

# Les modèles « SQALE » pour l'évaluation de code source logiciel



Pouvoir évaluer de manière objective la qualité du code source d'une application logicielle est un avantage considérable. La méthode "SQALE" (Software Quality Assessment based on Lifecycle Expectations) d'évaluation de code source logiciel développée par inspearit offre :

- une excellente visibilité et une synthèse homogène sur la qualité de l'applicatif,
- un diagnostic quantitatif objectif applicable de manière générique,
- un hiérarchisation des constats permettant l'optimisation du plan de correction,
- une mise en œuvre simple et rapide par son automatisation,
- une complémentarité naturelle avec les méthodes d'évaluation de processus.

La méthode "SQALE" présentée ici comporte l'application de deux modèles originaux - un modèle qualité homogène et hiérarchique et un modèle d'analyse basé sur le concept de remédiation.

## Modèle Qualité et Modèle d'analyse

Le besoin de connaître, d'évaluer la qualité du code logiciel dont on a demandé ou payé le développement n'est pas nouveau. Lorsqu'on recette une application, en complément de tests fonctionnels et de performances, on aimerait bien pouvoir qualifier facilement et de façon précise d'autres attributs de la livraison. On aimerait, par exemple, savoir si le code supportera facilement les demandes d'évolution qui ne manqueront pas d'apparaître, ou s'il sera facile de faire assurer sa maintenance par une équipe tierce.

Bien que certains grands donneurs d'ordre aient développé leurs méthodes, voire leurs propres outils pour satisfaire ce besoin d'évaluation, ces pratiques n'ont pas été généralisées faute de méthode standard permettant la systématisation de ce type d'évaluation dans des délais et des coûts réduits.

Cette absence de méthode pour évaluer des logiciels est ressentie aussi par les responsables de développement de logiciel. Il leur manque des moyens standards et simples pour évaluer la qualité du code logiciel et piloter leurs projets selon les trois axes majeurs qui sont la qualité, les coûts et les délais.

Plus généralement, si l'on prend une organisation qui développe des logiciels, on peut positionner ses capacités et la maturité de ses processus sur une échelle étalonnée comme celle du modèle standard CMMI. En ce qui concerne le code logiciel qu'elle développe, il n'existe pas de système complet et calibré pour positionner, coter la qualité des codes sources qu'elle produit.

Pourtant, les modèles de bonnes pratiques de développement tels que le CMMI pour l'Acquisition<sup>1</sup> ou CMMI pour le Développement<sup>2</sup> requièrent de contrôler et de suivre la qualité des produits pendant toute la durée du projet de développement.

C'est le cas, par exemple, de la pratique SP 1.2 du PA (Process Area) PPQA (Product and Process Quality Assurance):

**Le CMMI permet d'évaluer les processus mis en œuvre par une organisation. Pour évaluer le code développé, il manque un standard qui soit concrètement applicable.**

« Objectively Evaluate Work Products and Services » et de la pratique SP 2.3 du PA SAM (Supplier Agreement Management):  
« Evaluate Selected Supplier Work Products ».

Ces pratiques seraient plus faciles à mettre en œuvre, s'il y avait une méthode standard et simple pour pratiquer ces évaluations de code source.

Les deux constituants principaux d'une méthode d'évaluation sont d'une part un modèle qualité et d'autre part un modèle d'analyse.

■ Le modèle qualité définit les caractéristiques qualités (appelés parfois aussi facteurs) attendues dans le contexte de l'évaluation et les décompose en sous caractéristiques (ou sous facteurs) plus détaillées. Cette décomposition permet d'en déduire les contrôles précis à effectuer lors de l'évaluation du produit.

■ Le modèle d'analyse précise les règles permettant de caractériser ou de noter les produits (ou leurs constituants) en fonction des mesures effectuées sur les points de contrôle identifiés dans le modèle qualité. C'est ce qui est nommé « model » ou « analysis model » dans la norme ISO/IEC 15939<sup>3</sup>.

En ce qui concerne les modèles qualité, les plus connus sont ceux de Boehm<sup>4</sup>, de McCall<sup>5</sup>, de l'ISO 9126.<sup>6</sup> Ils sont généraux et n'ont pas été spécialement développés pour l'évaluation du code source d'une application logicielle.

