

Connecting Research and Practice for Software Product Quality Evaluation and Certification

A Software Laboratory's 25-Year Journey

Javier Verdugo , Jesús Oviedo , and Moisés Rodríguez ,
AQCLab and University of Castilla-La Mancha

Mario Piattini , University of Castilla-La Mancha



Digital Object Identifier 10.1109/MS.2024.3357119
Date of publication 22 January 2024; date of current version 11 April 2024.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see <https://creativecommons.org/licenses/by/4.0/>

// We summarize difficulties encountered in the quest for transferring to the industry research work conducted on software metrics, which led to the creation of a startup that became an international accredited laboratory. We focus on the evaluation of two quality characteristics: functional suitability and maintainability. //

METRICS FOR EVALUATING software quality, and especially maintainability, are as old as the software engineering discipline itself, since the earliest work in this field dates from 1968.¹ The following decades saw the definition of metrics for the structured and object-oriented programming paradigms; methods for deriving software metrics, such as the goal question metric²; methods for the empirical validation of metrics³; and the publication of the series of standards ISO/IEC 25000 for software evaluation and measurement, known as *software product quality requirements and evaluation (SQuaRE)*.⁴

Introduction

In this article, we present our research journey in the field of software product quality and its application in the industry, culminating in the establishment of a laboratory for evaluating software maintainability initially and, later, also functional suitability

based on ISO/IEC 25000. Through the services provided by the laboratory we would complement the internal quality assurance processes of software development companies, allowing them to obtain rigorous feedback from a specialized third party, and even be able to obtain a quality certificate for their product from a certification body.

point of view of its maintainability, the nonfunctional requirement type on which we had previously conducted research.

Throughout these projects, the main “complaints” that we usually encountered were, on the one hand, that most of the metrics and measurement proposals were too “academic” and not practical enough,

Challenges and Barriers in the Transfer of Software Quality Evaluation to the Industry

In the early 2000s, we saw an opportunity to provide software quality evaluation services to companies, initially focusing on maintainability. However, there were several barriers to the feasibility of this idea:

- At that time, many software development companies were not yet aware of product quality and did not invest in tools and techniques in this area. In fact, advanced tools like SonarQube, which are nowadays common in the industry, did not appear on the market until 2009.
- There were not many business cases that really proved to companies that the investment in software maintainability was actually worthwhile, despite the huge cost software maintenance typically entails (which often exceeds 80% of a project’s budget).
- Some companies to whom we presented our maintainability evaluation service stated that they would not bother to make their products more maintainable unless their customers demanded it. Some companies would even consider it detrimental to their business, as they were charging their customers on turnkey projects for the time they dedicated to maintenance.
- As in the case of products in other sectors, companies demanded not only a third-party evaluation, but also an “official” (accredited) quality certificate that could endorse the adequacy of their certified products to their customers.

From the outset, we focused on the internal quality of software from the point of view of its maintainability, the nonfunctional requirement type on which we had previously conducted research.

Here we detail the efforts we made and the challenges we faced in this journey, and also share the lessons we learned and some valuable feedback we obtained from the companies that we have collaborated with throughout the years. Additionally, we provide some details of the model that we use to evaluate maintainability.

First Steps in the Software Quality Evaluation Journey

In the late 1990s, with the establishment of several software factories close to our university and the increase of software development outsourcing, we—under the Alarcos Research Group—participated in several projects aimed at improving the quality of software products. From the outset, we focused on the internal quality of software from the

and on the other hand, that there was no certification scheme for software products, similar to those that exist for other types of products, such as electronics, food, chemicals, etc.

In fact, despite all of the existing research on metrics, and especially those related to software maintainability, it has not been easy to establish them as a basis for “certifying” the quality of the software product, in contrast to models, such as the capability maturity model/ capability maturity model integration (CMM/ CMMI), or standards, such as ISO/IEC 15504 and the subsequent ISO/IEC 33000 series, which were much more quickly adopted for the assessment and certification of software process quality. Although there have been some promising proposals,^{5,6,7} they have struggled to make the leap into the industry.

