The Squale Model - A Practice-based Industrial Quality Model

Karine Mordal-Manet, Françoise Balmas, Simon Denier, Stéphane Ducasse,
Harald Wertz, Jannik Laval, Fabrice Bellingard, Philippe Vaillergues

The Squale Model – A *Practice*-based Industrial Quality Model
Long version of the short paper presented at ICSM 2009

Karine Mordal-Manet[1]   Françoise Balmas[1]   Simon Denier[2]   Stéphane Ducasse[2]
Harald Wertz[1]   Jannik Laval[2]   Fabrice Bellingard[3]
Philippe Vaillergues[3]

[1] RMoD Team, INRIA, Lille, France
[2] Qualixo, Paris, France
[3] LIASD, University of Paris 8, France
kmordal@gmail.com
http://www.squale.org

Abstract

ISO 9126 promotes a three-level model of quality (factors, criteria, and metrics) which allows one to assess quality at the top level of factors and criteria. However, it is difficult to use this model as a tool to increase software quality.

In the Squale model, we propose the adjunction of practices *as an intermediate level between metrics and criteria. Practices abstract from raw information at the source (metrics, tool reports, audits) to provide the developer with technical guidelines to respect. Moreover, practice marks can be adjusted using formulae to suit company development habits or exigences: for example bad marks can be stressed to point to places which need the most attention. Dashboards allow one to spot faulty practices and find the source elements responsible for the bad marks. The Squale model has been developed and validated over the last couple of years in an industrial setting with Air France-KML and PSA Peugeot-Citroën. Over 100 projects with a total of more than seven millions lines of code have been assessed and steered using Squale.*

1 Introduction

Software quality aims at setting up standards to achieve and measuring the conformance of a project with its given standard. Companies use software quality as a mean to assess risks in the course of software development. Such risks include for example faulty software, inertia to change, misunderstanding, which turn out in longer time to market and higher costs. Thus a second goal is to provide means to increase software quality in the form of guidelines and recommendations to reduce these risks.

With respect to these goals, software quality models classify different categories of risks and describe how such risks can be assessed from project data, including source code, documentation and project conventions.

Software quality models often present multiple levels of quality assessment in a top-down fashion. ISO 9126 [9] describes such a quality model with the three levels of factors, criteria, and metrics. Factors ("non-technical" quality properties) are decomposed into criteria (high level technical properties), which are further decomposed in terms of metrics computable on the project data.

However, this model is difficult to use as a mean to increase software quality. The model can not explain what to do at the metrics level (*i.e.*, project sources) to improve factors and criteria levels [12].

In addition, top-down models aggregate metrics value using simple average functions, which smooth bad metric values, potentially hiding bad quality at low level. Stéf ►*not that clear*◄

We consider the following principles important to define an *efficient* software quality model in nowadays software development process:

- *roundtrip model* – the model should assess high level quality from project data, but also pinpoint the components at low level responsible for the (bad) quality.

- *stress function* – the model should stress bad quality in the project, calling for a quick focus from the developer.

We propose the adjunction of *practices* as an intermediate level between metrics and criteria. They cover different aspects of a software project——including documentation,

programming conventions, and test coverage. Practices abstract from raw information at the source (metrics, tool reports, audits) to provide the developer with technical guidelines to respect. Examples of practices include:

- complex methods should be more documented than simple methods,

- complex classes should be more covered by tests than trivial ones,

- spaghetti code should be avoided, or

- dependency cycles should be minimized.

Moreover, practice marks can be adjusted using formulae to suit company development habits or standards. For example low marks can be stressed to point to places (method, class, package...) which need the most attention.

The roundtrip model can be achieved through dashboards, which show all practices as well as marks for all components for a given practice. It allows one to analyze defective practices and find the project components responsible for the bad marks.

This model has been first implemented by Qualixo enterprise in a industrial setting with Air France-KLM, beginning in 2006. It is intensively used to monitor multiple projects, for a total of seven MLOC. Since then other companies such as PSA Peugeot-Citroën (PSA) are using it. The model is now promoted as the open-source Squale quality model.

The paper is structured as follows. Section 2 gives an overview of the Squale model. Section 3 presents in details the level of practices, discussing the use of practices and the functions to compute practice marks. Section 4 reports on different instances of the model in industrial settings. Section 5 discusses related work and Section 6 concludes with perspectives.