Lorsqu'on essaye de les utiliser pour effectuer des analyses de code source, on se heurte à un certain nombre de problèmes dont les principaux sont les suivants :

■ Mélange de caractéristiques fonctionnelles et externes avec des caractéristiques principalement intrinsèques.

■ Orientation du modèle sur des points de vue différents de celui d'un évaluateur

en charge de statuer sur la qualité d'un code source.

■ Non orthogonalité des caractéristiques.

Mais, ce qui représente la principale faiblesse de tous ces modèles, c'est qu'ils ne sont pas accompagnés d'une méthode détaillée et concrète permettant de noter une application selon les critères qu'ils établissent.

On se retrouve à devoir créer sa propre méthode d'évaluation. Par exemple on peut, pour chacune des sous-caractéristiques, demander à un panel d'experts ayant parcouru le code de donner une note entre 1 et 5. En faisant la moyenne des notes, on identifiera les sous-caractéristiques avec des bonnes moyennes et celles avec des moyennes faibles. On aura ainsi identifié les points forts et les points faibles du logiciel. On voit tout de suite que cette méthode est coûteuse, longue et peu reproductible.

Nous avons développé la méthode SQALE et ses 2 modèles pour satisfaire nos clients ayant besoin d'évaluer le code source d'applications stratégiques.

Afin de contribuer au travail collectif indispensable vers l'obtention d'un standard d'évaluation, nous décrivons ici les modèles SQALE que nous utilisons avec succès depuis plusieurs années.

## Le modèle qualité SQALE

Ce modèle possède quatre caractéristiques particulières :

■ Il est construit par une approche générique prenant en compte le cycle de vie du code source.

■ Il est organisé en niveaux successifs.

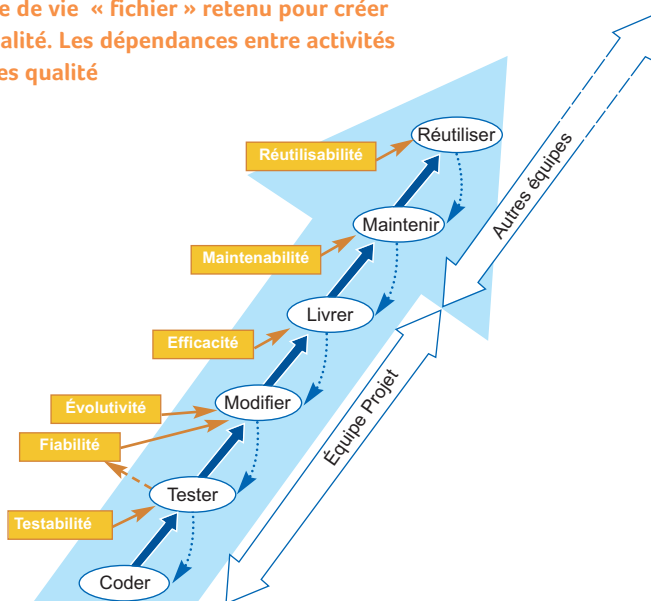
■ C'est un modèle d'exigences, et non une liste de bonnes pratiques à prendre en considération.

■ Il prend en compte les points de vue du réalisateur et celui de l'utilisateur du logiciel.

Pour déterminer les caractéristiques qualité de base de notre modèle, nous avons utilisé une démarche qui s'appuie sur le cycle de vie typique d'un fichier de code source logiciel. Nous avons retenu cette granularité parce que c'est celle usuellement prise pour attribuer un « propriétaire » dans les projets de développement. C'est aussi la granularité supportée par les outils de gestion de configuration de code et souvent celle utilisée pour la compilation. Quel que soit le cycle de vie général (cycle en V, cycle en spirale...) d'une application, un fichier va typiquement suivre une chronologie, un cycle de vie, illustré dans la figure 1.

Même s'il y a, au niveau macroscopique, des allers et retours entre différentes phases (par exemple entre le codage et le test), ce diagramme illustre une chrono-

**Figure 1 : Le cycle de vie « fichier » retenu pour créer notre modèle Qualité. Les dépendances entre activités et caractéristiques qualité**



gie générale et des règles de bon sens :

- On ne va pas livrer un code qui n'a pas été testé.
- Il y aura toujours des évolutions à prendre en compte ou des corrections à faire avant de livrer.
- On ne va pas réutiliser un code qui n'est pas maintenable.

