correlation is $r = -0.2616742$, which is a negative and rather small correlation. This result is statistically significant due to the small p-value, which is 0.008708.

From this result we may conclude that no positive relation exists between the reputation of software products and their quality. Therefore, *high reputation does not necessarily mean high quality*.

5.7. Comparative analysis of quality assessment approaches

The purpose of this experiment is to evaluate whether the assessment results provided by the basic model reflect the real quality of the assessed software products. A positive outcome of this experiment will indicate that the quality models produced by the proposed framework are reliable and can used in practice for the quality evaluation of software products.

For this purpose, a comparison of the QATCH assessment results against competitive approaches is necessary. However, no quality evaluation data of real-world open source or commercial software products are publicly available in the literature, so as to be used for such a comparison. Thus, similarly to (Deissenboeck, Heinemann, Herrmannsdoerfer, Lochmann, & Wagner, 2011) and (Wagner et al., 2015), we decided to base our comparison on quality assessment results of software products offered by experts in the field of software quality. In particular, we used the quality ranking of five open source Java products, provided by nine Java experts during the Linzer Software-Verkostung (Gruber, Plösch, & Saft, 2010) (LSV). The investigated software products were: JabRef, Checkstyle, Log4j, RSSOwl and TV-Browser. In order for our comparison to be as precise as possible, we selected the versions that were closer to the date of the LSV conference (October 2008).

Table 8
The Spearman's rank correlation coefficient (r) between the ranking provided by the basic model and the corresponding rankings provided by the experts and the Quamoco framework respectively.

Ranking of Reference	r	$p-value$
Experts	1	0
Quamoco 2015	0.9	0.0374
Quamoco 2011	0.97	0.0048

Initially, we employed the QATCH basic model for the quality evaluation of the five selected software products, and ranked them based on the produced quality scores. Subsequently, we retrieved the product ranking offered by the experts of the LSV and the one produced by the Quamoco framework. It should be noted that the authors of Quamoco have resulted to two slightly different rankings of the selected products in two different publications (i.e. (Deissenboeck, Heinemann, Herrmannsdoerfer, Lochmann, & Wagner, 2011) and (Wagner et al., 2015)). For completeness reasons we have used both for the comparison. The quality scores of the products, which were calculated by the basic model, and the resulting product ranking, along with the two individual rankings produced by the Quamoco framework and the one offered by the experts are presented in Table 7.

All products are ranked on the exact quality scores assigned by QATCH. It is obvious that QATCH ranking perfectly matches the ranking provided by the experts. This indicates that the QATCH basic model accurately ranks the assessed products with respect to their quality, as it is perceived by the experts in the field.

Additionally, the Spearmans rank correlation coefficient (r) (Spearman, 1987) is employed in order to ensure that the investigated rankings are consistent. A positive and close to one correlation value indicates that the two investigated rankings are highly consistent. Additionally, in order to ensure that a resulting positive correlation is statistically significant, similarly to (Deissenboeck, Wagner, Pizka, Teuchert, & Girard, 2007) and Wagner et al. (2015), we formulate the following hypothesis (and its corresponding null hypothesis):

H_1: *There is a statistically significant positive correlation between the investigated rankings.*

H_0: *There is not a statistically significant positive correlation between the investigated rankings.*

which is tested with confidence level 95% (a=0.05). This hypothesis is tested for the product ranking provided by QATCH against the three individual rankings provided by the Quamoco framework and the experts. The calculated correlations, along with their p-values are presented in Table 8.

From the results presented in Table 8 it is obvious that the correlation between the ranking derived by QATCH and the expert-based ranking receives the highest possible value (i.e. the value of 1), indicating that the assessment results provided perfectly match those provided by the experts. The calculated correlation between the ranking obtained by QATCH and those derived by the assessment results of the Quamoco framework were found to be 0.9 and 0.97, respectively. Thus, there is a high positive correlation between the rankings, which indicates that our ranking of the software products is highly consistent with the corresponding product orderings derived by the Quamoco. The p-values of the calculated correlations were found to be below 0.05 in all three cases, which means that the null hypothesis can be rejected.