2 The Squale quality model

The Squale model is inspired by the factors-criteria-metrics model (FCM) of McCall [15]. However, while McCall defined a top-down model to express the quality of a system, the Squale model promotes a bottom-up approach, aggregating low-level measures into more abstract quality elements. This approach ensures that the computation of top-level quality assessments is always grounded by concrete repeatable measures or audit on actual project components.

The Squale model introduces the new level of *practices* between criteria and metrics. Practices are the key elements which bridge the gap between the low-level measures, based on metrics, rule checkers or human audits, and the top-level quality assessments—expressed through criteria and factors. Thus the Squale model is composed of four levels (see Figure 1): factors, criteria, practices, and measures.

The three top levels of Squale use the standard mark system defined by the ISO 9126 standard. All quality marks take their value in the range $[0; 3]$, as shown in Figure 1, to support an uniform interpretation and comparison:

- between 0 and 1, the goal is not achieved;

- between 1 and 2, the goal is achieved but with some reservations;

- between 2 and 3, the goal is achieved.

The following subsections briefly present the four levels of the Squale model, from the bottom measures to the top factors.

2.1 Measures

A *measure* is a raw information extracted from the project data.

The Squale model takes into account different kinds of measure to assess the quality of a software project: automatically computable measures that can be computed easily and as often as needed, and manual measures which have a predefined life time and must be updated mainly after major changes to the software.

The automatically computable measures are divided into three groups. The first group is composed of metrics [8, 13, 5] like Number of Lines of Code [6], Hierarchy Nesting Level or Depth of Inheritance Tree [10], or cyclomatic complexity [14]. A preliminary analysis selected only relevant metrics [2][1]. However, Squale is able to adapt to a wide range of metrics provided by external tools. The second group is composed of rules checking analysis like syntactic rules or naming rules, which verify that programming conventions are enforced in the source code and allow one to correct some bugs. These rules are defined before starting the project and must be known by developers. The third group is composed of measures which qualify the quality of tests applied to the project such as test coverage. This group may also contain security vulnerability analysis results.

The manual measures express the analysis made by human expertise during audits. These measures qualify the documentation needed for a project, such as specification documents or quality assurance plan. They verify also that the implementation of the project respects the documented constraints.

A measure is computed with respect to its scoping entity in the project data: method, class, package, or the project itself for an object-oriented software.

---

[1] http://www.squale.org/quality-models-site/
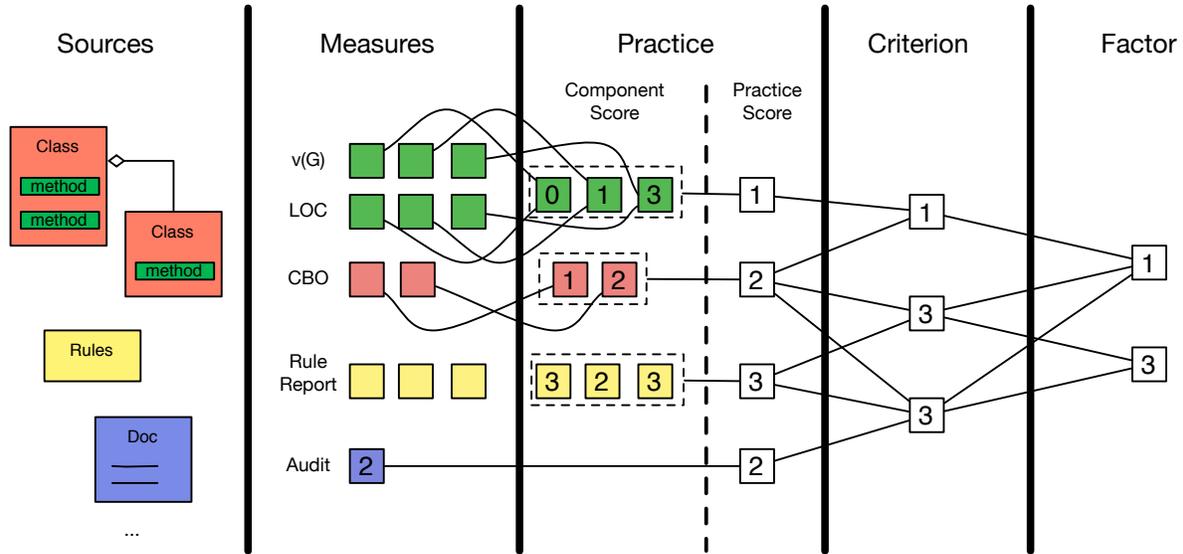
Figure 1. Data sources and levels of the Squale model.