Nous nous sommes demandés quels étaient les pré requis majeurs en termes de qualité pour chacune des étapes du cycle. Nous avons utilisé les caractéristiques et sous caractéristiques des 3 modèles précités, ce qui nous a permis d'établir les dépendances illustrées dans la figure 1.

Notons que nous avons rajouté la caractéristique «réutilisabilité» qui est absente du modèle ISO 9126.

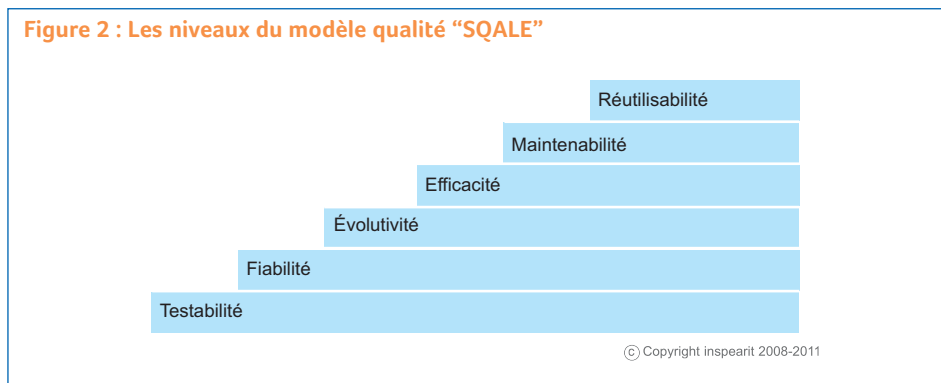
C'est ainsi que cette démarche positionne chronologiquement les attentes relatives à la qualité d'un code source logiciel. La première caractéristique étant la testabilité et la dernière étant la réutilisabilité.

On obtient au final un modèle en couches représenté en figure 2. Ce modèle de base peut être adapté en fonction du cycle de vie prévu pour le code. Notre démarche étant générique, elle permet par exemple, et si nécessaire, de rajouter des couches correspondant à d'autres attentes telles que la portabilité ou à la sécurité du code.

Partant de ces caractéristiques qualité (ou facteurs) de premier niveau, nous avons complété notre modèle par deux niveaux de détails complémentaires. Un niveau de sous-caractéristiques et un niveau correspondant à des points de contrôle sur le code. Pour ce faire nous avons utilisé l'approche classique Goal/Question/Metrics de Basili.<sup>8</sup> Les points de contrôle précis dépendent des situations, en particulier des langages et des environnements d'exécution du code.

Afin d'identifier les points de contrôle, nous nous sommes référés à la littérature existante sur le sujet et notamment à la taxonomie très complète établie par Diomodis Spinellis<sup>10</sup>. Les figures 3 et 4 présentent quelques éléments du modèle qualité complet. Il faut noter que, lorsque nécessaire, nous avons utilisé des sous caractéristiques correspondant à des sous activités bien précises du cycle de vie du

Figure 2 : Les niveaux du modèle qualité "SQALE"



logiciel. Ainsi la testabilité est décomposée en testabilité unitaire et testabilité d'intégration correspondant aux activités de test unitaire et de test d'intégration. De même la maintenabilité est décomposée en lisibilité et compréhensibilité car les activités de lecture/navigation et de compréhension sont considérées comme les activités principales d'un mainteneur amené à travailler sur un code développé par un tiers.

