Task Force Sub Group 3 - Review of Software Quality Attributes and Characteristics

David, Mario (LIP) Colom, Miguel (ENS Paris-Saclay) Garijo, Daniel (UPM)
Castro, Leyla Jael (ZB MED) Louvet, Violaine (CNRS)
Ronchieri, Elisabetta (INFN/CNAF) Torquati, Massimo (UNIPI)
del Caño, Laura (CSIC/CNB) Leong, Cerlane (CSCS)
Van den Bossche, Maxime (KU Leuven) Campos, Isabel (CSIC/IFCA)
Di Cosmo, Roberto (INRIA)

January 30, 2024

Contents

1 Quality Characteristics

A significant notion in software engineering is *quality*, understood as a series of desirable particularities that are met in *good* computer programs, meaning that they are well adapted to their purpose, accurate, highly available, or any other property which makes them excellent.

Whereas this is a concept which is easy to understand, arriving at a consensus on how to properly define it, is a significantly more complex task. Indeed, the problem can be addressed from plenty of different perspectives, each of them giving more or less importance to some particularities.

The present document regards the work performed by *sub-group 3* **Software Quality** in research.

As such, it is deemed important to have a definition of Research Software (**RS** from here on). Thus, the sub-group has taken the following definition [1]:

"Research Software (RS) is commonly used to refer to software used and/or generated in a research context, including and not limited to scientific, non-scientific, commercial, academic and non-academic research."

Another, more thorough definition, comes from the "Source Codes and Softwares College from the Committee for Open Science" in France, it encompasses the previous definition:

"Research software is developed to meet specific scientific needs. It is designed, maintained, and used by scientists (researchers and engineers) and research institutions, possibly on an international scale. It can result from research work as well as support it, notably through publications before/on/around/with the software. It can be formalized in different ways (a platform, a middleware, a workflow or a library, a module or plugin of another software) and thus be in interaction in an ecosystem or on the contrary be more autonomous." [1]

It is not the goal of this report to propose any particular definitions, but to analyse how the problem has been addressed in relevant works in the literature and how the characteristics can be classified as attributes. As such, in Section 1.1 we detail the method used for a survey of the literature.

In Section 1.2 we review what are, according to the literature, the *Quality Characteristics* that can be related to good software. The characteristics can be related to quality from very different points of view (ranging from metrics of the source code to service operability, just as an example). These are classified as *Quality Attributes and Metrics* in Section 2.

It's quite important to define the notion of **Verification and Validation** (**V&V**) of the *Quality Attributes and Metrics*, since this is the cornerstone to prove the correctness and user requirements satisfaction of a given Software or service. The following definitions from 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology [2] apply:

- Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.
- Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of the respective phase.

1.1 Software Quality Models: survey

The classification into characteristics and attributes is based on a thorough review of the literature, in order to avoid biases on the classification and recommendations with the personal preferences or pre-established beliefs of the members of the *sub-group*. We followed the protocol and methodology proposed by Kitchenham and Charters [3] consisting of the following steps:

1. **Source selection and search**: We searched in the SCOPUS database, including the top five journals in software engineering related to software [2] and articles of the "International Conference on Software Engineering", one of the

[1](`https://www.ouvrirlascience.fr/research-software-as-a-pillar-of-open-science/`).
[2]`https://research.com/journals-rankings/computer-science/software-programming`

top venues for software engineering. We also added documents and web resources that the Task Force *sub-group* considered relevant. The search included the keywords *"software quality"* in the title of the publications. The following journals were considered:

- IEEE Transactions on Software Engineering.
- Empirical Software Engineering.
- Journal of Systems and Software.
- Software & Systems Modeling.
- Information and Software Technology.
- IEEE Software.
- Software Quality Journal.

The following SQL query was used to filter out the articles in this step:

```
TITLE ( software  AND quality )  AND
    ( LIMIT-TO ( EXACTSRCTITLE ,  "Software Quality Journal" )
    OR  LIMIT-TO ( EXACTSRCTITLE ,  "Proceedings International Conference On Software Engineering" )
    OR  LIMIT-TO ( EXACTSRCTITLE ,  "IEEE Transactions on Software Engineering" )
    OR  LIMIT-TO ( EXACTSRCTITLE ,  "Empirical Software Engineering" )
    OR  LIMIT-TO ( EXACTSRCTITLE ,  "Journal of Systems and Software" )
    OR  LIMIT-TO ( EXACTSRCTITLE ,  "Software & Systems Modeling" )
    OR  LIMIT-TO ( EXACTSRCTITLE ,  "Information and Software Technology" )
    OR  LIMIT-TO ( EXACTSRCTITLE ,  "IEEE Software" )
    )  AND  ( LIMIT-TO ( SUBJAREA ,  "COMP" )  OR  LIMIT-TO ( SUBJAREA ,  "ENGI" ) )
```

We obtained 272 results with this process. Additional filtering was applied with the following criteria:

- Articles with no abstracts.
- Articles which were simple summaries of already existing proceedings.
- Articles that a preliminary review of the abstract and title made clear that were out of our scope.
- Articles that did not propose any quality dimensions. For example, those papers that just discuss practices.

2. **Inclusion and exclusion criteria**: The most important criterion for not considering articles in our survey was to exclude those which simply did not belong to the software engineering field. We did not attempt to tighten much the criteria for the exclusion, as we intended to gather as much related material as possible to be filtered out in the next field. Another early exclusion criterion was the language; we excluded non-English articles.

3. **Selection procedure**: After the previous selection, 147 papers remained for further analyses by reading their title and abstract. This allowed us to assess the pertinence of the article to the software engineering domain and their relevance to be used as a source for general quality software recommendations, assuming it mentioned software quality attributes. Although the protocol that we followed ensured that no human bias was introduced, it also prevented some very relevant papers related to Quality from being considered. Four additional articles were added in this step [4, 5, 6, 7], after the *sub-group* discussed the issue.

4. **Review process**: After the previous analysis and selection procedure, 19 relevant articles remained. These are the ones which are finally reviewed in our survey. We performed this step by groups of 2 or 3 reviewers per article, in our *sub-group*.

The list of relevant articles obtained by this procedure, allowed us to define a list of characteristics and attributes which are presented in the following sections.