Around 50 to 200 different measures are used in various instances of the Squale model. Usable measures depend on the available tools, the current stage in the project life-cycle, and the requirements of the company.

2.2 Practices

A *practice* assesses the respect of a technical principle in the project (such as *complex classes should be more documented than trivial ones*). It is directly addressed to the developer in terms of good or bad property with respect to the project quality. Good practices should be fulfilled while bad practices should be avoided. The overall set of practices expresses rules to achieve optimum software quality from a developer's point of view. Around 50 practices have been defined based on Air France quality standards. However, the list of practices is not closed and such practices can be adjusted.

A practice combines and weights different measures to assess the fulfillment of technical principles. A practice mark can be computed for an individual element of the source code. A global mark for the practice adjusts the variations of the individual marks. We detail this aspect in Section 3.1.

For example, the *comment rate* practice combines the *comment rate per method LOC* and *cyclomatic complexity* of a method to relate the number of comments in the source code with the complexity of the method: the more complex the method, the more comments it should have.

2.3 Criteria

A *criterion* assesses one principle of software quality (*safety, simplicity*, or *modularity* for example). It is addressed to managers as a detailed level to understand more finely project quality. The criteria used in the Squale model are adapted to face the special needs of Air France and PSA. In particular, they are tailored for the assessment of quality in *information systems*.

A criterion aggregates a set of practices. A criterion marks is computed as the weighted average of the composed practice marks. Currently around 15 criteria are defined.

For example, the following practices:

- comment rate (per method with respect to cyclomatic complexity)

- inheritance depth

- documentation achievement (human audit with respect to project requirements)

- documentation quality (rule checking of programming conventions)

define the *comprehension* criterion.

2.4 Factors

A *factor* represents the highest quality assessment to provide an overview of project health (*Functional capacity* or *reliability* for example). It is addressed to non-technical persons. A factor aggregates a set of criteria. A factor mark is computed as the average of the composed criteria marks.

3

The six factors used in the Squale model are inspired by the ISO 9126 factors and refined based on the experience and needs of engineers from PSA, Air France, and Qualixo.

For example, the following criteria :

- Homogeneity

- Comprehension

- Simplicity

- Integration Capacity

define the *capacity to correct* factor. This means that a system should be easier to correct when it is homogeneous (respect of architectural layers and of programming conventions for names), simple to understand and modify (good documentation, manageable size), and conveniently coupled.

3 Practices in Detail

We now present in detail the practice layer and its specification as it defines the backbone of the Squale model. We describe the marking system which allows one to map measures onto global practice marks and to adjust such marks to company culture (*e.g.*, to stress practices with bad marks). We show the use of dashboards to analyze practices and perform quality diagnosis. We describe the adaptability of the model to the specificity of a project.

A global mark for a practice is computed in two steps:

Individual mark Each element (method, class, or package in object-oriented programs) targeted by a practice is given a mark with respect to its measures. For example, the two metrics composing the *comment rate* practice, *cyclomatic complexity* and *source line of code*, are defined at the method level; thus a *comment rate* mark can be computed for each method.

Global mark A global mark for the practice is computed using a weighted average of the previous individual marks.

The different formulae also normalize practice marks to enable comparison between practices on a common scale.

For example, Table 1 summarizes elements that define the *comment rate* practice. Its definition—the rate of comments in the lines of code— determine which measures are used to compute it mark: the cyclomatic complexity and the number of source lines of code. The scope defined for this practice—method—correspond to the scoping metrics used for this practice. The formulae used in the two steps mentioned above are regrouped in a set of equations for individual mark and an equation to obtain the global practice mark from individual mark.

Practice name	*Comment rate*
Scope	Method
Measures	Cyclomatic complexity: v(G) and SLOC
Definition	Qualify the rate of comments in the lines of code.
Formula	Continuous
Individual mark	If $v(G) < 5$ and $sloc < 30$ then Imark = 3 else : $Imark = \%\_comments\_per\_loc$ $/(1 - 10^{(-v(G)/15)})$
Practice mark	$mark = -log_{20}(average(20^{-Imark}))$

Table 1. The *comment rate* **practice**

Practice name	*Functional specifications*
Scope	Project
Measures	Audit by expert
Definition	Verify if there are functional specifications (FS) for the project
Formula	Discrete
Individual mark	Imark = 0 if no FS Imark = 1 or 2 if FS but not entirely correct Imark = 3 if FS correct
Practice mark	Same as individual mark

Table 2. The *functional specifications* **practice**

3.1 Individual mark

The formulae for computing individual marks come as two kinds, discrete or continuous. An individual mark is computed from measures in multiple ranges into a single mark in the range $[0; 3]$:

- discrete formula. An example of such formula is given in Table 2 for the *functional specifications* practice.