After researching a solution to these issues, we determined that aligning our software product evaluation service to the ISO/IEC 25000 standard family would be the optimal approach. This new series of standards revised ISO/IEC 9126 and proposed a set of software quality characteristics that consider both functional aspects (through functional suitability) and nonfunctional aspects, like security, performance efficiency, reliability, or maintainability. The latter was our primary interest and field of expertise, although we would consider other characteristics in our roadmap, given their relevance to software quality.

Several standards in this series would be especially relevant for us: ISO/IEC 25010 as the basis for our quality model, ISO/IEC 25040 as the reference for the evaluation process (defining the activities, tasks, and inputs and outputs for each of them), and ISO/IEC 25023 as the basis for the metrics to be used. There was a drawback to this approach, as ISO/IEC 25023 provides metrics, particularly in the case of maintainability, that developers may find unhelpful due to issues, such as the ones outlined below:

- Many of the proposed metrics, such as “modification capability,” which is defined as “ $X = A/B$, where A = number of items actually modified within specified duration and B = number of items required to be modified,” can only be measured once the system is in use. Instead, it would be more sensible to measure maintainability-related metrics during software development to prevent maintenance issues.
- With regards to metrics measured on the source code itself, only

cyclomatic complexity is considered, but many others, such as object-oriented metrics (NOC, DIT, etc.), duplicated code, or dependency cycles, are neglected.

- Analyzability metrics are also based on aspects to be checked when the system is in use, such as the extent to which the system logs trace operations of the software or system, rather than considering readability and other useful attributes of the code itself that would be useful for its maintenance.

This is often the problem with metrics proposed by academic researchers and standardization bodies: in principle they are well-thought and theoretically valid, but when put into practice by industry professionals, they may be difficult to measure or not as useful in identifying problems as intended.

Maintainability Evaluation Model

Because of the aforementioned flaws in the metrics defined in ISO/IEC 25023 for the maintainability characteristics, we decided not to use them in the evaluation model that we were defining, and instead use a set of metrics that could be measured taking only the software source code as input, so that they could be obtained at any point during the development phase.

In this way, we defined an evaluation model that specifies how to obtain quality scores for the maintainability characteristic and its five subcharacteristics through the definition of a set of quality properties, metrics, and functions.

The hierarchical evaluation model features maintainability at the top level, followed by its five subcharacteristics (analyzability, modularity, modifiability, reusability, and

testability) at the second level. The third level consists of a set of quality properties associated to one or more subcharacteristics. The bottom level comprises base and derived metrics, which are used to determine the properties’ scores. Table 1 shows the properties that influence the score for each subcharacteristic in the model.

A set of functions is used to derive the quality score for elements in the upper levels from those directly below. In this way, functions for quality properties in the model allow us to aggregate the value of base metrics for target elements (such as functions, classes, or packages) so that a quality score can be obtained for the whole product. The score for properties is a normalized value in the range [0, 100], as is the score for subcharacteristics that is derived from the score of their associated properties. Finally, the maintainability score takes a discrete value in the range [1, 5], derived from the subcharacteristics’ scores.

Table 2 presents a summary of the fundamental concepts concerning the elements in the lower levels of the model: quality properties and corresponding metrics. In some cases, the metrics are measured at the system level, whereas in other cases they are measured against more specific targets, such as functions/methods, classes, or packages. The values for these individual targets are aggregated to derive a score for the system with respect to each specific quality property. This aggregation process is carried out by calculating the number of elements of the system at different levels (typically three), leading to the creation of a profile from which the qualitative score is obtained.

Taking the complexity property as an example, the metric cyclomatic complexity is calculated for each function in the product under

Table 1. Association between properties and subcharacteristics in the maintainability model.