1.2 Software Quality Characteristics

We identified a list of significant quality characteristics mainly from three sources:

- The ISO/IEC 25010:2011(E) standards [8]; is a norm that proposes two models for the characteristics. The first group is related to the context in which the software product is used, and contains five characteristics. The second group has eight characteristics according to characteristics of the software of the computer system itself, without relying on how it is used.
- The ISO/IEC/IEEE 24765:2017 standard [9]; defines a common vocabulary for systems and software engineering, in a way that it is always applicable to general applications. This standardised list provides a vast quantity of very precise definitions that avoid trivial misinterpretations, and that therefore we apply in this report.
- The chapter *Design Fundamentals* of the Microsoft Application Architecture Guide [10]; the chapter *Quality Attributes* from the Microsoft's Application Architecture Guide also helped establish a common terminology in our definitions.

The analysis of the existing literature resulted in a significant number of characteristics which were associated with the concept of quality. All the characteristics were taken into account in our study, but some of them were filtered out since they were not useful to perform the classification in quality attributes. Indeed, some characteristics were pertinent at the time of the publication of the article, but after a few years they became obsolete. For example, producing a very small sized compiled binary *per se* as an indication of quality does not make much sense nowadays, whereas in the past it could be directly related to the ability of storing the program in a very limited memory or permanent media. Other characteristics were discarded because they were very specific to a given domain, such as real-time applications or critical systems. They remain valid characteristics to take into account, but they are not general enough or not applicable to Research Software or useful to the proposed recommendations.

We ended up with a list of 25 significant quality characteristics from the three above mentioned sources. The characteristics are defined as follows:

1. **Accessibility**; degree to which a product or system can be used by people with the widest range of diversity and capabilities to achieve a specified goal in a concrete context of use [8]. The range of capabilities includes diversity associated with age or any functional diversity. Accessibility can be specified or measured either as the extent to which a product or system can be used by users with diverse capabilities to achieve specified goals with effectiveness, efficiency, freedom from risk and satisfaction in a specific context of use, or by the presence of properties which specifically support the product's accessibility.
2. **Attractiveness**; degree to which a user interface enables pleasing and satisfying interaction with the user [8]. For example, one can include here properties of the product or system such as the use of colour and the nature of the graphical design. This characteristic is also known as *interface aesthetics*.
3. **Availability**; degree to which a system, product, or component is operational and accessible when required for use. We adapted this definition from [9].
4. **Confidentiality**; degree to which a product or system ensures that data are accessible only to those who have been authorised access [8]. This characteristic is a Security characteristic specialised on the privacy of the data.
5. **Compatibility**; degree to which a product, system, or component can exchange information with other products, systems or components, and performs its designed functions, sharing the same hardware or software environment [8]. This definition is very close to the one of *Interoperability*. Here we focus on the ability of the software to perform its functions in a suitable environment, whereas in Interoperability we refer to the format of the data which is exchanged.
6. **Ease of use (or *usability*)**; degree to which a product or system can be used by particular users to achieve their own goals with effectiveness, efficiency, and satisfaction in a given context of use. Usability can either be specified or measured as a product quality characteristic in terms of its sub-characteristics, or specified or measured directly by a subset of System/Software Product Quality [8].
7. **Fault tolerance**; degree to which a system, product, or component operates as intended despite the presence of hardware or software faults. We adapted this definition from [9].

8. **Functional suitability**; degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions [8].
9. **Installability**; degree of effectiveness and efficiency a product or system can be successfully added or removed in a specified environment [8].
10. **Interoperability**; degree to which two or more systems, products, or components can share data, encoded in agreed formats. This definition was adapted from [9].
11. **Maintainability**; degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers [8]. Modifications can include corrections, improvements, or adaptation of the software to changes in the environment, and in the requirements and functional specifications. Modifications include both those carried out by specialised support staff, as well as those carried out by business or operational staff or end users. Maintainability can be interpreted as either an inherent capability of the product or system to facilitate maintenance activities, or the quality-in-use experienced by the maintainers for the goal of maintaining the product or system. It includes installation of updates and upgrades.
12. **Modifiability**; degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading the product [8].
13. **Operability** and **Manageability**; degree to which a product or system has attributes that make it easy to operate and control [8]. *Operability* corresponds to *Controllability*, that is, the operator's error tolerance and the conformity with users' expectations.
14. **Performance**; related to how well the system works taking into account the amount of resources used under stated conditions [8]. The *resources* might include other software products, the software and hardware configuration of the system, and any needed supplies (as for example print paper or storage media).
15. **Portability / Adaptability**; degree of effectiveness and efficiency with which a system, product, or component can be transferred from one hardware, software or other operational or usage environment to a different one [8]. Portability can be interpreted as either an inherent capability of the product or system to facilitate porting activities, or the quality-in-use experienced for the goal of porting the product or system.
16. **Recoverability**; degree to which, in the event of an interruption or a failure, a product or system can recover the directly affected data and re-establish the desired state of the system [8]. Following a failure, a computer system will typically be down for some time, which is determined by its *recoverability.*
17. **Reliability**; degree to which a system, product, or component provides specific functions under concrete conditions for a specified period of time. This definition is adapted from [9]. Lack of reliability can sometimes be caused by flaws in the requirements, the design, and the implementation, as well as because of contextual changes.
18. **Resource utilisation**; degree to which the amount and types of resources consumed by a product or system when performing its functions, meet the requirements [8]. Human resources are included in this category.
19. **Reusability**; degree to which a software artefacts (say, code, executable files, or any assets) can be exploited in other systems, or utilised to build other artefacts [8].
20. **Safety**; degree to which a product or system mitigates the potential personal risk to humans or to system components, in the intended contexts of use [8].
21. **Scalability**; this characteristic can be seen from the point of view of the system, or the applications. In the first case it's the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands. Examples would include how well a hardware system performs when the number of users is increased, how well a database withstands growing numbers of queries, or how well an operating system performs on different classes of hardware. Enterprises that are growing rapidly should pay special attention to scalability when evaluating hardware and software [11]. When referring to algorithms, protocols, or applications it can be defined as being able to efficiently handle a growing demand of work or need of more performance by

means of adding more resources to the system on which the software is running. Resources can be added both to single nodes (vertical scalability) and to the system as a whole (horizontal scalability) [12].

