Software Improvement Group

SIG/TÜVIT EVALUATION CRITERIA TRUSTED PRODUCT MAINTAINABILITY

Version 13.0 (May 26, 2021)

SOFTWAREIMPROVEMENTGROUP.COM

Colophon

This document was reviewed and approved by the following people:

Reinier Vis	Dennis Bijlsma	Haiyun Xu
Head of evaluation laboratory	Head of Evaluators	Quality Manager
r.vis@sig.eu	d.bijlsma@sig.eu	h.xu@sig.eu

Confidential

TABLE OF CONTENTS

1. INTRODUCTION

Software product quality is a major concern for those that develop, maintain, enhance, acquire, or use software products or software-intensive systems. Technical quality of software products pertains to the ease and speed by which the software allows itself to be modified to keep pace with the changing needs of its users or other stakeholders.

This document specifies *the SIG/TÜViT Evaluation Criteria* for the quality mark:

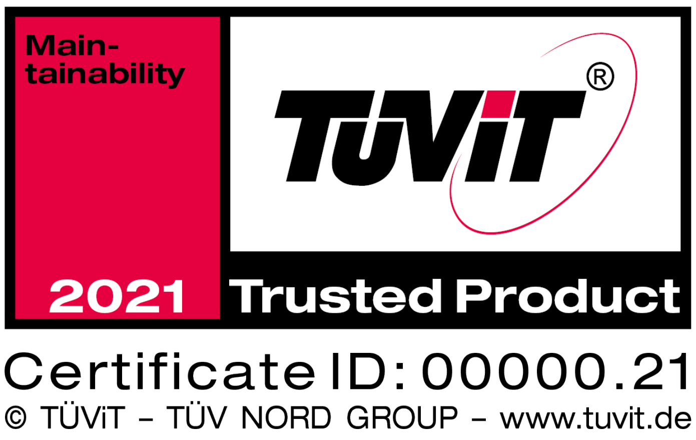

Figure 1 TÜViT Trusted Product Maintainability

Some quality aspects pertain to the current functions of the software and are primarily of interest to the users of software products. Other quality aspects pertain to the ease and speed by which the software allows itself to be modified to keep pace with the changing needs of its users. The latter aspects concern the *technical* rather than the *functional* quality of software products.

The *SIG/TÜViT Evaluation Criteria* for the quality mark *TÜViT Trusted Product Maintainability* are intended for the standardized evaluation and certification of the technical quality of the source code of software products. The purpose of such evaluation and certification is to provide an instrument to developers for guiding improvement of the products they create and enhance, and to acquirers for comparing, selecting, and accepting pre-developed software.

The scope of these *Evaluation Criteria* is limited to the internal quality characteristic of *maintainability* and its sub-characteristics of *analyzability, modifiability, testability, modularity* and *reusability*. The evaluation concerns the source code of a software product, not the *behaviour* of the product in a test or production environment. The criteria can be used in combination with criteria for other notions of software quality, such as the quality of software processes, the quality of software professionals, or the quality of software installations.

The assessment of the quality characteristic of maintainability and its sub-characteristics is based on the assessment of software product properties by source code analysis. These product properties are *volume, duplication, unit complexity, unit size, unit interfacing, module coupling, component balance, component independence, and component entanglement*. The numerical thresholds used in aggregation and rating are calibrated against a large benchmark repository of software product measurements.

The *Evaluation Criteria* define 5 increasing quality levels for maintainability represented by one to five stars, as outlined in Chapter 5. A certificate together with the quality mark *TÜViT Trusted Product Maintainability* can be issued for software products having successfully passed the evaluation based on these criteria and reaching an overall quality level of at least three stars.

1.1 CHANGES WITH RESPECT TO THE PREVIOUS VERSION

There are no significant changes.

2. EVALUATION SCOPE

The *SIG/TÜViT Evaluation Criteria Trusted Product Maintainability* are intended for the standardized evaluation and certification of the technical quality of the source code of software products.

In general, the following kinds of software artefacts are considered for evaluation:

- Source code files (programs, scripts, web pages)
- Generated code files undergoing manual modification after generation
- Data definition files (data schemas)
- Data manipulation files (data queries)
- The active content of web pages

In addition to the software artefacts, a description of the product and its top-level components (their number, their names, their mapping onto software artefacts, and a description of their role in the product) shall be provided for evaluation.