		Subcharacteristics				
		Analyzability	Modularity	Modifiability	Reusability	Testability
Properties	Rule violations	X	X	X	X	X
	Code duplication	X		X		X
	Complexity	X		X		X
	Class structuring	X	X			X
	Function size	X	X			
	Code documentation	X			X	
	Package structuring	X	X		X	X
	Cohesion	X	X			
	Dependency cycles		X	X	X	X
	Abstractness and instability balance		X	X	X	X

evaluation. The profile of the product is then obtained by classifying the functions into three levels (depending on the range in which their cyclomatic complexity falls, as shown in Table 2) and obtaining the number of functions in each of these levels. Using this profile, the complexity evaluation function provides the score in the range [0, 100].

As another example, in the case of the rule violations property, the classification into levels is based on the severity of the issues. The profile in this case consists of the coding rule violations density for each of the three levels. The evaluation function is then applied to this profile to derive the score for the property.

Implementation and Accreditation of the Laboratory

In 2009, in order to respond to the industry’s need for an official

Table 2. Detail of properties and metrics in the maintainability model.

Property	Base metric	Target	Range of values		
			Level 3	Level 2	Level 1
Rule violations	Coding rule violations per lines of code	System	Low severity	Medium severity	High severity
Code duplication	Percentage of duplicated code	System	[0, 4)	[4, 7)	[7, 100]
Complexity	Cyclomatic complexity	Function	[1, 10]	(10, 15]	[15, ∞)
Class structuring	Number of functions	Class	[0, 12]	(12, 18]	(18, ∞)
Function size	Lines of code	Function	[1,15]	(15, 30]	[0] U (30, ∞)
Code documentation	Percentage of lines of commentary	System	(22, 50)	(10, 22] U [50, 60)	[0, 10] U [60, 100]
Package structuring	Number of classes	Package	[1, 15]	(15, 25]	[0] U (25, ∞)
Cohesion	Lack of cohesion on methods 5	Class	[0, .33)	[.33, .66)	[.66, 2]
Dependency cycles	Percentage of packages involved in dependency cycles	System	[0]	(0, 50]	(50, 100]
Abstractness and instability balance	Distance from the main sequence (abstractness + instability – 1)	Package	[0, .43)	[.43, .68)	[.68, 1]

certification of the quality of software products, we contacted AENOR, the leading international certification body in the Spanish market. One of the requirements set by AENOR was that, to be taken as the basis for official certification, the software quality evaluations had to be carried out by a laboratory accredited to ISO/IEC 17025, the international standard that establishes the “general requirements for the competence of testing and calibration laboratories.”

Thus, the idea of AQCLab, a laboratory for the evaluation of software quality, was born. The goal for the laboratory would be to provide its clients, software development companies, with software quality evaluation services carried out by an expert third party. The evaluations of the laboratory would be intended to be complementary to—and not substitute for—the continuous quality assurance improvement practices that companies may have ingrained in their development lifecycle. In this way, the success of a company’s quality assurance practices would be reflected in their software products and, through the evaluation of the laboratory and subsequent certification, they could earn a mark of achievement that would help them to gain the trust from potential customers.

In addition, this laboratory would enable collaborative innovation between academia and industry, allowing not only the transfer of our software evaluation research to the industry, but also the improvement of university teaching by incorporating lessons learned from the application of this research into several degree subjects related to software engineering.

To become an accredited laboratory, we contacted Entidad Nacional

de Acreditación (ENAC), the Spanish accreditation body (signatory to the International Laboratory Accreditation Cooperation mutual recognition arrangement), and we began to study and implement the requirements that we would have to meet.

With these matters in mind, we started to set up the laboratory, and since the requirements were not completely clear from the beginning, we decided to take an agile approach for this purpose.⁸ This would allow us to make any necessary changes as we went deeper into implementing the requirements of ISO/IEC 17025 and adapting them to our particular case. The whole process took us from 2008 to 2010, when we completed the quality model for maintainability,⁹ the evaluation methodology and all of the laboratory procedures, and the technological environment with the tools necessary for measurement and evaluation. Of all the work that we carried out in this process, the following achievements are worth highlighting:

- *Defining a set of metrics that could “predict” quality before software products reach the market or production environment:* As indicated above, many metrics proposed by the ISO/IEC 25023 standard are meant to be measured when the product is already in use. However, we intended to determine whether a software product was of sufficient quality during its development stage. For this reason, we discarded the metrics proposed by ISO/IEC 25023 and, instead, we carried out systematic literature reviews, consulted with experts, and tested dozens of metrics until we selected a set that would be useful in a

practical development context and would reflect adequately the maintainability properties of the software product.