- continuous formula. Table 1 shows a continuous nonlinear formula for the *comment rate* practice.

A discrete marking system is simple to implement and easy to read. It is well adapted to manual measures such as audits. For example, the practice for *functional specifications* described in Table 2 is given a mark in a discrete range. If there is no functional specification, the mark 0 is given. If functional specifications are consistent with the client requirements, the mark 3 is given. The two intermediate marks are used to qualify existing yet incorrect functional specifications. Thus this mark assesses two information: the existence of functional specifications and their consistency. While the practice can only be evaluated by an

NAVIGATION DANS LES COMPOSANTS DU PROJET

▸ Aide

Composant actuel : **Classe CpdTransgressionBO**
▸ Filtrer

| Contenu du composant | Résultats du composant | Information générale |

Notes du composant sur les pratiques (seules celles qui ont été calculées sont affichées).

Total : 6

	Pratique	Valeur (comprise entre 0 et 3) ⇵	
	Profondeur d'héritage	3.0	
	Couplage efferent	3.0	
	Cohésion Classe	2.0	
	Couteau suisse	3.0	
	Couplage afferent	3.0	
	Nombre de méthodes	3.0	

Total : 6

Valeurs des mesures sur le composant.

Total : 14 ◂◂ précédent *1 2* suivant ▸▸

Métrique	Valeur	
Nombre de classes dérivées	0.0	
Cohésion de la classe	21.0	
Couplage afférent	0.0	
Couplage entre classes	1.0	
Profondeur d'héritage	3.0	
Nombre de méthodes publiques	6.0	
Nombre pondéré de méthodes par classes	7.0	
Nombre de méthodes accessibles	20.0	
Nombre de classes	0.0	
Nombre de méthodes	7.0	

Total : 14 ◂◂ précédent *1 2* suivant ▸▸

Figure 2. Dashboard view of (individual) practice marks and metric values for a class.
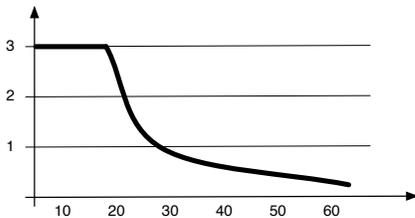
Figure 3. Sample graph for a practice mark based on one measure.

expert, the discrete range limits the subjectivity of the given mark.

Discrete marking is not adapted to all practices. For metrics-based practices, the discrete formula introduces staircase values and threshold effects, which smoothes detailed information and triggers wrong interpretation. When surveying the evolution of quality, it hides slight fluctuations—progression or regression—of an individual element.

A continuous formula is used to avoid this phenomenon. It better translates the variations of metric values on the mark scale. Indeed, such formulae are first built around a couple of measure-mark binding, agreed upon by the experts. For example, the continuous equation in Figure 3 should give the mark 1 for value 27 and the mark 2 for value 21, then the mark 0.5 for value 45. Then, the formula is defined as a linear or non-linear equation which best approximate those special values and allows one to interpolate marks for any value.

Figure 3 shows a mixed example using discrete and continuous equations of correspondence between a single measure (x axis) and its given mark (y axis). First there is a threshold of 20 below which the mark is automatically 3 (the continuous equation is clipped). It is the maximal value which allows one to achieve the goal. Above this threshold, the individual mark decreases following an exponential curve: the individual mark tends quickly towards zero.

The different formulae defined in the Squale model have been determined closely with developers of Air France and Qualixo. For each formula, the experience on concrete projects and comments has been integrated in the model.