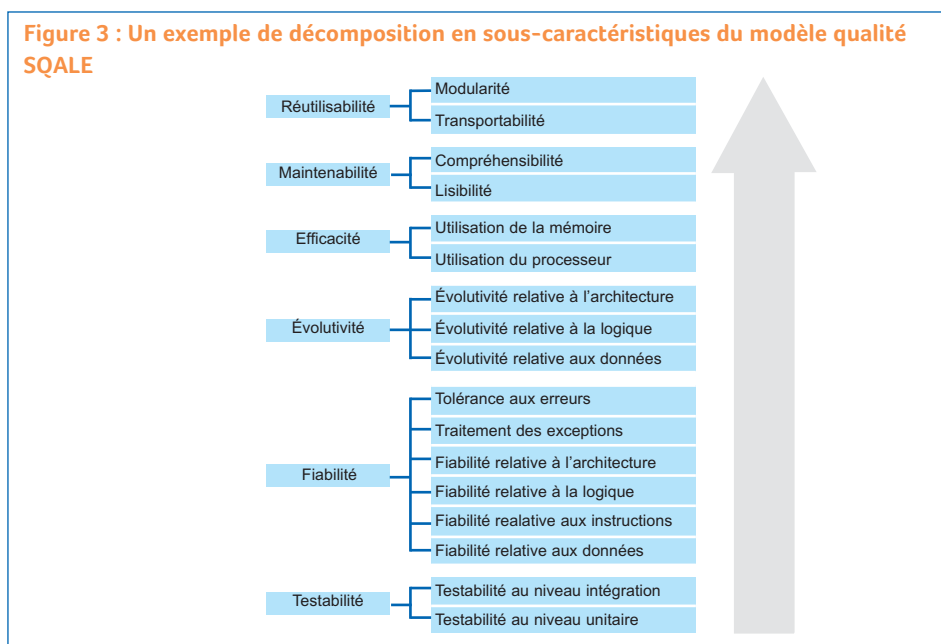
Dans cette construction, une sous-caractéristique ou un point de contrôle retenu dans le modèle n'est pas dupliqué dans une couche supérieure. Cette absence de duplication est un point essentiel du modèle qualité SQALE.

Ceci a pour conséquence, par exemple, qu'aucun critère lié à la testabilité n'apparaît dans la couche maintenabilité. Il ne reste plus que les sous-caractéristiques « lisibilité » et « compréhensibilité » associées à cette caractéristique. Les sous-caractéristiques telles que la testabilité ou l'évolutivité qui sont traditionnellement associées à la maintenabilité dans les modèles

qualité cités précédemment, constituent des couches plus basses du modèle SQALE. C'est la conséquence du fait que ces caractéristiques sont nécessaires plus en amont dans le cycle de vie d'un fichier de code source logiciel.

Le modèle qualité SQALE possède également la propriété importante d'être un modèle d'exigences. Chaque contrôle présent dans le modèle est une exigence qualité à satisfaire. Ceci est en conformité au premier « Concept absolu » défini par Ph. Crosby<sup>7</sup> «Quality must be defined as conformance to requirements. Consequently, the non conformance detected is the absence of quality ». Pour ce faire, les points de contrôles retenus dans notre modèle qualité sont des « must » caractérisant sans controverse les codes sources de qualité. Les exigences de base, ou point de contrôle sont, soit des métriques avec un seuil précis en deçà duquel le code n'est pas conforme, soit des règles de codage universellement reconnues pour leur bien fondé et n'acceptant pas d'exception.

Figure 3 : Un exemple de décomposition en sous-caractéristiques du modèle qualité SQALE



Également, les exigences proposées se doivent d'être non contradictoires entre elles. En effet, il doit être possible, en pratique, de livrer un code sans défaut, en regard des exigences retenues.

La dernière particularité du modèle qualité SQALE que nous avons cité est sa capacité à supporter deux points de vue, en l'occurrence, celui du « développeur » et celui du « propriétaire » comme illustré par la figure 5.

En lisant le modèle horizontalement, on peut analyser les non-conformités en fonction de la couche (caractéristique) ou de la sous-couche (sous-caractéristique) à laquelle elles correspondent et visualiser leur impact direct sur une activité de développement (test, prise en compte des demandes d'évolution...).

Figure 4 : Extrait d'une décomposition du modèle qualité SQALE en points de contrôle