22. **Security**; degree to which a product or system protects the information and data it manages (say, stores or transmits) in a way that access is only given to persons or systems with the appropriate level of authorisation they were granted [8]. Related to Security we can also find:
    - **Survivability**; degree to which a product or system continues to fulfil its mission by providing essential services in a timely manner despite the presence of attacks, covered by **Recoverability**.
    - **Immunity**; degree to which a product or system is resistant to certain attacks, covered by **Integrity**.
23. **Supportability**; is it defined as the ability of the system to provide helpful information to identify and resolve issues in case of malfunction [10]. The existence of a helpdesk, issue tracking, bug reporting, or related services also contribute to supportability [5].
24. **Testability**; degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component. Then the tests might be run to determine whether the criteria have been met [8].
25. **Time behaviour**; degree to which the response, processing times, and throughput rates of a product or system meet the requirements when performing its functions [8].

2 Software Quality Attributes and Metrics

The *Software Quality Characteristics* which are classified in the previous section can actually be seen from very different points of view, thus allowing for a different type of classification of characteristics in different classes, which we will refer to as *Quality Attributes and Metrics*.

The survey and procedure described in Section 1.1 allowed us to identify 239 *Quality Attributes and Metrics* that were gathered into a single table. We found that, depending on the source, the same attribute or metric could be found under different names, but with a very similar definition of meaning. Those were merged to prevent duplication.

We ended up with 126 *Quality Attributes and Metrics*, it includes the references to the original articles used. We considered attributes which objectively could be treated as *metrics*. The attributes were subdivided into six categories and, for each metric or attribute, a new codename was proposed in order to have a coherent naming convention throughout the document. The following categories and their codenames are proposed:

- Source Code Metrics (**EOSC-SCMet**): 17 metrics. Metrics related to the source code, such as the number of lines of code or the number of assertions, for example.
- Time and Performance Metrics (**EOSC-TMet**): 11 metrics. Metrics related to time or periods of time. For example, the number of resolved bugs per period of time.
- Qualitative Attributes (**EOSC-Qual**): 27 attributes. Qualitative attributes are obtained in general through surveys to, or some manual analysis by software developers, administrators, or users. In general, these are not possible to automate and are in general subjective.
- DevOps - Software Release and Management Attributes (**EOSC-SWRelMan**): 34 attributes. Largely based on the *DevOps* methodology, they can be automated for Verification & Validation. Although the possibility of having code reviews in the software development process is a manual step.
- DevOps - Testing Attributes (**EOSC-SWTest**): 25 attributes. Again based in DevOps, these are related to software testing and can also be automatically verified. For example, whether integration tests are used in the system.
- Service Operability Attributes (**EOSC-SrvOps**): 12 attributes. They refer to scientific services or platforms in operation. For example, whether the system provides monitoring and accounting services.

Each attribute or metric entry has the following elements, as shown in Figure 1:

- Codename: naming convention proposed by the *sub-group*;
- Name: as found in the source reference;
- Associated characteristics: one or more from subsection 1.2;
- Definition: obtained from aggregating or merging the source references;
- Reference to the source articles.

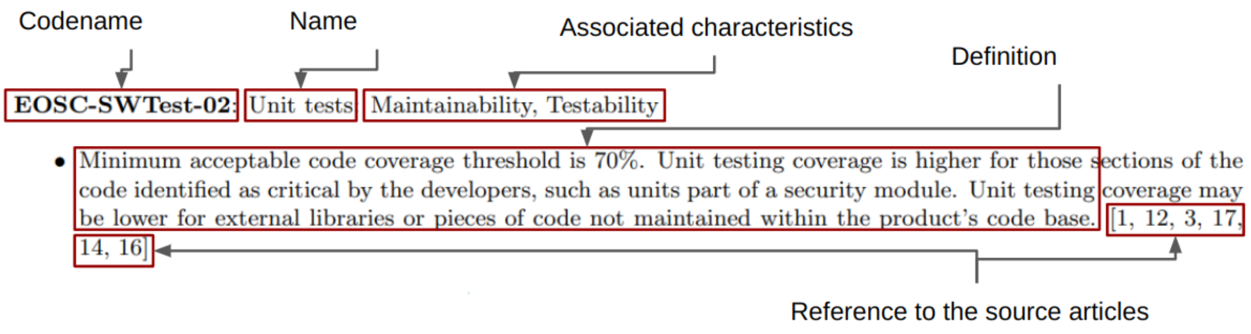
Codename
Name
Associated characteristics
Definition
EOSC-SWTest-02: Unit tests Maintainability, Testability
Minimum acceptable code coverage threshold is 70%. Unit testing coverage is higher for those sections of the code identified as critical by the developers, such as units part of a security module. Unit testing coverage may be lower for external libraries or pieces of code not maintained within the product's code base. [1, 12, 3, 17, 14, 16]
Reference to the source articles

Figure 1: The structure of each Attribute or Metric entry.

2.1 Source Code Metrics: EOSC-SCMet

EOSC-SCMet-01: Size of the software application: Maintainability.

- The software product's rebuild value is estimated from the number of lines of code. This value is calculated in man-years using the Programming Languages Table of the Software Productivity Research [13].

EOSC-SCMet-02: % of redundant code: Maintainability, Modifiability.

- A line of code is considered redundant if it is part of a code fragment (larger than 6 lines of code) that is repeated literally (modulo white-space) in at least one other location in the source code [13].

EOSC-SCMet-03: # Lines of code: Maintainability.

- Number of lines for the whole software project or components / modules / classes / functions / methods [14, 13].

EOSC-SCMet-04: % of assertions: Maintainability.

- Percentage of lines of source code containing assertions. Assertions are used as a means for demonstrating that the program is behaving as expected and as an indication of how thoroughly the source classes have been tested on a per class level [15].

EOSC-SCMet-05: Cyclomatic Complexity: Maintainability.

- Cyclomatic complexity (i.e., the number of linearly independent paths through a program's source code) of the whole software component or modules/components/classes/functions/methods. Cyclomatic complexity is created by calculating the number of different code paths in the flow of the program. A program that has complex control flow requires more tests to achieve good code coverage and is less maintainable [16, 14, 13, 15, 17].

EOSC-SCMet-06: Cyclomatic Complexity test/source ratio: Maintainability.