Examples of artefacts that do not qualify as source code and are not considered for evaluation are the following:

- Data files
- Generated code files, other than those manually modified after generation
- Code that is maintained by a third party (e.g. third-party libraries)
- Binary files (e.g. executables, object code, images)
- Configuration files (e.g. metadata files, property files)
- Database migration files
- End-user documentation (e.g. manual pages)
- Administrator or operator documentation (e.g. installation instructions)

Note that the scope of these evaluation criteria does not include any kind of software quality other than product quality. For example, quality of software processes (e.g. development process, maintenance process, testing process, acquisition process) or quality of software professionals (e.g. knowledge of staff about software technologies and or their practical skills) are not included in the evaluation scope.

3. EVALUATION AREAS

The *SIG/TÜViT Evaluation Criteria Trusted Product Maintainability* follow the concept of the ISO/IEC 25010 international standard for software product quality [ISO 25010]. This standard elaborates the notion of software product quality into characteristics and sub-characteristics, and provides recommendations for association of metrics to these quality (sub-)characteristics.

3.1 QUALITY CHARACTERISTICS

The *SIG/TÜViT Evaluation Criteria* are targeted at the *technical quality* of the source code of software products, where technical quality means the *internal* quality characteristic of *maintainability* and its sub-characteristics according to the ISO/IEC 25010 international standard [ISO 25010]. These sub-characteristics are *analyzability, modifiability, testability, modularity* and *reusability*.

- **Maintainability:** "The degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers", where modifications can include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. It also includes installation of updates and upgrades. It 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 [ISO 25010, §4.2.7].

- **Analyzability:** "The degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified" [ISO 25010, §4.2.7.3].

- **Modifiability:** "The degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality" [ISO 25010, §4.2.7.4].

- **Testability:** "The 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" [ISO 25010, §4.2.7.5].

- **Modularity:** "The 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" [ISO 25010, §4.2.7.1].

- **Reusability:** "The degree to which an asset can be used in more than one system, or in building other assets" [ISO 25010, §4.2.7.2].

In terms of the quality perspectives distinguished by ISO/IEC 25010, the evaluation criteria include *internal* quality characteristics (i.e. concerning intrinsic properties), but exclude *external* quality characteristics (i.e. concerning behaviour in a test environment) or *quality-in-use* characteristics (i.e. concerning behaviour in a production environment). The evaluation concerns technical quality of the source code of a software product independent of its context of use or its role inside a system. In particular, the evaluation does not concern fulfilment of functional requirements or suitability for specified tasks.

3.2 SOFTWARE PRODUCT PROPERTIES

The quality characteristics of maintainability and its sub-characteristics are determined by measuring a set of software product properties. These properties are *volume, duplication, unit complexity, unit size, unit interfacing, module coupling, component balance*, *component independence, and component entanglement*.

- **Volume:** The overall size of the source code of the software product. Size is determined from the number of lines of code per programming language normalized with industry-average productivity factors for each programming language. Volume shall be rated on a scale that is independent of the type of the software product.

- **Duplication:** The degree of duplication in the source code of the software product. Duplication concerns occurrence of identical fragments of source code in more than one place in the product.

- **Unit complexity:** The degree of complexity in the units of the source code. The notion of unit corresponds to the smallest executable parts of source code, such as methods or functions.

- **Unit size:** The size of the source code units in terms of the number of their source code lines.

- **Unit interfacing:** The size of the interfaces of the units of the source code in terms of the number of interface parameter declarations.

- **Module coupling:** The coupling between modules in terms of the number of incoming dependencies for the modules of the source code. The notion of module corresponds to a grouping of related units.

- **Component balance:** Component balance is the product of the system breakdown, which is a rating for the number of top-level components in the system, and the component size uniformity, which is a rating for the size distribution of those top-level components. The notion of top-level components corresponds to the first subdivision of the source code modules of a system into components, where a component is a grouping of source code modules[1].

- **Component independence:** Component independence is a rating for the percentage of code in modules that have no incoming dependencies from modules in other top-level components. The choice of top-level components will affect which dependencies will be considered to be from the outside of the top-level component.

- **Component entanglement:** Component Entanglement indicates the percentage of communication between top-level components in the system that are part of commonly recognized architecture anti-patterns.

These software product properties have been chosen to be largely mutually independent. In particular, the total volume of a software product does not (statistically) predetermine the values for the other properties.

The determination of these software product properties is based on objective, repeatable, and accurate measurements at the level of source code lines, units, modules and components. When the software product under evaluation consists of source code written in more than a single programming language, these measurements are conducted per programming language, after which the results per programming language are combined into whole-product results. The combination method shall take into account the relative volume of artefacts written in each programming language.

3.3 SOFTWARE PRODUCT DESCRIPTION

In addition to the determination of the above product properties, a description of the product and its top-level components shall exist. This description shall satisfy the following minimal properties:

- The description identifies the product boundaries and its overall function.

[1] Where in the remainder of this document we refer to component, it is intended to mean top-level component. This term is further explained in [SIG 2021a].