- *Expressing the evaluation results in a way that would be easier for potential clients to understand:* We decided to establish a quality product scale, analogous to one used in process quality assessments by CMMI and ISO/IEC 33000, with values that would range from 1 (very low quality) to 5 (very high quality). For this purpose, we proposed a function based on the use of profiles that would provide standardized quality values for each property, subcharacteristic, and characteristic of the quality model.¹⁰
- *Supporting a reasonable set of technologies and programming languages:* Supporting all existing programming languages is not feasible, since there are relevant differences among them in how the metrics can be measured, which programming rules should be checked, and which tools can be used for these tasks. Therefore, we had to decide which subset to support, finally opting for the technologies that were most used and had the best future projections at the time (Java, .NET, PHP, JavaScript, Python, etc.).
- *Having a unified tool environment that supports the measurement and evaluation of software product quality:* To achieve this, we reviewed many measurement tools, since at that time integrated platforms, such as SonarQube, were not mature. It was also necessary to create a tool that would apply the evaluation functions to obtain the quality values for the elements in the upper levels of the model.

- *Carrying out a correlational study to test that the results obtained with our model and environment correspond to actual maintenance effort:* To this end, we carried out the evaluation of hundreds of products implemented with different languages and made the necessary adjustments to the metric thresholds established in our model.

At the beginning of 2012, we obtained the ISO/IEC 17025 accreditation for maintainability evaluations. In the following years, we identified that the industry was interested not only in the maintainability of software, but also in determining whether the products they would purchase were functionally complete and correct. Therefore, we subsequently followed this same process to implement the evaluation of functional suitability in conformance to ISO/IEC 25000. To achieve this, we had to define the corresponding metrics (test completeness, functional correctness, etc.), select and develop supporting tools, carry out new correlational studies, etc. Finally, in 2015, we became the first laboratory to be accredited for the evaluation of this characteristic. Once accredited, we established a collaboration with certification bodies, such as AENOR so that functional suitability certificates could be issued, based on our evaluations.

The model that we use in functionality suitability evaluations is presented in detail in Rodríguez et al.¹⁰ In this case, it is worth noting that the metrics proposed in ISO/IEC 25023 were found to be useful, although some adaptations were still required to make them fully operational in our model, such as including the coverage of code executed by

tests as a metric that affects the score of the properties related to the functional correctness subcharacteristic.

Software Product Certification

Throughout this decade, we have evaluated nearly a hundred products from companies of various sizes, industries, and countries. These companies, originating from countries such as Spain, Italy, Portugal, and Peru, have ranged from small businesses with only a few staff members to large banking companies with thousands of employees. The evaluated products featured a wide range of technologies, such as Java, C, .NET, PHP, JavaScript, and Python, among others. Despite the difficulty in achieving the required quality level for certification due to its rigorosity, approximately 30 products have obtained a certificate so far, which can be consulted on the ISO 25000 portal (<https://iso25000.com/index.php/en/certified-products>).

After conducting a high number of evaluations, we have identified several recurring problems. In the case of maintainability, companies do not always use during their development cycle a tool that helps them to monitor compliance with coding standards and good practices. This leads to source code that does not adhere to said good practices, has high complexity, a lot of duplicated code and, consequently, a high level of technical debt that impacts very negatively in its maintainability.

As for the case of functional suitability, companies do not always have a well-defined testing process. As a result, not all requirements have test cases to verify their correct and complete implementation, and therefore it is not possible to guarantee the correct functioning of their

software. In addition, many organizations are unfamiliar with the concept of code coverage, resulting in many fragments of code remaining untested.