- Ratio between the sum of cyclomatic complexities of all tests and the source code [15].

EOSC-SCMet-07: Number of arguments: Maintainability.

- Number of arguments or parameters used in functions. Functions with many parameters may be a symptom of bad encapsulation [13, 17].

EOSC-SCMet-08: Number of function calls: Maintainability.

- Measures the inter-dependencies between unique classes through parameters, local variables, return types, method calls, generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration. Good software design dictates that types and methods should have high cohesion and low coupling. High coupling indicates a design that is difficult to reuse and maintain because of its many inter-dependencies on other types [14, 17].

EOSC-SCMet-09: Modularity: Maintainability, Functional suitability.

- Degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components. Also the number of modules [8, 14, 18, 17, 7].

EOSC-SCMet-10: Number of comments: Modifiability.

- Number of lines corresponding to comments for the whole software or per modules/components / classes / functions / methods [16, 14, 17].

EOSC-SCMet-11: Maintainability Index (MI): Maintainability.

- Degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers. The Maintainability Index (MI) calculates an index value between 0 and 100 that represents the relative ease of maintaining the code. A high value means better maintainability. Colour coded ratings can be used to quickly identify trouble spots in the code. A green rating is between 20 and 100 and indicates that the code has good maintainability. A yellow rating is between 10 and 19 and indicates that the code is moderately maintainable. A red rating is a rating between 0 and 9 [8, 14].

EOSC-SCMet-12: Internal cohesion: Maintainability.

- Cohesion describes how related the functions within a single module are. Low cohesion implies that a given module performs tasks which are not very related to each other and hence can create problems as the module becomes large [9, 14].

EOSC-SCMet-13: Test class to source class ratio: Reliability.

- Control measure to counter the confounding effect of class size. The coefficient is calculated by NOTC/NOSC (NOTC = number of test classes, NOSC = number of source classes) [15].

EOSC-SCMet-14: Coupling Between Objects (CBO) ratio: Maintainability.

- Ratio between the CBO in the tests and in the whole source code. The higher the inter-object class coupling, the more rigorous the testing should be [15].

EOSC-SCMet-15: Depth of inheritance Tree (DIT) ratio: Maintainability.

- Ratio between the DIT of the tests and the DIT of the source code. A higher DIT indicates desirable reuse but adds to the complexity of the code because a change or a failure in a super class propagates down the inheritance tree [15].

EOSC-SCMet-16: Weighted Methods per Class (WMC) ratio: Maintainability.

- Ratio between the WMC of the tests with respect to the WMC of the whole source code. This measure serves to compare the testing effort on a method basis. WMC is an indicator of fault-proneness [15].

EOSC-SCMet-17: Source lines of code (SLOC) ratio: Maintainability.

- SLOC of the whole project with respect to the minimum SLOC of its components [15].

2.2 Time and Performance Metrics: EOSC-TMet

EOSC-TMet-01: Effort required for changes: Reliability, Supportability.

- Time and resources dedicated to resolve an issue [14].

EOSC-TMet-02: # Resolved bugs: Supportability.

- Number of resolved bugs per period of time [14].

EOSC-TMet-03: # Open bugs: Supportability.

- Number of open bugs/issues per period of time [14].

EOSC-TMet-04: Defect rates: Maintainability.

- The number of outstanding defects in a product per period of time [19].

EOSC-TMet-05: Integrity: Integrity, Maintainability.

- Resource cost expended to solve problems caused by inconsistencies within the system. This may be measured in terms of staff time employed to fix problems and user time wasted. Also, degree to which a system, product or component prevents unauthorised access to, or modification of, computer programs or data [8, 20].

EOSC-TMet-06: Maintainability: Maintainability.

- Measured by the resources spent in terms of time and cost in keeping a system up and running over a period of time [20, 21].

EOSC-TMet-07: Adaptability: Reusability. Adaptability.

- Degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments [8, 20, 21].

EOSC-TMet-08: # Registered users: Operability.

- The metric Number of registered users is to be collected [5].

EOSC-TMet-09: # Active users: Operability.

- The metric Number of active users over a given period of time may be collected [5].

EOSC-TMet-10: Amount of computing resources: Operability, Performance Efficiency.

- The metric amount of computing resources per user or group of users is collected. An example is CPU x hours, resource utilisation [8, 5].

EOSC-TMet-11: Amount of storage resources: Operability, Performance Efficiency.

- The metric amount of storage resources per user or group of users is collected. An example is GByte x hours. Resource utilisation and capacity [8, 5].

2.3 Qualitative Attributes: EOSC-Qual

EOSC-Qual-01: Complexity of diagrams: Maintainability. Reusability.

- Complexity of diagrams (UML) for the whole software or modules/components [14].

EOSC-Qual-02: Complexity of architecture: Maintainability. Reusability.

- Architecture showing modules and interactions [14, 22].

EOSC-Qual-03: Complexity of a use case: Maintainability. Reusability. Usability.

- Complexity of use case diagrams (UML) [14].

EOSC-Qual-04: Sustainable community: Supportability.

- An active community is behind the software product [18].

EOSC-Qual-05: User satisfaction: Attractiveness.

- Degree to which a user interface enables pleasing and satisfying interaction for the user [8, 22].

EOSC-Qual-06: Usability: Usability.

- Measured using user surveys. However, it may also be assessed in terms of calls upon support staff, e.g. number of requests for help or support staff time expended. Also, degree to which a product or system can be used by specified users to achieve specific goals with effectiveness, efficiency and satisfaction in a specified context of use [8, 22, 20, 21].

EOSC-Qual-07: Reliability: Reliability.

- The reliability of systems from the user's point of view is concerned with three things: (1) How often does it go wrong? (2) How long is it unavailable? (3) Is any information lost at recovery? Also, degree to which a system, product or component performs specific functions under specified conditions for a specified period of time [8, 20, 21].

EOSC-Qual-08: Timeliness: Supportability.

- Assessed in terms of the costs of non-delivery. These will possibly include staff time and lost sales. It may also be assessed in terms of the number of days departure from the date agreed with client [20].

EOSC-Qual-09: Cost-Benefit efficiency: Maintainability.

- Measured in simple financial terms. The costs of installing and maintaining the system are weighed against the assessment of business benefits [20].