- The description identifies all top-level components of the product.
- The description of the top-level components is such that any software artefact within the evaluation scope belongs to exactly one top-level component.
- The description identifies the role of each top-level component in the product.
- The description contains top-level components of appropriate number and size to facilitate maintenance of the product.

The high-level description of the software product, its top-level components, their functionality, interfaces, and interrelationships shall provide a global overview of the software product architecture and shall be appropriate for the understanding of the overall product to facilitate its maintenance.

4. EVALUATION RESULTS

The evaluation leads to an assignment of evaluation results as a number of stars between one and five, where more stars represent better performance. The assignment is done for every product property, for every quality sub-characteristic, and for the main characteristic of maintainability.

The assignment of evaluation results to measurement areas proceeds in a number of steps. In the first step, the measurement results per programming language are used to assign overall evaluation results to the product properties. In the second step, the evaluation results for the product properties are used to assign evaluation results to the sub-characteristics of maintainability. In the final step, the evaluation results for the sub-characteristics are used to assign a single evaluation result to the technical quality or *maintainability* of the product as a whole.

The assignment of evaluation results at the level of quality sub-characteristics based on the evaluation results at the level of product properties takes the following relationships into account.

- The **volume** of source code negatively impacts its **analyzability**, since the diagnosis of faults or parts to be modified is more difficult or time-consuming when a larger volume of source code needs to be taken into consideration.
- The **duplication** of source code negatively impacts its **analyzability**, since the diagnosis of faults or parts to be modified is more difficult or time-consuming when the same or similar parts occur multiple times in different places of the source code.
- The **unit size** of source code negatively impacts its **analyzability**, since the diagnosis of faults or parts to be modified is more difficult or time-consuming when the source code is grouped into a low number of large units rather than into a high number of small units.
- The **component balance** of source code positively impacts its **analyzability**, since components that are unbalanced are more likely to contain unrelated functionality and are therefore harder to analyze.

- The **duplication** of source code negatively impacts its **modifiability** since changes to duplicated pieces of code have to be made at different places in the source code.
- The **unit complexity** of source code negatively impacts its **modifiability** since the implementation of a modification in a specific part of the source code is more difficult or time-consuming when this part is complex.
- The **module coupling** of a system negatively impacts its **modifiability** since a change to a module that is frequently used in other parts of the code also requires changes to the pieces of code that use this module.

- The **volume** of source code negatively impacts its **testability** since a larger system has more code that needs to be tested.
- The **unit complexity** of source code negatively impacts its **testability**, since the validation of modifications to a software unit requires more test cases when the unit is complex.
- The **component independence** of source code positively impacts its **testability** since components that have a higher number of connections between them are more difficult to test in isolation.

- The **module coupling** of source code negatively impacts its **modularity** since a change to a module that is frequently used in other parts of the code also requires changes to the pieces of code that use this module.
- The **component balance** of source code positively impacts its **modularity** since unbalanced components or a large number of components are an indicator of bad modularity.
- The **component independence** of source code positively impacts its **modularity** since components that are more connected to each other can be considered less independent and are harder to change separately.
- The system's **component entanglement** negatively affects the system's **modularity**, since entangled components are harder to modify or replace [Neeraj 2005].

- The **unit size** of source code negatively influences **reusability** since larger units are more likely to contain more than one separate piece of functionality, thus making it harder to reuse each sub-function.
- The **unit interfacing** of source code negatively influences **reusability** since units with a high number of parameters are harder to instantiate because more knowledge about the context and expected parameter values for each of the parameters is required to reuse a unit.

These relationships between the product properties and quality sub-characteristics reflect expert opinion about the most significant influences of the various product properties on the quality sub-characteristics. They are summarized in the following table:

	Volume	Duplication	Unit size	Unit complexity	Unit interfacing	Module coupling	Component balance	Component independence	Component entanglement
Analyzability	■	■	■				■		
Modifiability		■		■		■			
Testability	■			■				■	
Modularity						■	■	■	■
Reusability			■		■				

A square indicates that a product property contributes to the maintainability sub-characteristic. All five sub-characteristics contribute to the main quality characteristics of *maintainability*. The weights of the different contributions are discussed in the following chapter.

5. CALIBRATION OF THE EVALUATION METHOD

Software quality measurements are meaningful when the measurement results reflect the relative position of a product within a large set of comparable products that have been evaluated recently. Thus, the numerical thresholds and weighting factors used in the evaluation shall be calibrated against a benchmark repository of source code analysis results of a large number of software products. The benchmark repository shall satisfy the following minimum requirements.