Overall, from the results we may conclude that the evaluation results provided by QATCH adequately reflect the experts opinion regarding the quality of the selected software products and is in high accordance with those provided by the Quamoco tool chain. Thus, quality models produced by QATCH may constitute a reliable solution for the evaluation of software product quality.

6. Conclusion

In this paper we introduced QATCH, an integrated framework that allows the derivation of reliable custom quality models that can be used for the evaluation of software products. The framework adopts state-of-the-art approaches, in order to produce readable and understandable three-layered hierarchical quality models. Static analysis and benchmarking is performed for threshold derivation, which makes the assessment process automatic and highly objective. Two individual multi-criteria decision-making techniques are provided by the framework for weights elicitation. In particular, QATCH supports the Analytic Hierarchy Process and proposes its fuzzy alternative in order to model the uncertainties that underlie in experts judgments. This method is further enhanced in order to offer the experts the opportunity to express both their opinions and their uncertainty into linguistic values, making the weights elicitation process easier and more intuitive. Finally, a GUI facilitates the user in performing the quality model derivation process in three clearly defined steps.

QATCH assesses the quality of software products in a holistic manner, based on the international standard, ISO 25010. It offers objectivity and automation in the derivation of thresholds through crowd-sourcing of open source repositories, and transparency in the calculation of the final quality assessment and all the intermediary steps. It is an expandable framework in the sense that metrics from dynamic analyses can be added to the model and contribute to the final outcome. Moreover, QATCH facilitates weight elicitation with techniques accessible to both experts and non-experts in order to easily customize the models to their requirements.

In order to investigate both the validity and the added value of the proposed framework in the field of software quality, a set of experiments were performed. A carefully calibrated quality model, the basic model, was derived, based on widely used Java projects, and was employed for the quality assessment of software products written in the Java programming language. The performed experiments ascertained the proper system operation and the independence of the models with regards to the size of the software product under assessment. By assessing both automatically generated and user-developed software products, the contribution of quality models towards the incorporation of root-cause analysis, and therefore the improvement of software product quality, was highlighted. A comparison of the fuzzy weight generation technique with its deterministic counterpart showed a close correlation between the results of these two methods, leading to the conclusion that the proposed fuzzy AHP technique constitutes a valid fuzzy alternative of the deterministic AHP approach. No correlation was found between the reputation of the benchmark products and their quality score, which led to the conclusion that the reputation is not an indicator of software product quality at least in the investigated case. Finally, QATCH results were evaluated against other competitive approaches and expert evaluations, and were found valid.

Future research and development efforts may include:

- The expansion of the system so that it will be able to assess the quality of software products developed in different programming languages other than Java.
- The extension of the framework in order to support other multi-criteria decision-making techniques for weights elicitation.
- The addition of other static analysis tools.
- The addition of dynamic analysis metrics.
- The adoption of more advanced parallel techniques (such as MPI, CUDA, Map-Reduce etc.) for the acceleration of the static analysis.
- The development of a dashboard for the offline quality evaluation tools that will facilitate the developers in the quality control and monitoring of their software products.

Acknowledgments

This research did not receive any specific grant from any agencies in the public, commercial, or not-for-profit sectors.

References

Al-Jamimi, H. A., & Ahmed, M. (2013). Machine learning-based software quality prediction models: State of the art. In *2013 international conference on information science and applications (icisa)* (pp. 1–4). doi:10.1109/ICISA.2013.6579473.

Alves, T. L., Correia, J. P., & Visser, J. (2011). Benchmark-based aggregation of metrics to ratings. In *Proceedings - joint conference of the 21st international workshop on software measurement, iwsm 2011 and the 6th international conference on software process and product measurement, mensura 2011* (pp. 20–29). doi:10.1109/IWSM-MENSURA.2011.15.

Alves, T. L., Ypma, C., & Visser, J. (2010). Deriving metric thresholds from benchmark data. *Ieee international conference on software maintenance, icsm*. doi:10.1109/ICSM.2010.5609747.