EOSC-Qual-10: Accessibility: Technical accessibility.

- Degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use [8, 21].

EOSC-Qual-11: Accountability: Performance, Resource utilisation.

- Degree to which the actions of an entity can be traced uniquely to the entity. Its usage can be measured; critical segments of code can be instrumented with probes to measure timing, whether specified branches are exercised, etc. [8, 21].

EOSC-Qual-12: Accuracy: Performance, Functional suitability.

- Code outputs are sufficiently precise to satisfy their intended use [21].

EOSC-Qual-13: Communicativeness: Ease of use.

- Metric which facilitates the specification of inputs and provides outputs whose form and content are easy to assimilate and useful [21, 6].

EOSC-Qual-14: Completeness: Supportability, Manageability.

- All software parts are present and each part is fully developed [8, 21].

EOSC-Qual-15: Conciseness: Supportability, Resource utilisation.

- Redundant information about the software code is not present [21].

EOSC-Qual-16: Consistency: Maintainability, Interoperability, Compatibility.

- Source code contains uniform notation, terminology and symbology within itself [21, 6].

EOSC-Qual-17: Legibility: Supportability, Maintainability.

- Software function is easily discerned by reading the code [21].

EOSC-Qual-18: Modifiability: Modifiability.

- Degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality [8, 21].

EOSC-Qual-19: Robustness: Safety.

- Software can continue to perform despite some violation of the assumptions in its specification [21].

EOSC-Qual-20: Self-containedness: Supportability.

- Software performs all its explicit and implicit functions within itself [21].

EOSC-Qual-21: Self-descriptiveness: Supportability, Ease of use.

- Software contains enough information for a reader to determine or verify its objectives, assumptions, constraints, inputs, outputs, components, and revision status [21].

EOSC-Qual-22: Structuredness: Modifiability, Reusability.

- Software possesses a definite pattern of organisation of its interdependent parts [21].

EOSC-Qual-23: Understandability: Supportability.

- Software purpose is clear to the inspector [21].

EOSC-Qual-24: Intellectual Property: Supportability, Reusability.

- There are multiple statements embedded into the software product describing unrestricted rights and any conditions for use, including commercial and non-commercial use, and the recommended citation. The list of developers is embedded in the source code of the product, in the documentation, and in the expression of the software upon execution. The intellectual property rights statements are expressed in legal language, machine-readable code, and in concise statements in language that can be understood by laypersons, such as a pre-written, recognisable licence [7].

EOSC-Qual-25: Extensibility: Attractiveness, Modifiability.

- There is evidence that the software project has been extended externally by users outside of the original development group using existing documentation only. There is a clear approach for modifying and extending features across and in multiple scenarios, with specific documentation and features to allow the building of extensions which are used across a range of domains by multiple user groups. There may be a library available of user-generated content for extensions and user generated documentation on extension is also available [7].

EOSC-Qual-26: Standards compliance: Functional suitability.

- Compliance with open or internationally recognised standards for the software and software development process, is evident and documented, and verified through testing of all components. Ideally independent verification is documented through regular testing and certification from an independent group [7].

EOSC-Qual-27: Internationalisation and localisation: Usability.

- Demonstrable usability: Software has been tested with multiple pseudo or genuine translations [7].

2.4 DevOps-SW Release and Management Attributes: EOSC-SWRelMan

EOSC-SWRelMan-01: Open source : Supportability, Maintainability, Availability, Reusability.

- Following the open-source model, the source code being produced is open and publicly available to promote the adoption and augment the visibility of the software developments [4, 6].

EOSC-SWRelMan-02: Version Control System (VCS): Supportability, Maintainability.

- Source code uses a Version Control System (VCS). All software components delivered by the same project agree on a common VCS [4].

EOSC-SWRelMan-03: Source code hosting: Supportability, Maintainability.

- Source code produced within the scope of a broader development project resides in a common organisation of a version control repository hosting service [4].

EOSC-SWRelMan-04: Working state version: Maintainability.

- The main branch in the source code repository maintains a working state version of the software component. Main branch is protected to disallow force pushing, thus preventing untested and unreviewed source code from entering the production-ready version. New features are merged in the main branch whenever the agreed upon SQA criteria is fulfilled [4].

EOSC-SWRelMan-05: Changes branches: Maintainability.

- New changes in the source code are placed in individual branches. Branches follow a common nomenclature, usually by prefixing, to differentiate change types (e.g. feature, release, fix) [4].

EOSC-SWRelMan-06: Good patching practice: Maintainability.

- Secondary long-term branch that contains the changes for the next software release exists. Next release changes come from the individual branches. Once ready for release, changes in the secondary long-term branch are merged into the main branch and versioned. At that point in time, main and secondary branches are aligned. This step marks a production release [4, 6].

EOSC-SWRelMan-07: Support: Maintainability, Operability.

- Existence of an issue tracking system or helpdesk, facilitates structured software development. Leveraging issues to track down both new enhancements and defects (bugs, documentation typos). Applies as well to services to report operational and user issues [16, 7, 4, 5].

EOSC-SWRelMan-08: Code review: Maintainability.

- Code reviews are done in the agreed peer review tool within the project, with the following functionality: (a) Allows general and specific comments on the line or lines that need to be reviewed. (b) Shows the results of the required change-based test executions. (c) Allows to prevent merges of the candidate change whenever not all the required tests are successful. Exceptions to this rule cover the third-party or upstream contributions which MAY use the existing mechanisms or tools for code review provided by the target software project. This exception is only allowed whenever the external revision life cycle does not interfere with the project deadlines [16, 18, 22, 4].

EOSC-SWRelMan-09: Semantic Versioning : Maintainability.

- Semantic Versioning specification is followed for tagging the production releases [4, 6].

EOSC-SWRelMan-10: Open-source licence: Supportability, Maintainability, Reusability.

- As open-source software, source code adheres to an open-source licence to be freely used, modified and distributed by others. Non-licensed software is exclusive copyright by default [4, 6].

EOSC-SWRelMan-11: Metadata: Supportability, Maintainability, Availability.

- A metadata file (such as Codemeta or Citation File Format) exists alongside the code, under its VCS. The metadata file is updated when needed, as is the case of a new version [4].

EOSC-SWRelMan-12: Packaging: Installability.