- The benchmark repository shall include products developed according to a range of different development methods, using a range of different programming languages, to the benefit of companies from a range of different industrial categories.
- The software products used for calibration shall include a substantial number of modern products, developed with modern programming languages.
- The total volume of the software products shall exceed 10 million lines of code. Small, medium-sized, and large products shall all be represented.
- The information in the benchmark shall be curated by experts in software engineering quality and statistics.

At the level of product properties, calibration is carried out according to the following guidelines. The relative number of products in the repository to which a given number of stars is assigned shall approximately follow the following distribution:

- **5 stars:** 5% of the products
- **4 stars:** 30% of the products
- **3 stars:** 30% of the products
- **2 stars:** 30% of the products
- **1 star:** 5% of the products

The best 5% of the products in the repository in terms of a given property (e.g. *unit complexity*) shall be assigned a 5 star rating; the next best 30% shall be assigned a 4 star rating; and so on. The worst 5% of the products receives a single star.

At the level of quality sub-characteristics and at the level of maintainability, the rating assignments are done based on those at the level of product properties. The distribution of stars at those levels may be different. At the level of quality characteristics, the relative importance of product properties for each sub-characteristic and the relative importance of sub-characteristics for the final evaluation results for maintainability are established by knowledge elicitation from a panel of software quality experts. A structured method will be used for knowledge elicitation.

6. ISSUANCE OF CERTIFICATE AND CORRESPONDING QUALITY MARK

To be eligible for certification with the quality mark *TÜViT Trusted Product Maintainability*, the evaluation results for a software product shall satisfy a number of minimal conditions:

- The rating at the level of the main quality characteristic of *maintainability* is 3 stars or more.
- The rating at the level of each sub-characteristic of maintainability is 2 stars or more.
- The documentation requirements must be fulfilled.

When these three conditions are satisfied and documented in an evaluation report of a licensed evaluation facility, a certificate can be awarded to the software product with a mention of the rating of the product at the level of *maintainability*.

7. SUMMARY OF THE EVALUATION PROCEDURE

The procedure for evaluating a software product against the evaluation criteria of SIG/TÜViT Trusted Product Maintainability is documented in detail in the *SIG Evaluation Guideline for Software Product Quality* [SIG 2018]. The procedure is structured according to the ISO/IEC 25040 International Standard for software product evaluation [ISO 25040].

The procedure involves the following steps:

- **Scoping:** In this step, the scope of the evaluation is established. This means that the source code files to be taken into consideration in the evaluation are identified. It may for instance be necessary to keep generated files out of scope. The scope is documented in a scope definition, which is presented to the evaluation client for confirmation.

- **Measuring:** In this step, the source code files within scope of the evaluation are measured. This results in a collection of measurement values at the level of source code lines, source code units, and source code modules.

- **Aggregation:** In this step, the measurements at the level of source code lines, units, and modules are aggregated to the level of properties of the product as a whole.

- **Rating:** In this step, the aggregated measurement values are used to assign ratings to the evaluation areas (product properties and quality sub-characteristics) listed in Chapter 4.

After these steps, the scope definition, measurement results, and ratings are recorded in an evaluation report.

8. REFERENCES

- [ISO 25010] International Organization for Standardization. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Systems and software quality models, 2010

- [ISO 25040] International Organization for Standardization. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Evaluation process, 2011

- [SIG 2020] SIG Evaluation Guideline for Software Product Quality, Version 12.0, Software Improvement Group, 2020.

- [SIG 2020a] SIG Evaluation Criteria for Software Product Quality – Guidance for Producers, Version 12.0, Software Improvement Group, 2020.

- [Neeraj 2005] Sangal, Neeraj and Jordan, Ev and Sinha, Vineet and Jackson, Daniel. “Using Dependency Models to Manage Complex Software Architecture”, In the Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA), pages 167—176,2005

APPENDICES

APPENDIX A - CONTACT INFORMATION

A.1 SOFTWARE IMPROVEMENT GROUP

Reinier Vis MSc
Head of Evaluation Laboratory

Fred. Roeskestraat 115
1076 EE Amsterdam
The Netherlands

T:	+31 20 314 09 50
E:	r.vis@sig.eu
W:	http://www.sig.eu

A.2 TUV INFORMATIONSTECHNIK GMBH

TÜV NORD GROUP
Dr. Christoph Sutter
Certification Division Manager

Langemarckstr. 20
45141 Essen
Germany

T:	+49 201 8999-582
F:	+49 201 8999-555
E:	c.sutter@tuvit.de
W:	https://www.tuvit.de/en/index.htm

Software Improvement Group

Fred. Roeskestraat 115
1076 EE Amsterdam
The Netherlands

www.softwareimprovementgroup.com
marketing@softwareimprovementgroup.com

Getting software right for a healthier digital world