Arar, O., & Ayan, K. (2016). Deriving thresholds of software metrics to predict faults on open source software: Replicated case studies. *Expert Systems with Applications, 61*, 106–121.

Baggen, R., Correia, J. P., Schill, K., & Visser, J. (2012). Standardized code quality benchmarking for improving software maintainability. *Software Quality Journal, 20*(2), 287–307. doi:10.1007/s11219-011-9144-9.

Baldassari, B. (2013). SQuORe: A new approach to software project assessment. *International Conference on Software & Systems Engineering and their Applications*.

Bansiya, J., & Davis, C. (2002). A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering, 28*(1), 4–17. doi:10.1109/32.979986.

Boehm, B., Brown, J., Kaspar, H., Lipow, M., Macleod, G., & Merrit, M. (1976). *Characteristics of software quality*. Amsterdam, Netherlands: North-Holland.

Boender, C., de Graan, J., & Lootsma, F. (1989). Multi-criteria decision analysis with fuzzy pairwise comparisons. *Fuzzy Sets and Systems, 29*(2), 133–143. doi:10.1016/0165-0114(89)90187-5.

Buckley, J. J. (1985). Fuzzy hierarchical analysis. *Fuzzy Sets and Systems, 17*(3), 233–247. doi:10.1016/0165-0114(85)90090-9.

Büyüközkan, G., Kahraman, C., & Ruan, D. (2004). A fuzzy multi-criteria decision approach for software development strategy selection. *International Journal of General Systems, 33*(April 2015), 259–280. doi:10.1080/03081070310001633581.

Chang, C. W., Wu, C. R., & Lin, H. L. (2008). Integrating fuzzy theory and hierarchy concepts to evaluate software quality. *Software Quality Journal, 16*(2), 263–276. doi:10.1007/s11219-007-9035-2.

Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476–493. doi:10.1109/32.295895.

Chiu, N.-H. (2009). An early software-quality classification based on improved grey relational classifier. *Expert Systems with Applications, 34*, 10727–10734.

Chiu, N.-H. (2011). Combining techniques for software quality classification: An integrated decision network approach. *Expert Systems with Applications, 38*, 4618–4625.

Coleman, D. (1992). Assessing maintainability. In *Proceedings 1992 software engineering productivity conference, hewlett-packard* (pp. 525–532).

Deissenboeck, F., Heinemann, L., Herrmannsdoerfer, M., Lochmann, K., & Wagner, S. (2011). The quamoco tool chain for quality modeling and assessment. *2011 33rd International Conference on Software Engineering (ICSE)*, 1007–1009. doi:10.1145/1985793.1985977.

Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., & Girard, J. F. (2007). An activity-based quality model for maintainability. In *Ieee international conference on software maintenance, icsm* (pp. 184–193). doi:10.1109/ICSM.2007.4362631.

Dimiter Driankov, M. R., & Hellendoorn, H. (1996). *An introduction to fuzzy control*. Springer-Verlag Berlin Heidelberg. doi:10.1007/978-3-662-03284-8.

Dromey, R. G. (1995). A model for software product quality. *IEEE Transactions on Software Engineering, 21*(2), 146–162. doi:10.1109/32.345830.

DuBois, D. (1997). *Fuzzy sets and systems: Theory and applications*. Orlando, FL, USA: Academic Press, Inc.

Gruber, H., Plösch, R., & Saft, M. (2010). On the validity of benchmarking for evaluating code quality. *International conferences on software measurement IWSM/MetriKon/Mensura*.

Heitlager, I., Kuipers, T., & Visser, J. (2007). A practical model for measuring maintainability. *6th international conference on the quality of information and communications technology (QUATIC 2007)*, 30–39. doi:10.1109/QUATIC.2007.8.

Hovemeyer, D., & Pugh, W. (2004). Finding bugs is easy. *ACM SIGPLAN Notices, 39*(12), 92. doi:10.1145/1052883.1052895.