- Production-ready code is built as an artefact that can be installed on a system [7, 4, 6].

EOSC-SWRelMan-13: Register/publish artefact: Installability.

- The built artefact is uploaded and registered into a public repository of such artefacts [4].

EOSC-SWRelMan-14: Notification upon registration: Installability.

- Upon success of the package delivery process, a notification is sent to predefined parties such as the main developer or team [4].

EOSC-SWRelMan-15: Code deployment: Installability.

- Production-ready code is deployed as a workable system with the minimal user or system administrator interaction leveraging software configuration management (SCM) tools [4].

EOSC-SWRelMan-16: Software Configuration Management (SCM) as code: Installability.

- A software configuration management (SCM) module is treated as code. Version controlled, it resides in a different repository than the source code to facilitate the distribution [4].

EOSC-SWRelMan-17: SCM tool: Installability.

- All software components delivered by the same project agree on a common SCM tool. However, software products are not restricted to provide a unique solution for the automated deployment [4].

EOSC-SWRelMan-18: SCM code changes: Installability.

- Any change affecting the applications deployment or operation is subsequently reflected in the relevant SCM modules [4].

EOSC-SWRelMan-19: SCM official repositories: Installability.

- Official repositories provided by the manufacturer are used to host the SCM modules, thus augmenting the visibility and promote external collaboration [4].

EOSC-SWRelMan-20: Installability: Installability.

- Degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment. Also, a production-ready SW or service is deployed as a workable system with the minimal user or system administrator interaction leveraging Infrastructure as Code templates [8, 5].

EOSC-SWRelMan-21: Preserve immutable infrastructures: Installability.

- Any future change to a deployed Service is done in the form of a new deployment, in order to preserve immutable infrastructures [5].

EOSC-SWRelMan-22: Infrastructure as Code (IaC) validation: Installability.

- IaC is validated by specific (unit) testing frameworks for every change being done [5].

EOSC-SWRelMan-23: Packaging of tarballs: Installability.

- Tarballs are not included in the distribution directory. Packaged tarballs are not extracted in the distribution directory [6].

EOSC-SWRelMan-24: Design for upgradability: Compatibility.

- Installation layout supports different versions and code releases [6].

EOSC-SWRelMan-25: Provide checksums: Maintainability.

- Code releases provide checksums with each binary (tarballs, RPMs, etc.) [6].

EOSC-SWRelMan-26: Documentation version controlled: Supportability.

- Documentation is version controlled. [5].

EOSC-SWRelMan-27: Documentation as code: Supportability, Maintainability, Reusability.

- Documentation is treated as code and resides in the same repository where the source code lies [4].

EOSC-SWRelMan-28: Documentation formats: Supportability, Maintainability, Reusability.

- Documentation uses plain text format using a markup language, such as Markdown or reStructuredText. All software components delivered by the same project agree on a common markup language [4, 6].

EOSC-SWRelMan-29: Documentation online: Supportability, Availability.

- Documentation is online available in a documentation repository. Documentation is rendered automatically [4, 5].

EOSC-SWRelMan-30: Documentation updates: Supportability, Maintainability, Reusability.

- Documentation is updated on new software or service versions involving any substantial or minimal change in the behaviour of the application [4, 5, 6].

EOSC-SWRelMan-31: Documentation updates if inaccurate/unclear: Supportability, Maintainability, Reusability.

- Documentation is updated whenever reported as inaccurate or unclear [4, 5].

EOSC-SWRelMan-32: Documentation production: Supportability, Maintainability, Reusability.

- Documentation is produced according to the target audience, varying according to the software component specification. The identified types of documentation and their content are README file(one-paragraph description of the application, a "Getting Started" step-by-step description on how to get a development environment running, automated test execution how-to, links to external documentation below, contributing code of conduct, versioning specification, author list and contacts, licence information and acknowledgements), Developer documentations (Private API documentation, structure and interfaces and build documentation), Deployment and Administration documentations (Installation and configuration guides, service reference card, FAQs and troubleshooting) and user documentations (Public API documentation and command-line reference) [18, 7, 4, 5, 6].

EOSC-SWRelMan-33: Documentation PID: Supportability.

- Documentation has a Persistent Identifier (PID) [5].

EOSC-SWRelMan-34: Documentation licence: Supportability.

- Documentation has a non-software licence [5].

2.5 DevOps - Testing Attributes: EOSC-SWTest

EOSC-SWTest-01: Code style: Maintainability, Testability.

- Each individual software complies with a de-facto code style standard for all the programming languages used in the codebase. Multiple standard style compliance is possible [4, 6].

EOSC-SWTest-02: Unit tests: Maintainability, Testability.

- Minimum acceptable code coverage threshold is 70%. Unit testing coverage is higher for those sections of the code identified as critical by the developers, such as units part of a security module. Unit testing coverage may be lower for external libraries or pieces of code not maintained within the product's code base [18, 15, 21, 7, 4, 6].

EOSC-SWTest-03: Test doubles: Functional suitability, Testability.

- When working on automated testing, Test Doubles (i.e., objects or procedures that look and behave like their release-intended counterparts, but are actually simplified versions that reduce the complexity and facilitate testing) are used to mimic a simplistic behaviour of objects and procedures [4, 5].

EOSC-SWTest-04: Test-Driven Development (TDD): Functional suitability, Maintainability, Testability.

- Software requirements are converted to test cases, and these test cases are checked automatically. Also, degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met [8, 19, 22, 4].

EOSC-SWTest-05: API testing: Functional suitability, Testability.

- API testing MUST cover the validation of the features outlined in the specification (aka contract testing) [5].

EOSC-SWTest-06: Integration testing: Functional suitability, Testability, Interoperability.

- Whenever a new functionality is involved, integration testing guarantees the operation of any previously-working interaction with external services [8, 5].

EOSC-SWTest-07: Functional testing: Functional suitability, Testability.

- Functional testing covers the full scope -e.g. positive, negative, edge cases- for the set of functionality that the software or service claims to provide [8, 5].

EOSC-SWTest-08: Performance testing: Functional suitability, Testability.

- Performance testing is carried out to check the Software Application performance under varying loads [8, 5].

EOSC-SWTest-09: Stress testing: Functional suitability, Testability.