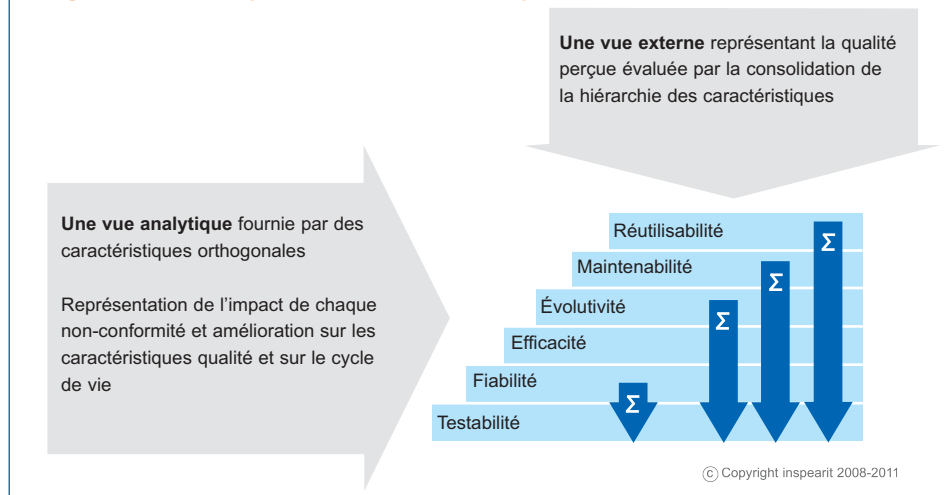
Maintenabilité	Compréhensibilité	Les méthodes ont une complexité essentielle $ev(G) \leq 5$
Maintenabilité	Compréhensibilité	Il n'y a pas d'instruction 'continue'
Maintenabilité	Compréhensibilité	Il n'y a pas d'instruction 'break' en dehors d'un block 'switch'
Maintenabilité	Compréhensibilité	Le taux de commentaire de chaque fichier est $> 25\%$
Maintenabilité	Lisibilité	Les accolades fermantes '}' sont seules sur une ligne
Maintenabilité	Lisibilité	Il n'y a pas d'instruction mise en commentaire

En lisant le modèle verticalement, on peut analyser les conséquences directes et indirectes des non-conformités aux exigences de bas niveau sur les caractéristiques telles que la maintenabilité ou la réutilisabilité.

## Le modèle d'analyse SQALE

Tout d'abord, illustrons par un exemple le fait que le modèle d'analyse est le point le plus délicat d'une démarche d'évaluation produit.

Figure 5 : Les deux points de vue du modèle qualité SQALE



Pour ce faire, imaginons qu'un chef de projet souhaite caractériser la maintenabilité de son application par une note globale.

Pour établir cette note, il faut agréger des notes de base. L'agrégation doit se faire selon deux dimensions :

- Selon la hiérarchie des éléments de l'application. La note globale de maintenabilité de l'application résultant forcément de la maintenabilité de chacun de ses éléments et sous éléments (par exemple de chacune des classes, elles mêmes constituées de méthodes).

- Selon le modèle qualité qui a défini les sous caractéristiques qui contribuent à la maintenabilité d'une application (en l'occurrence la lisibilité et la compréhensibilité).

nostics erronés et par conséquent amène à de mauvaises décisions.

En continuant sur l'exemple pris, on peut considérer que pour s'assurer de la maintenabilité de l'application, on va vérifier (sans que cela soit suffisant) que le code soit bien commenté. En supposant que dans le contexte, le seuil de commentaire ait été fixé à 25% par fichier. Si la moitié de l'application a un taux de commentaire de 15% et l'autre moitié un taux de 40%, un système d'agrégation basé sur la moyenne va donner un taux de commentaire de 32,5% et par conséquent une note acceptable pour la maintenabilité globale de l'application.

Malheureusement, et en pratique, ce n'est pas le surcroît de commentaire dans une partie du code qui aidera à maintenir la partie où les commentaires font défaut. Chaque partie, pour être maintenable, doit disposer de commentaires en quantité nécessaire et suffisante.

Le modèle d'analyse SQALE évite ce type de travers.

Comme nous l'avons précisé auparavant, notre modèle qualité est un modèle « d'exigences ». De par la manière dont il a été construit, la qualité visée correspond à une absence totale de non conformités et comme le précise Ph. Crosby, évaluer un code source peut se ramener à évaluer la distance qui le sépare de sa cible qualité.

Pour évaluer cette distance, nous avons défini et implémenté le concept d'index de remédiation. Cet index associé à chaque constituant du code (par exemple un fichier ou un composant) représente l'effort de remédiation qui serait nécessaire pour corriger les non conformités constatées par rapport aux exigences du modèle.

Comme le concept d'index de remédiation représente une charge de travail, la consolidation se fait par simple addition d'informations homogènes, ce qui constitue un avantage important. Un index de composant se calcule par addition des index de ses constituants. Un index de caractéristique se calcule par addition des index de ses sous-caractéristiques. Eux-mêmes sont calculés par addition des index associés à chaque point de contrôle concerné.

Les index de base sont attribués selon des règles qui respectent les principes suivants :

**Beaucoup d'évaluations de code reposent sur un système d'agrégation par moyenne. Cette approche n'est pas satisfaisante, elle produit des diagnostics erronés et par conséquent amène à de mauvaises décisions.**

■ Ils prennent en compte le coût unitaire de remédiation de chaque non-conformité. Coût qui dépend principalement de sa typologie.