Figure 2 shows an example of a dashboard for a class. The dashboard summarizes the individual marks of the class for different practices in the first pane and the values computed for different metrics in the second pane. It allows developers to obtain an accurate view of the quality of this class and help developers to determine if it must be improved.
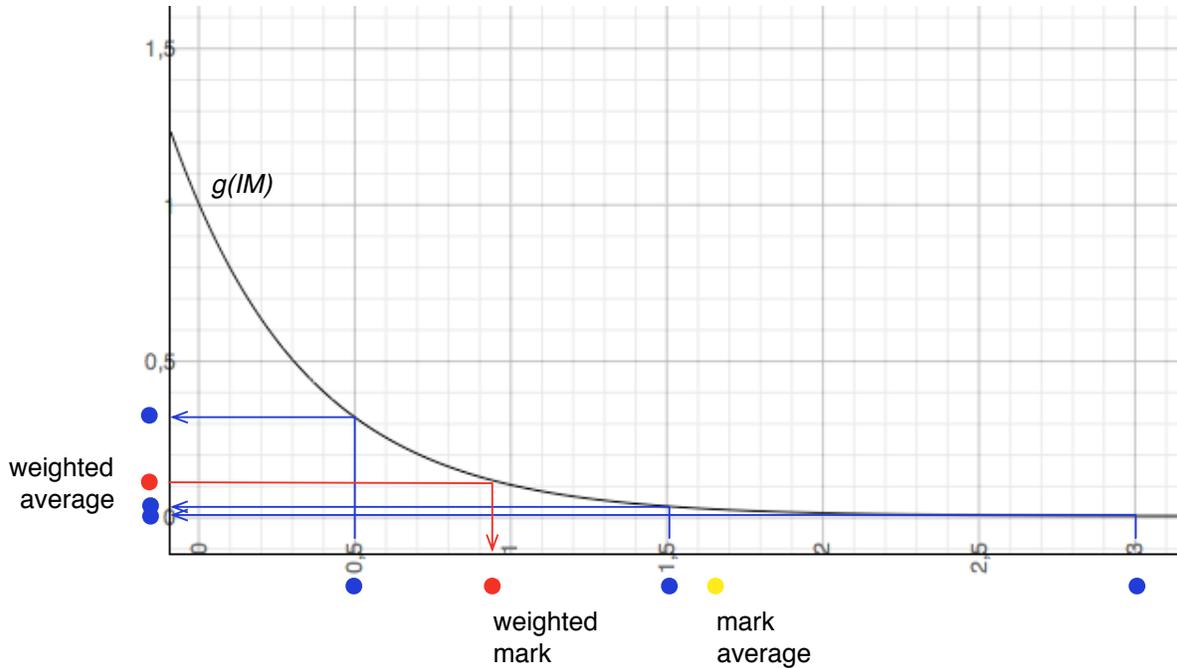
**Figure 4. Computation of practice mark: individual marks are translated in the weighted space where low marks have more weight; the weighted average is then translated back in the original space to give the weighted mark, significantly lower than the normal mark average.**

3.2 Practice mark

The global practice mark is obtained from the individual marks through a weighted average. The weighting function allows one to adjust individual marks for the given practice in order to stress or loosen tolerance for bad marks:

- a hard weighting is applied when there is a really low tolerance for bad individual marks in this practice. It accentuates the effect of poor marks in the computation of the practice mark. The global mark falls in the range $[0; 1]$ as soon there is a few low individual marks.

- a medium weighting is applied when there is a medium tolerance for bad individual marks. The global mark falls in the range $[0; 1]$ only when there is an average number of low individual marks.

- a soft weighting is applied when there is a large tolerance for bad individual marks. The global mark falls in the range $[0; 1]$ only when there is a large number of low individual marks.

Weighting is chosen to highlight critical practices: hard weighting leads to a low practice mark much faster than soft weighting.

The computation of the practice mark is a two-step process. First a weighting function is applied to each individual mark:

$$g(IM) = \lambda^{-IM}$$

where IM is the individual mark and λ the constant defining the hard, medium, or soft weighting. This formula translates individual marks into a new space where low marks have significantly more weight than others. The average of the weighted marks will reflect the more important weight of the low marks. Then the inverse function $g^{-1}(IM) = -log_\lambda(IM)$ is applied on the average to come back in the range $[0; 3]$.

Thus the global mark for a practice is:

$$mark = -log_\lambda \left(\frac{\sum_1^n \lambda^{-IM_n}}{n} \right)$$

where λ varies to give a hard, medium, or soft weighting.

Figure 4 illustrates how the $g(IM)$ function and its inverse works to reflect low individual marks in the practice mark. There are three individual marks (blue dots on the x axis) at 0.5, 1.5, and 3. This series gives a normal average around 1.67, above two of the marks. Instead, the marks are translated in the weighted space (red arrows) where the 0.5 mark is significantly higher than the two other marks.

Figure 5. Dashboard view of (global) practice marks and bad components for this practice.

The weighted average (red dot on y axis) is then translated back in the mark range with the value of 0.93. The lower weighted mark for the practice, compared to the normal average, is a clear indication that something is wrong, despite the high mark of 3.