ISO/IEC (2001). *ISO/IEC 9126-1: Software engineering – Product quality – Part 1: Quality model*. Geneva, Switzerland.

ISO/IEC (2011). *ISO/IEC 25010: Systems and software engineering – systems and software quality requirements and evaluation (SQuare) – System and software quality models*. Geneva, Switzerland.

Kanellopoulos, Y., Antonellis, P., Antoniou, D., Makris, C., Theodoridis, E., Tjortjis, C., & Tsirakis, N. (2010). Code quality evaluation methodology using the ISO/IEC 9126 standard. *CoRR, abs/1007.5*.

Letouzey, J. L., & Coq, T. (2010). The sqale analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code. In *2010 second international conference on advances in system testing and validation lifecycle* (pp. 43–48). doi:10.1109/VALID.2010.31.

Liang, G.-S., & Wang, M.-J. J. (1991). A fuzzy multi-criteria decision-making method for facility site selection. *International Journal of Production Research, 29*(11), 2313–2330. doi:10.1080/00207549108948085.

Lochmann, K. (2012). A benchmarking-inspired approach to determine threshold values for metrics. *SIGSOFT Software Engineering Notes, 37*(6), 1–8. doi:10.1145/2382756.2382782.

Lochmann, K., & Heinemann, L. (2011). Integrating quality models and static analysis for comprehensive quality assessment. *Proceeding of the 2nd international workshop on emerging trends in software metrics - WETSoM '11*, 5. doi:10.1145/1985374.1985378.

McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering, SE-2*(4), 308–320. doi:10.1109/TSE.1976.233837.

McCall, J., Richards, P., & Walters, G. (1977). *Factors in software quality*. Springfield: National Technical Information Service.

Mordal-Manet, K., Balmas, F., Denier, S., Ducasse, S., Wertz, H., Laval, J., … Vaillergues, P. (2009). The squale model - A practice-based industrial quality model. In *Ieee international conference on software maintenance, ICSM* (pp. 531–534). doi:10.1109/ICSM.2009.5306381.

Saaty, T. L. (2008). Decision making with the analytic hierarchy process. *International Journal of Services Sciences, 1*(1), 83. doi:10.1504/IJSSCI.2008.017590.

Spearman, C. (1987). The proof and measurement of association between two things. By C. Spearman, 1904. *The American Journal of Psychology, 100*(3–4), 441–471. doi:10.1037/h0065390.

Sterling, C. (2010). *Managing software debt: Building for inevitable change* (1st). Addison-Wesley Professional.

Van Laarhoven, P. J. M., & Pedrycz, W. (1983). A fuzzy extension of Saaty's priority theory. *Fuzzy Sets and Systems, 11*(1–3), 199–227. doi:10.1016/S0165-0114(83)80082-7.

Wagner, S., Goeb, A., Heinemann, L., Kläs, M., Lampasona, C., Lochmann, K., … Trendowicz, A. (2015). Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology, 62*, 101–123. doi:10.1016/j.infsof.2015.02.009.

Wagner, S., Lochmann, K., Heinemann, L., Klas, M., Trendowicz, A., Plosch, R., … Streit, J. (2012). The Quamoco product quality modelling and assessment approach. *2012 34th international conference on software engineering (ICSE)*, 1133–1142. doi:10.1109/ICSE.2012.6227106.

Wang, Y. M., & Chin, K. S. (2011). Fuzzy analytic hierarchy process: A logarithmic fuzzy preference programming methodology. *International Journal of Approximate Reasoning, 52*(4), 541–553. doi:10.1016/j.ijar.2010.12.004.

Yuen, K. K. F., & Lau, H. C. W. (2011). A fuzzy group analytical hierarchy process approach for software quality assurance management: Fuzzy logarithmic least squares method. *Expert Systems with Applications, 38*(8), 10292–10302. doi:10.1016/j.eswa.2011.02.057.

Zadeh, L. (1965). Fuzzy sets. *Information and Control, 8*(3), 338–353. doi:10.1016/S0019-9958(65)90241-X.