Corriger un problème de présentation (mauvaise indentation, suppression de code mort) n'a pas le même coût unitaire qu'une correction qui implique la création et l'exécution de nouveaux tests unitaires, voir des tests d'intégration et des tests de non régression.

■ Ils prennent en compte le nombre d'occurrences des non-conformités. Par exemple un fichier qui a trois méthodes nécessitant d'être découpées car trop complexes aura un index triple qu'un fichier contenant une seule méthode trop complexe, toutes choses étant égales par ailleurs.

Finalement, les index de remédiation permettent de mesurer et de comparer à travers une mesure commune (de type coût) des non conformités d'origine et de type très différents. Des non conformités de type violation de règle de codage, de type dépassement de seuil pour une métrique ou présence d'un "anti pattern" pourront être comparées à travers leurs impacts relatifs sur la valeur de l'index.

## Utilisation concrète

Nous avons utilisé les modèles qualité et d'analyse SQALE pour effectuer de nombreuses évaluations de code source de toutes tailles et dans de nombreux domaines.

Que ce soit des applications Cobol, Java ou des applications embarquées en Ada ou C++, nous avons utilisé le même modèle qualité en couche.

Dans le cas des applications en Java, Ada et C++, le modèle qualité comprend, bien sûr,

Figure 6 : Graphe de synthèse des index de l'application entière

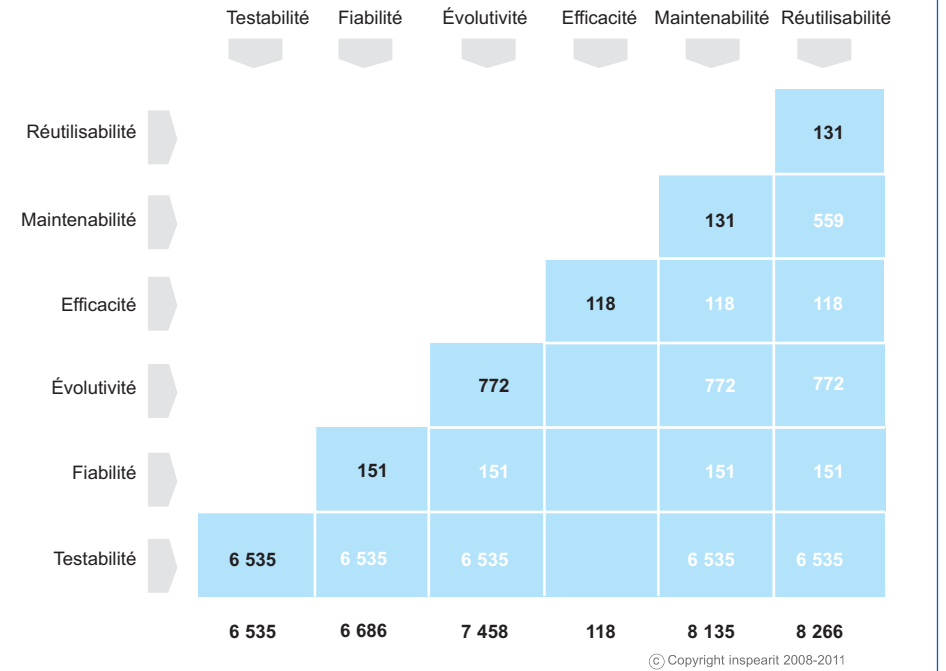
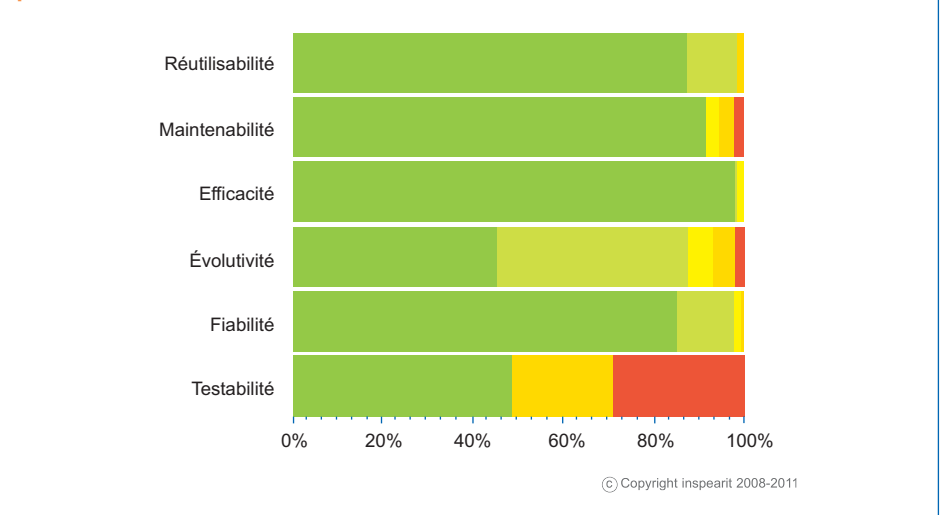