3.3   Practice role during development

A practice is considered as "a process for elementary quality" to respect or to avoid for developers. Through a formal definition tuned according to company standards, practices enable to bridge the gap between the developer's point of view and the leader's point of view. The low-level quality indicators interpreted in term of practices can be understood by both developers and managersFurthermore, they constitute a guideline for the developers to correct their code and obtain a project following the quality standards of their company.

The global practice mark represents a comparative reference for each individual mark and allows one to focus on low individual marks with respect to the practice. For example, the *inheritance depth* practice mark is the weighted average of the mark of each class. The mark computed for a class could be easily compared with the mark of the practice to determine if this class is in the average of the project or abnormally high—the weight applied to the metric possibly strengthening this abnormal result.

Figure 5 shows an example of the dashboard for the *afferent coupling* practice. It gives the mark for this practice and the distribution of their individual marks. The third window details the three worst components for this practice with their name, their individual marks and their metric values. This dashboard allows one to highlight these bad

elements.

Furthermore, if a practice has a low mark, its definition determines also the diagnosis to improve it. Let us take an example with the *comment rate* practice. A low mark for this practice means that there are globally not enough comments in the source code. Moreover, due to the cyclomatic complexity metric used in the definition of this practice, individual marks give us more indications: we can determine which method needs more comments. Indeed, the methods with the lowest marks for this practice are not well commented with respect to their complexity—it is not the ones that simply have the lowest ratio of comment lines per lines of code. Simple metrics cannot provide this indication. It is the practice—computed from an adequate combination of metrics—that makes sense.

Table 3 shows a list of some practices defined in the Squale model. These table constitutes an example of what kind of practices our model exploits.

3.4   Adaptability of the practices

The Squale model defines principles for factors-criteria-practices-measures structures and for the different formulae, especially at the practice level. Practices are based on measures and these ones depend on paradigm and technologies within the development context. For example, metrics dedicated to object-oriented programming do not apply to Cobol programs. In the same way, practices depend on the availability of measures since they depend on the tools the company owns.

Furthermore the actual set of practices used in the Squale model, along with the formula and weights, depends on the type of project and the quality standards for the company.

Practice name	Definition
Inheritance depth	qualify the use of inheritance in an object-oriented project.
Comment rate	qualify the comment rate in regard of the complexity.
Method size	qualify the size of methods.
Swiss army knife	This Practice search for the utility classes which are often very difficult to maintain. This classes are generally child or parent less, with few attributes but very much methods.
Class cohesion	qualify the relation between methods and class.
Efferent coupling	compute the efferent coupling for a class. Analyze the dependance between one class and the other classes.
Afferent coupling	compute the number of classes which depend on the studied class.
Spaghetti code	qualify the complexity and the structure of code for highlighting complex code.
Dependency cycle	detect the packages cycle for highlighting a bad packaging or a poor design.
Layer respect	determine the level of layer respect compared to the initial project. Compute the number of transgression.
Naming standard	Determine the level of compliance for naming rules for the project.
Quality Assurance Plan	Verify if there is a Quality Assurance Plan accorded with the methodology of the enterprise.
Functional specifications	Verify if there is functional specification file for the project. Qualify this file.
Documentation quality	Qualify the quality of technical documentation in according to the requirements of the enterprise. This documentation allows developers to understand quickly the code. This practice look for comments in code and detect the lines of code in comments.
Integration test coverage	Qualify the level of integration test coverage.
Functional limits testing	Qualify the Functional limits tests.

Table 3. Examples of practices defined in the Squale model

For example, Air France does not use the same set of practices for its information system than PSA (although most are shared).

The Squale model is adaptable: it is customized for each project it is applied to, the weights applied to measures and practices are refined in an iterative and interactive process with the project team. Such iterative process is important since it makes sure that the team project understands and agrees the practices with the global objectives.

4 Industrial evaluation

The Squale model was first designed by the Qualixo company and Air France in 2006. After several months of experiments and validation of successive versions, they implemented the Squale quality platform. Since 2008, the Squale model is being reviewed by a French research consortium to enhance it [4] and the Squale quality platform is now released as open source software[2].

The Squale tool can monitor projects by applying its model to the collected measures. This software allows navigation between different screens showing global marks for factors, criteria, and practices, as well as individual marks and measures for each relevant element and practice. Figure 6 shows a reporting of Squale with the factor marks obtained by the Squale software itself ⟨Stéf⟩ ►argh not to itself
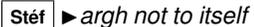
or say that due to nondisclosure agreement we cannot show Air France code results◄ . Each factor is detailed with: its marks for the previous evaluation; its mark for the current evaluation; a meteorologic symbol which gives a symbolic meaning to the mark and an arrow whose direction indicates the change with respect to the previous evaluation.