Over the years, we have also been collecting feedback from organizations that went through the whole process regarding the advantages that evaluating, improving, and then certifying their software products brought them. Here are some of the most significant insights we have received:

- It enabled reducing the number of corrective maintenance incidents for the software product by up to 75%.
- The complexity of the product was decreased by up to 45%.
- In some cases, maintenance times were diminished by up to 30%.
- Several products reduced their number of lines of code by as much as 40% after eliminating duplicated and dead code.
- The products considerably increased the code coverage achieved by the tests, in many cases surpassing 80% of the source code.
- In addition, several companies managed to establish traceability between the quality of their processes with standards such as ISO/IEC 33000, and the quality of the software product with the ISO/IEC 25000 series.

Lessons Learned

As a result of the work carried out in the laboratory, knowledge transfer between academia and industry has been boosted over the years in two ways: in terms of transfer from academia to industry, an increasing number of development companies



JAVIER VERDUGO is the technical director of AQCLab, 13005 Ciudad Real, Spain, and a part-time professor and part of the Alarcos Research Group, University of Castilla-La Mancha, 13071 Ciudad Real, Spain. His research interests include software product and data quality evaluation, software processes, and systems security. Verdugo received his Ph.D. in computer science from the University of Castilla-La Mancha. Contact him at jverdugo@aqclab.es.



MOISÉS RODRÍGUEZ is the CEO of AQCLab, 13005 Ciudad Real, Spain, and an associate professor at the Escuela Superior de Informática, part of the Alarcos Research Group, University of Castilla-La Mancha, 13071 Ciudad Real, Spain. His research interests include software processes, product, and data quality. Rodríguez received his Ph.D. in computer science from the University of Castilla-La Mancha. Contact him at mrodriguez@aqclab.es.



JESÚS OVIEDO is a consultant in software quality at AQCLab, 13005 Ciudad Real, Spain, and a part-time professor and part of the Alarcos Research Group, University of Castilla-La Mancha, 13071 Ciudad Real, Spain. His research interests include software product and data quality evaluation, software processes, and systems security. Oviedo received his M.Sc. in computer science from the University of Castilla-La Mancha. Contact him at joviedo@aqclab.es.



MARIO PIATTINI is a full professor at the Escuela Superior de Informática, and leads the Alarcos Research Group, University of Castilla-La Mancha, 13071 Ciudad Real, Spain. His research interests include software engineering and information systems quality. Piattini received his Ph.D. in computer science from the Technical University of Madrid. He is a Member of IEEE. Contact him at mario.piattini@uclm.es.

have implemented software quality-control processes, and companies acquiring software have become more aware of software quality, including requirements in their bidding processes concerning the certification of compliance to ISO/IEC 25000. On the other hand, as far as the transfer from industry to academia is concerned, every year more and more university students are being trained in aspects of software quality, which we have learned from our experience in

industry, with the intention of preparing them to put this knowledge into practice in their careers.

In addition to the previous points made in this article, the main lesson we have learned from this journey is realizing the mistake we made in thinking that the process of transferring research to industry was linear, and that we needed to spend many years researching and validating metrics for software quality and then, once we had the right metrics, we could go to industry to


put them into practice. We probably made a simplification mistake by not walking side by side with industry¹¹ right from the beginning, because as Mikkonen et al.¹² point out, this approach is no longer valid.

In retrospect, it would have been more suitable to use what is known today as the *coproduction process*,¹³ involving all of the entities of the ecosystem (organizations interested in evaluating and certifying

their software products, evaluation laboratories, certification bodies, software quality consultants, and tool vendors) from the beginning, to understand their requirements, needs, and functioning processes. The problem with this ecosystem is that some actors did not yet exist when we started, or at least, did not yet act as such. It was not until the ecosystem started to operate that the different entities were able to play their respective roles.