Figure 7 : Répartition des fichiers en fonction de leur index pour chaque caractéristique qualité



des points de contrôles à base de métriques orientées objet pour évaluer par exemple la testabilité, l'évolutivité, et la réutilisabilité.

Il comprend aussi le contrôle de l'absence d'"anti patterns" tels par exemple ceux identifiés par Brown<sup>10</sup>.

Les index sont calculés en fonction des coûts moyens de remédiation estimés par l'équipe de développement. Les seuils d'index permettant de faire une cotation en 5 niveaux (de mauvais à excellent) sont proposés par les responsables en charge de l'application.

Pour restituer efficacement les forces et les faiblesses du code évalué nous utilisons des graphes permettant de bien visualiser :

■ Dans quelle partie du code il y a le plus de risques.

■ Quels critères qualités sont les plus concernés.

■ Quel est l'impact direct des non conformités constatées.

Nous présentons ici (Figures 6 et 7) quelques uns des indicateurs que nous utilisons dans nos restitutions et qui selon notre expérience donnent une bonne visibilité et un bon pouvoir de décision à nos interlocuteurs chefs de projet.

Dans le cas de l'application présentée en exemple, l'index de testabilité est élevé. De ce fait, les autres caractéristiques

externes ne peuvent être satisfaites (ou approchées). Ceci malgré la relative bonne conformité aux points de contrôle relatifs aux caractéristiques de plus haut niveau. Le graphe présente un constat qui peut sembler paradoxal.

En effet, pour améliorer notablement la maintenabilité perçue par le « propriétaire » de l'application, il est préférable et plus efficace d'améliorer la testabilité que de rajouter des commentaires.

Les indicateurs d'analyse mentionnés ont permis de localiser précisément les modifications prioritaires à apporter pour augmenter la testabilité et par conséquent la qualité générale de l'application. L'analyse de la densité d'index (telle que présentée par exemple en figure 8) a aussi permis de comparer la distribution des non-conformités entre les fichiers de code réutilisés (R), modifiés (M) et créés (C).

## Conclusion

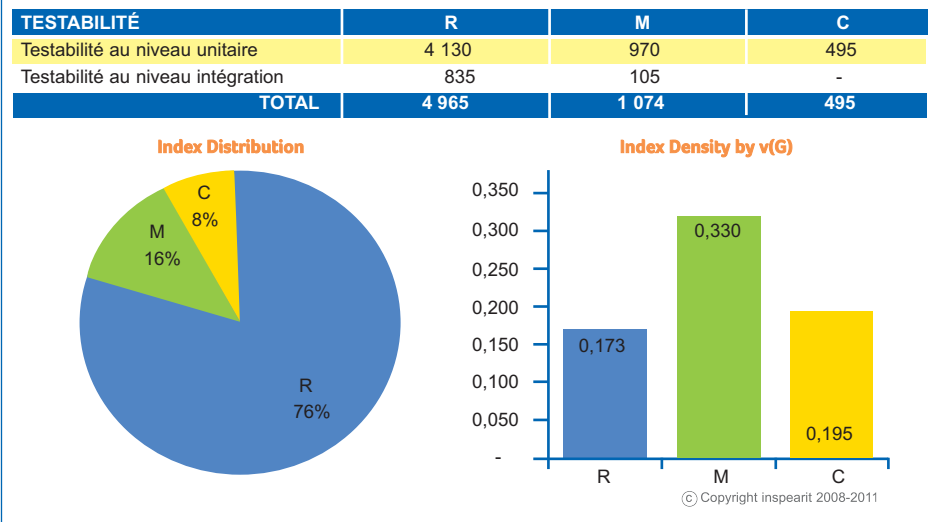
Plus qu'un modèle qualité, nous proposons une démarche générique pour développer des modèles qualité déduit du cycle de vie du logiciel et par conséquent permettant de bien spécifier les attentes relative à la qualité du code. Les modèles ainsi obtenus lèvent un certain nombre de faiblesses associées aux modèles faisant actuellement référence.