- Stress testing is carried out to check the Service to determine the behavioural limits under sudden increased load [5].

EOSC-SWTest-10: Scalability testing: Functional suitability, Testability.

- Scalability testing is carried out to check the Service ability to scale up and/or scale out when its load reaches the limits [5].

EOSC-SWTest-11: Elasticity testing: Functional suitability, Testability.

- Elasticity testing is carried out to check the Service ability to scale out or scale in, depending on its demand or workload [5].

EOSC-SWTest-12: Open Web Application Security Project (OWASP): Security.

- Application is compliant with Open Web Application Security Project (OWASP) secure coding guidelines [4].

EOSC-SWTest-13: Static Application Security Testing (SAST): Security.

- Source code uses automated linter tools to perform static application security testing (SAST) that flag common suspicious constructs that may cause a bug or lead to a security risk (e.g. inconsistent data structure sizes or unused resources) [4].

EOSC-SWTest-14: Security code reviews: Security.

- Security code reviews for certain vulnerabilities is done as part of the identification of potential security flaws in the code. Inputs come from automated linters and manual penetration testing results [20, 7, 4].

EOSC-SWTest-15: No world-writable files or directories: Security.

- World-writable files or directories are not present in the product's configuration or logging locations [4].

EOSC-SWTest-16: Public endpoints and APIs encrypted: Security.

- The Service public endpoints and APIs are secured with data encryption [5].

EOSC-SWTest-17: Strong ciphers: Security.

- The Service uses strong ciphers for data encryption [5].

EOSC-SWTest-18: Authentication and Authorisation: Security, Technical accessibility.

- The Service has an authentication and authorisation mechanism. The Service validates the credentials and signatures [7, 5].

EOSC-SWTest-19: API security assessment: Security.

- API testing includes the assessment of the security-related criteria [5].

EOSC-SWTest-20: Service compliance with data regulations (GDPR): Security.

- The Service handles personal data in compliance with the applicable regulations, such as the General Data Protection Regulation (GDPR) within the European boundaries [5].

EOSC-SWTest-21: Dynamic Application Security Testing (DAST): Security.

- DAST checks are executed, through the use of ad-hoc tools, directly to an operational Service in order to uncover runtime security vulnerabilities and any other environment-related issues (e.g. SQL injection, cross-site scripting or DDOS). The latest release of OWASP's Web Security Testing Guide and the NIST's Technical Guide to Information Security Testing and Assessment are considered for carrying out comprehensive Service security testing [5].

EOSC-SWTest-22: Interactive Application Security Testing (IAST): Security.

- Interactive Application Security Testing (IAST), analyses code for security vulnerabilities while the app is run by an automated test. IAST is performed to a service in an operating state [5].

EOSC-SWTest-23: Security penetration testing: Security.

- Penetration testing (manual or automated) is part of the application security verification effort [5].

EOSC-SWTest-24: Security assessment: Security.

- The security assessment of the target system configuration is particularly important to reduce the risk of security attacks. The benchmarks delivered by the Centre for Internet Security (CIS) and the NIST's Security Assurance Requirements for Linux Application Container Deployments are considered for this task. Also, the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorisation [8, 5].

EOSC-SWTest-25: Security as Code (SaC) Testing: Security.

- IaC testing covers the security auditing of the IaC templates (SaC) in order to assure the deployment of secured Services. For all the third-party dependencies used in the IaC templates (including all kinds of software artefacts, such as Linux packages or container-based images) [5].

2.6 Service Operability Attributes: EOSC-SrvOps

EOSC-SrvOps-01: Acceptable Usage Policy (AUP): Supportability.

- An Acceptable Usage Policy (AUP) is declared. AUP Is a set of rules applied by the owner, creator or administrator of a network, Service or system, that restrict the ways in which the network, Service or system may be used and sets guidelines as to how it should be used. The AUP can also be referred to as: Acceptable Use Policy or Fair Use Policy [5].

EOSC-SrvOps-02: Access Policy and Terms of Use: Supportability.

- An Access Policy or Terms of Use (APTU) is declared. APTU represents a binding legal contract between the users (and/or customers), and the Provider of the Service. The Access Policy mandates the users (and/or customers) access to and the use of the Provider's Service [5].

EOSC-SrvOps-03: Privacy policy: Supportability.

- A Privacy Policy is declared, with a data privacy statement informing the users (and/or customers), about which personal data is collected and processed when they use and interact with the Service. It states which rights the users (and/or customers) have regarding the processing of their data [5].

EOSC-SrvOps-04: Operational Level Agreement (OLA): Supportability.

- The Service includes an Operational Level Agreement (OLA) with the infrastructure where it is integrated [5].

EOSC-SrvOps-05: Service Level Agreement (SLA): Supportability.

- The Service includes Service Level Agreement (SLA) with user communities [5].

EOSC-SrvOps-06: Monitoring service public endpoints: Availability, Reliability.

- The Service public endpoints are monitored, such as probes measuring the http or https response time [5].

EOSC-SrvOps-07: Monitoring service public APIs: Availability, Reliability.

- The Service public APIs are monitored through the use of functional tests [5].

EOSC-SrvOps-08: Monitoring service Web Interface: Availability, Reliability.

- The Service Web interface is monitored, this can be accomplished through the use of automated and periodic functional tests [5].

EOSC-SrvOps-09: Monitoring security public endpoints and APIs: Availability, Reliability.

- The Service is monitored for public endpoints and APIs secured and strong ciphers for encryption [5].

EOSC-SrvOps-10: Monitoring security DAST: Availability, Reliability.

- The Service is monitored with DAST security checks [5].

EOSC-SrvOps-11: Infrastructure monitoring: Availability, Reliability.

- The Service is monitored for infrastructure-related criteria [5].

EOSC-SrvOps-12: Monitoring with Unit tests: Availability, Reliability.

- IaC (unit) tests are reused as monitoring tests, thus avoiding duplication [5].

References

[1] Morane Gruenpeter et al. *Defining Research Software: a controversial discussion.* Zenodo, Sept. 13, 2021. DOI: 10.5281/zenodo.5504016. URL: https://zenodo.org/record/5504016 (visited on 01/28/2022).

[2] "IEEE Standard Glossary of Software Engineering Terminology". In: *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[3] B. Kitchenham and S. Charters. *Guidelines for performing systematic literature reviews in software engineering.* 2007.