The validation of the Squale model is based on industrial feedback from Air France and PSA. One hundred projects are currently monitored by Squale at Air France, including business applications for freight or marketing, management applications for personnel management, or technical applications like frameworks. Of these hundred monitored projects, twenty are actively using it to improve their code base, which led to $6,000$ increased marks during one year. On the whole, Squale monitors about seven MLOC.

The Squale software has also been in use by PSA for nearly one year. It monitors around 0.9 MLOC dispatched in ten Java applications: two frameworks, seven business applications (marketing and manufacturing applications) and one component library. The most important application supports the coordination of the flow of vehicles in factories. Its size is near 200 KLOC.

In these companies, the Squale model is well accepted by developers as well as by managers which show interest in the model results. They noted an improvement of the quality for some projects but we cannot yet quantify this improvement, since the Squale project is still in an early

Projet: squalix

Facteur	Note précédente	Note	Evolution
Architecture	1.9	1.9	→
Capacité fonctionnelle	-	-	-
Evolutivité	2.0	2.0	→
Maintenabilité	2.2	2.2	→
Fiabilité	-	-	-
Réutilisabilité	2.0	2.0	→

Figure 6. The Squale audit view.

stage of deployment from an industrial perspective.

5   Related Work

Hierarchic quality models like the ISO 9126 Standard [9] or the McCall model [15] give an overall quality assessment of a system but they don't describe enough the low-level details and metrics needed to qualify this quality. Such models are top-down driven but clearly lack the connexion with source code. Another difficulty with these models is that they fail to translate the influence of individual components.

ISO 9126. ISO 9126 [9] is an international standard for the evaluation of software quality. It is the normalization of several previous attempts. It presents a set of six general characteristics that gives a software quality overview: functionality, reliability, usability, efficiency, maintainability, portability. Each characteristic is divided into subcharacteristics. It offers a top-down look on software quality and seems to be a good consensus to represent the overall quality since it is understood by end-users as well as project managers. This approach gives a standardized model but does not take into account all aspects of quality [1] and does not specify enough how to determine the factors which compose the model. The main question about this model is how such high-level factors can be linked with low-level metrics [12]?

McCall Factors Criteria Metrics (FCM). McCall [15] has defined a model called factor-criteria-metric to express the quality of a system. He identified 50 factors and selected the 11 most important ones which should represent the external vision of the quality. These factors are characterized by 23 criteria which represent the internal vision of the quality: the programmer's point of view.

This model is complete but very difficult to apply because of the 300 metrics needed to compute it. It is imple-

mented in several commercial tools but the correspondence between metrics and criteria is not clearly defined as already reported by Marinescu and Ratiu [12]. An important weakness is the lack of connexion between a criterium and the potential problem it reflects. When a criterium has a poor mark we don't know exactly what the cause of the problem. Even if the criterium is computed with a single metric it does not give the solution to improve the quality. And when the criterium is computed with several metrics, it becomes very difficult to determine how to remedy to the problem. The Squale model inspired by the ISO 9126 and the McCall model keep the advantage of the overall view of the quality but bring a new dimension of this kind of model witch allow to keep all the details: practices give in the same time the quality of the project and the way to improve this quality.

QMOOD The Quality Model for Object-Oriented Design (QMOOD) model is also a hierarchic model based on ISO 9126. He is composed by four levels: design quality attributes, object-oriented design properties, object-oriented design metrics and object-oriented design components. These high-level attributes are assessed using a set of empirically identified and weighted object-oriented design properties [3]. This model is made for object-oriented program and does not qualify any other paradigm. Furthermore, it qualifies "only" the object-oriented design: it does not care about the quality of implementation or if the rules of programming are respected for example.

Factor-Strategy. Marinescu and Ratiu [12] raised the following question *How should we deal with measurement results?* and propose to bridge the gap by linking quality factor to source code entities using detection strategies. They introduce detection strategies [11] as a generic mechanism for analyzing a source code model using metrics. The use of metrics in the detection strategies is based on mechanisms for filtering and composition. Based on the detection strat-

9

egy mechanism, a new quality model is proposed, called *Factor-Strategy*, using a decompositional approach. This model is relevant to measure object-oriented design but as the QMOOD model, it does not define the overall quality of a project. The adaptability of the Squale model allows to qualify any paradigm and practices provide a complete view of quality.