Les modèles qualité développés selon la méthode SQALE ont la particularité de présenter deux ressemblances avec le modèle CMMI. D'une part, comme celui-ci, ils organisent les exigences en niveaux correspondant à une logique de priorité.

D'autre part, comme le modèle CMMI que l'on peut utiliser pour évaluer la maturité ou la capacité, on peut utiliser les modèles qualité SQALE pour évaluer le code source de façon analytique ou selon le point de vue externe d'un utilisateur ou du propriétaire du code.

Ces similitudes des modèles et des méthodes d'évaluation permettent de mener rapidement et efficacement des évaluations conjointes processus/produit apportant une forte valeur ajoutée.

Figure 8 : Graphe de distribution des index et de leur densité



## Références

1. SEI, CMMI for Acquisition Version 1.2, CMU/SEI 2007-TR-017, Carnegie Mellon University, Pittsburg, 2007
2. SEI, CMMI for Development Version 1.2, CMU/SEI-2006-TR-008, Carnegie Mellon University, Pittsburgh, 2006
3. ISO/IEC, International Standard, «ISO/IEC 15939: 2007 (E), Systems and software engineering – Measurement process, 2007
4. Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merrit, M., Characteristics of Software Quality, North Holland, 1978
5. McCall, J. A., Richards, P. K., and Walters, G.F., Factors in Software Quality, The National Technical Information Service, No. Vol 1, 2 and 3, 1977
6. ISO, International Organization for Standardization, «9126-1:2001, Software engineering – Product quality, Part 1: Quality model», 2001
7. Crosby, P. B., Quality is free: the art of making quality certain, New York: McGraw-Hill, 1979
8. Victor Basili, Software Modeling and Measurement: The Goal/Question/ Metric Paradigm. Technical report. University of Maryland at College Park, 1992
9. Diomidis Spinellis, Code Quality The Open Source Perspective, ISBN: 0-321-16607-8, Addison-Wesley, 2006
10. Brown et al, Anti Patterns: Refactoring Software, Architectures and Projects in Crisis, ISBN:978-0-471-19713, John Wiley, 1998

## Le kit de découverte SQALE

**Découvrez les principes fondamentaux et les bénéfices de la méthode SQALE avec notre "kit de découverte SQALE". Ce kit contient :**

- Une formation d'un jour à la méthode SQALE
- L'identification (à travers quelques entretiens) des principaux "cas d'utilisation" de l'analyse de code dans le contexte de votre organisation
- Le développement (à l'aide d'ateliers dédiés) de vos propres modèles qualité et d'analyse. Ces modèles seront adaptés à votre environnement et serviront de références pour définir et évaluer la qualité de votre code source (pour l'un des langages suivants : Java, C, C++, C#, Cobol)
- L'évaluation concrète d'une de vos applications avec la méthode SQALE et vos propres modèles qualité et d'analyse (vous recevrez un rapport d'évaluation détaillé)
- Un atelier sur l'interprétation et l'utilisation des résultats.

**Les livrables de la prestation sont :**

- Votre modèle qualité et votre modèle d'analyse pour un langage de développement
- Un rapport d'évaluation
- Une liste d'actions directrices.

**Durée totale :** environ 20 jours



## À propos de inspearit

inspearit aide les entreprises et organismes des secteurs des services financiers, des télécommunications, des services publics, de la défense, de l'aérospatiale et de l'automobile à mieux gérer les risques informatiques en améliorant l'efficacité et la prévisibilité de leurs processus et logiciels, ainsi que la gestion de l'information. Basé à Paris, inspearit s'appuie sur une équipe de 300 consultants, ingénieurs et spécialistes de la formation répartis dans les principaux pays européens.

### Contact :

Alain Bloch  
alain.bloch@inspearit.com

Le Visium  
22 avenue Aristide Briand  
CS 90001  
94742 Arcueil Cedex  
France  
Tél : +33 (0)1 49 08 58 00