[4] Pablo Orviz et al. "A set of common software quality assurance baseline criteria for research projects". In: (2017). In collab. with Digital.CSIC and Digital.CSIC. DOI: 10.20350/DIGITALCSIC/12543. URL: https://digital.csic.es/handle/10261/160086 (visited on 02/10/2022).

[5] Pablo Orviz Fernández et al. "EOSC-synergy: A set of Common Service Quality Assurance Baseline Criteria for Research Projects". In: (June 15, 2020). In collab. with Digital.CSIC and Digital.CSIC. DOI: 10.20350/DIGITALCSIC/12533. URL: https://digital.csic.es/handle/10261/214441 (visited on 02/10/2022).

[6] Eric Steven Raymond. *Software Release Practice HOWTO.* Jan. 14, 2013. URL: https://tldp.org/HOWTO/Software-Release-Practice-HOWTO/.

[7] John Shepherdson. "CESSDA Software Maturity Levels". In: (Mar. 29, 2019). DOI: 10.5281/zenodo.2614050. URL: https://zenodo.org/record/2614050 (visited on 02/10/2022).

[8] ISO Central Secretary. *Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models: ISO/IEC 25010:2011.* ISO. 2017. URL: https://www.iso.org/standard/35733.html (visited on 06/24/2022).

[9] "ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary". In: *ISO/IEC/IEEE 24765:2017(E)* (2017), pp. 1–541. DOI: 10.1109/IEEESTD.2017.8016712.

[10] Microsoft. *Design Fundamentals.* 2010. URL: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ee658094(v=pandp.10) (visited on 2010).

[11] Gartner. *Gartner Glossary: Scalability.* 2023. URL: https://www.gartner.com/en/information-technology/glossary/scalability (visited on 2021).

[12] André B. Bondi. "Characteristics of Scalability and Their Impact on Performance". In: *Proceedings of the 2nd International Workshop on Software and Performance.* WOSP '00. Ottawa, Ontario, Canada: Association for Computing Machinery, 2000, pp. 195–203. ISBN: 158113195X. DOI: 10.1145/350391.350432. URL: https://doi.org/10.1145/350391.350432.

[13] Robert Baggen et al. "Standardized code quality benchmarking for improving software maintainability". In: *Software Quality Journal* 20.2 (June 2012), pp. 287–307. ISSN: 0963-9314, 1573-1367. DOI: 10.1007/s11219-011-9144-9. URL: http://link.springer.com/10.1007/s11219-011-9144-9 (visited on 06/24/2022).

[14] Sonia Montagud, Silvia Abrahão, and Emilio Insfran. "A systematic review of quality attributes and measures for software product lines". In: *Software Quality Journal* 20.3 (Sept. 2012), pp. 425–486. ISSN: 0963-9314, 1573-1367. DOI: 10.1007/s11219-011-9146-7. URL: http://link.springer.com/10.1007/s11219-011-9146-7 (visited on 06/24/2022).

[15] Nachiappan Nagappan et al. "Early estimation of software quality using in-process testing metrics: a controlled case study". In: *Proceedings of the third workshop on Software quality - 3-WoSQ.* the third workshop. St. Louis, Missouri: ACM Press, 2005, p. 1. ISBN: 978-1-59593-122-1. DOI: 10.1145/1083292.1083304. URL: http://dl.acm.org/citation.cfm?doid=1083292.1083304 (visited on 06/24/2022).

[16] Kamonphop Srisopha and Reem Alfayez. "Software quality through the eyes of the end-user and static analysis tools: a study on Android OSS applications". In: *Proceedings of the 1st International Workshop on Software Qualities and Their Dependencies.* ICSE '18: 40th International Conference on Software Engineering. Gothenburg Sweden: ACM, May 28, 2018, pp. 1–4. ISBN: 978-1-4503-5737-1. DOI: 10.1145/3194095.3194096. URL: https://dl.acm.org/doi/10.1145/3194095.3194096 (visited on 06/24/2022).

[17] H. Ogasawara, A. Yamada, and M. Kojo. "Experiences of software quality management using metrics through the life-cycle". In: *Proceedings of IEEE 18th International Conference on Software Engineering.* Berlin, Germany:

IEEE Comput. Soc. Press, 1996, pp. 179–188. ISBN: 978-0-8186-7247-7. DOI: 10.1109/ICSE.1996.493414. URL: http://ieeexplore.ieee.org/document/493414/ (visited on 07/28/2023).

[18] Mark Aberdour. “Achieving Quality in Open-Source Software”. In: *IEEE Software* 24.1 (Jan. 2007), pp. 58–64. ISSN: 0740-7459. DOI: 10.1109/MS.2007.2. URL: http://ieeexplore.ieee.org/document/4052554/ (visited on 06/24/2022).

[19] Lisa Crispin. “Driving Software Quality: How Test-Driven Development Impacts Software Quality”. In: *IEEE Software* 23.6 (Nov. 2006), pp. 70–71. ISSN: 0740-7459. DOI: 10.1109/MS.2006.157. URL: http://ieeexplore.ieee.org/document/4012627/ (visited on 06/24/2022).

[20] Alan Gillies. “Modelling software quality in the commercial environment”. In: *Software Quality Journal* 1.3 (Sept. 1992), pp. 175–191. ISSN: 0963-9314, 1573-1367. DOI: 10.1007/BF01720924. URL: http://link.springer.com/10.1007/BF01720924 (visited on 06/24/2022).

[21] B.W. Boehm, J.R. Brown, and M. Lipow. “Quantitative evaluation of software quality”. In: *Proceedings - 2nd International Conference on Software Engineering, ICSE 1976*. San Francisco; United States, Oct. 13, 1976, pp. 592–605.

[22] Wolfgang Zuser, Stefan Heil, and Thomas Grechenig. “Software quality development and assurance in RUP, MSF and XP: a comparative study”. In: *Proceedings of the third workshop on Software quality - 3-WoSQ*. the third workshop. St. Louis, Missouri: ACM Press, 2005, p. 1. ISBN: 978-1-59593-122-1. DOI: 10.1145/1083292.1083300. URL: http://dl.acm.org/citation.cfm?doid=1083292.1083300 (visited on 06/24/2022).