Assessment methodologies for free/open source software have started to emerge: OSMM, OpenBBR, QSOS, QUALOSS [7]. Those methodologies are based on ISO 9126 model and deal with the specificity of free/open source projects and as such broaden the scope of their model to include community-related attributes.

6 Conclusions and Perspectives

This paper presents the Squale model for software quality. Our model is inspired by the ISO 9126 standard. It introduces a new level for the assessment of practices in the hierarchy of factors, criteria, and measures. Our model is based on concretely defined and computable measures, which are combined to define the practices. The practices are the focal points around which the low-level needs of developers meet the top-level quality requirements of managers. This way, the Squale model gives both developers guidelines to improve the quality of a project—practices— and managers means to detect quality defects—criteria and factors—at an early stage of the project development.

The Squale model allows one to determine the quality of a project and control its evolution during the maintenance of a project, preventing deterioration. Moreover, using this model during the development of a project allows one to improve its quality. The Squale model stresses bad quality instead of averaging the quality in order to quickly focus on the wrong parts. It uses a set of measures combined into practices, formulae and weights to take into account the standards of the company and the technical specificity of a project. Practices and weights are customized with respect to these overall constraints. Air France-KLM and PSA Peugeot-Citroën have validated their own instances of the Squale model to monitor different information systems.

Since 2008, the Squale project assembles the Qualixo company, the Paqtigo company, Air France-KLM, PSA Peugeot-Citroen, INRIA and the University of Paris 8[3]. It aims to formalize new practices and a Squale metamodel which would decrease the time spent while customizing the

model. Moreover, the Squale model, due to its origin, is biased towards information systems. It does not suit projects concerning embedded softwares for example. Which measures and practices would be useful for a different domain remain to be done.

In future work we will study how the Squale model can be used to automatically describe a remediation plan to increase the quality of a project. Such a remediation plan should also assess the return on investment. It will provide strong arguments for managers dealing with quality process in their company.

References

[1] H. Al-Kilidar, K. Cox, and B. Kitchenham. The use and usefulness of the iso/iec 9126 quality standard. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 7 pp.–, Nov. 2005.

[2] F. Balmas, A. Bergel, S. Denier, S. ducasse, J. Laval, K. Mordal-Manet, H. Abdeen, and F. Bellinguard. Software metrics for java and cpp practices, v1, 2009.

[3] J. Bansiya and C. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28(1):4–17, Jan. 2002.

[4] A. Bergel, S. Denier, S. ducasse, J. Laval, F. Bellinguard, P. Vaillergues, F. Balmas, and K. Mordal-Manet. Squale – software quality enhancement. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR 2009), European Projects Track*, March 2009.

[5] L. C. Briand, J. W. Daly, and J. Wüst. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering: An International Journal*, 3(1):65–117, 1998.

[6] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[7] J.-C. Deprez, F. Monfilsc, M. Ciolkowskf, and M. Soto. Defining software evolvability from a free/open-source software. pages 29–35, Oct. 2007.

[8] N. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, London, UK, second edition, 1996.

[9] ISO/IEC. Iso/iec 9126-1 software engineering -product quality- part 1: Quality model, 2001.

[10] M. Lorenz and J. Kidd. *Object-Oriented Software Metrics: A Practical Guide*. Prentice-Hall, 1994.

[11] R. Marinescu. Detection strategies: Metrics-based rules for detecting design flaws. In *20th IEEE International Conference on Software Maintenance (ICSM'04)*, pages 350–359, Los Alamitos CA, 2004. IEEE Computer Society Press.

[12] R. Marinescu and D. Raţiu. Quantifying the quality of object-oriented design: the factor-strategy model. In *Proceedings 11th Working Conference on Reverse Engineering (WCRE'04)*, pages 192–201, Los Alamitos CA, 2004. IEEE Computer Society Press.

[13] R. C. Martin. Stability, 1997. www.objectmentor.com.

---

[3]This project is supported and labelled by the "Systematic - PARIS Region" competitive Cluster, and partially funded by Paris region and the DGE ("Direction Générale des Entreprises") in the context of the French Inter-ministerial R&D project 2006–2008 ("Projet R&D du Fonds Unique Interministériel").

[14] T. McCabe. A measure of complexity. *IEEE Transactions on Software Engineering*, 2(4):308–320, Dec. 1976.

[15] J. McCall, P. Richards, and G. Walters. *Factors in Software Quality*. NTIS Springfield, 1976.