As for the main drivers that lead to the improvement of quality in the software industry, we have identified the following based on our experience:

- It is necessary to create an ecosystem that brings together the necessary key actors, establishing clear relationships between them.
- Development companies need to implement tools and processes that allow them to continuously control quality throughout the development life cycle. Otherwise, episodic evaluations lead to a quality derailment that is difficult to get back on track.
- Organizations purchasing software products should be aware that they can demand objective levels of quality in line with international standards.
- Both software development and consumer organizations, as well as universities, should promote the training of current and future professionals in aspects related to software quality.

All of these lessons learned were subsequently applied in the development of a data-quality certification scheme based on the ISO/IEC 25012 standard.¹⁴ 

Acknowledgment

We thank the rest of our colleagues at the Alarcos Group and AQCLab, who have collaborated all these years to make the experiences presented in this article possible; Carlos Manuel Fernandez and Boris Delgado from AENOR for their support and collaboration in making our research materialize into a certification scheme in the software industry; and Elvira González from Entidad Nacional de Acreditación for her work and recommendations, which helped AQCLab to become the first and only laboratory accredited for software quality evaluations.

References

1. R. J. Rubey and R. D. Hartwick, "Quantitative measurement of program quality," in *Proc. 23rd ACM Nat. Conf.*, Jan. 1968, pp. 671–677, doi: 10.1145/800186.810631.
2. V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 6, pp. 728–738, Nov. 1984, doi: 10.1109/TSE.1984.5010301.
3. C. Wohlin et al., *Experimentation in Software Engineering: An Introduction*. New York, NY, USA: Springer-Verlag, 2000.
4. *Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)*, ISO/IEC Standard 25000 SQuaRE Series. [Online]. Available: <https://committee.iso.org/sites/jtc1sc7/home/projects/flagship-standards/iso-25000-square-series.html>
5. A. Alvaro et al., "Towards a software component certification framework," in *Proc. 7th Int. Conf. Qual. Softw.*, 2007, pp. 298–303, doi: 10.1109/QSIC.2007.4385511.
6. J. H. Yahaya et al., "SCfM_PROD: A software product certification model," in *Proc. 3rd Int. Conf. Inf. Commun. Technol., Theory Appl. (ICTTA)*, 2008, pp. 1–6, doi: 10.1109/ICTTA.2008.4530350.
7. P. Heck et al., "A software product certification model," *Softw. Qual. J.*, vol. 18, no. 1, pp. 37–55, 2009, doi: 10.1007/s11219-009-9080-0.
8. J. Verdugo et al., "Using agile methods to implement a laboratory for software product quality evaluation," in *Proc. Int. Conf. Agile Softw. Develop.*, 2014, pp. 143–156, doi: 10.1007/978-3-319-06862-6_10.
9. M. Rodriguez et al., "A hard look at software quality," *Qual. Prog.*, vol. 48, no. 9, p. 30, 2015.
10. M. Rodriguez et al., "Evaluation of software product functional suitability: A case study," *Softw. Qual. Professional*, vol. 18, no. 3, p. 18, 2016.
11. V. Garousi et al., "Practical relevance of software engineering research: Synthesizing the community's voice," *Empirical Softw. Eng.*, vol. 25, no. 3, pp. 1687–1754, 2020, doi: 10.1007/s10664-020-09803-0.
12. T. Mikkonen et al., "Continuous and collaborative technology transfer: Software engineering research with real-time industry impact," *Inf. Softw. Technol.*, vol. 95, pp. 34–45, Mar. 2018, doi: 10.1016/j.infsof.2017.10.013.
13. A. Sannö et al., "Increasing the impact of industry–Academia collaboration through co-production," *Technol. Innov. Manage. Rev.*, vol. 9, no. 4, pp. 37–48, 2019, doi: 10.22215/timreview/1232.
14. F. Gualo et al., "Data quality certification using ISO/IEC 25012: Industrial experiences," *J. Syst. Softw.*, vol. 176, Jun. 2021, Art. no. 110938, doi: 10.1016/j.jss.2021.110938.