Chapitre 1

Modèles de mesure de la qualité des logiciels

1.1. De la qualité logicielle

De manière générale, un logiciel de qualité s'entend comme un logiciel capable de répondre parfaitement aux attentes du client, le tout sans défaut d'exécution. Ainsi, on détermine la qualité logicielle comme un ensemble de règles et de principes à suivre au cours du développement d'une application afin de concevoir un logiciel répondant à ces attentes[1] [ABR 04]. La NASA par exemple, a déterminé un ensemble de procédures, d'instructions de travail et de règles pour s'assurer que chaque étape du développement s'effectue de manière adéquate[2] [ESM 06]. La qualité d'un logiciel se reflète non seulement dans les processus de développement mais aussi dans la qualité des éléments qui le constituent, la documentation, la présence de tests, etc.

Mesurer la qualité d'un logiciel consiste alors à déterminer son adéquation par rapport aux objectifs de départ et aux standards de programmation. Il faut donc définir précisément ce que l'application doit faire et comment elle doit le faire, tant d'un point de vue fonctionnel que d'un point de vue technique. Une fois ces objectifs fixés, on peut alors appliquer un ensemble de règles et de mesures afin de calculer la différence entre objectifs attendus et réalisation obtenue.

Obtenir une mesure de la qualité permet à la fois d'avoir une image précise du logiciel mesuré mais aussi de déterminer le comportement de celui-ci dans le temps :

Chapitre rédigé par Karine MORDAL et Jannik LAVAL et Stéphane DUCASSE.

1. http ://www.swebok.org.

2. http ://sw-assurance.gsfc.nasa.gov/disciplines/quality/index.php.

quels sont les risques de bogues, les éventuelles failles sécuritaires, les difficultés de maintenance, les freins à l'évolution, la viabilité à long terme, etc.

Un des objectifs de la mesure de la qualité logicielle consiste à sensibiliser les équipes de développement sur leurs méthodes de programmation. En effet, mesurer la qualité a également pour objectif de fournir des bonnes pratiques de travail et des indicateurs permettant d'augmenter la qualité des futurs développements.

Pour obtenir une image complète de la qualité d'un logiciel on fait appel à un modèle de qualité. Celui-ci regroupe des règles qui décrivent ce que doit être un logiciel de qualité et le répertorie en différents groupes. Le modèle est ensuite évalué à partir de mesures obtenues grâce au code source, à la documentation, aux annexes techniques, aux règles de conception ou toute autre information disponible pour le projet.

Un modèle de qualité comporte souvent plusieurs niveaux de lecture. Il se compose d'une couche de haut niveau qui décrit la qualité selon un point de vue très généraliste. Ce premier niveau est ensuite décrit plus spécifiquement, le tout pour atteindre ensuite le plus bas niveau, détaillé et technique.

Dans ce chapitre nous présentons quelques modèles de qualité standards pour ensuite détailler le modèle Squale. Il s'agit d'un modèle de qualité *open source* développé depuis plusieurs années dans un contexte industriel avec Air France-KLM et PSA Peugeot-Citroën[3]. Ce modèle de qualité offre à la fois une vue générale de la qualité du projet mesuré mais également une vue détaillée orientée développeur. Il propose une manière d'agréger les données issues du projet de façon à ne perdre aucune information. Il permet également de passer d'une vue détaillée à une vue globale et *vice versa*. Squale est un modèle qui détermine la qualité d'une application mais il propose également un plan de remédiation conçu comme une aide à la décision.

1.2. Des outils pour mesurer la qualité

Un modèle de qualité se compose d'un certain nombre de règles et de principes, réparties en différentes catégories. Appliquer un modèle de qualité consiste à mesurer ces règles. Pour y parvenir, le modèle collecte un certain nombre d'informations issues du projet dont des métriques de code.

1.2.1. *Les métriques de code*

Il existe un nombre considérable de métriques et certaines d'entre elles ne sont pas calculées de la même manière selon l'outil utilisé. C'est pourquoi il convient de

3. http ://squale.org.

déterminer quelles métriques seront utiles et comment elles doivent précisément être calculées [BAL 09, FEN 96, JON 08, LAN 06a].

On peut toutefois classer les métriques en deux grandes catégories :

– les métriques primitives. Il s'agit des métriques qui mesurent des propriétés de base du code source telles que le nombre de lignes de code, la complexité cyclomatique ou encore les métriques de Chidamber et Kemerer [CHI 94] (profondeur d'héritage, nombre de sous classes, etc.) ou les métriques de Lorenz et Kidd [LOR 94] (nombre de méthodes, nombre de méthodes héritées, etc.) ;

– les métriques de design. Elles déterminent si le code source respecte les principes de conception préalablement définies. Il s'agit de métriques telles que les métriques de couplage ou de cohésion de classes, ou encore les métriques d'architecture de *packages* comme les métriques de couplage afférent et efférent de Martin [MAR 97].

De même, une application peut fournir d'autres sources de renseignements qu'il faut savoir interpréter dans le cadre d'une démarche qualité. Les modèles de qualité doivent donc fournir des indications précises sur l'utilisation et l'évaluation de ces informations. Il s'agit de donner du sens en termes de qualité à des informations brutes.

1.2.2. *Les principaux modèles de qualité hiérarchiques*

Les modèles les plus connus actuellement sont des modèles hiérarchiques qui recensent les principes de qualité en partant des exigences globales et des principes les plus généraux pour descendre vers les métriques qui permettent de les mesurer. Ceci implique que la mesure de la qualité ne peut débuter qu'une fois le modèle totalement spécifié et que les premiers résultats ne peuvent être obtenus qu'une fois la collecte des données suffisante.

1.2.2.1. *ISO 9126*

ISO 9126 [ISO 01] est une norme standard internationale visant à évaluer la qualité logicielle. Elle normalise et classifie un certain nombre de principes qualité. Réalisée par le comité technique JTC 1 de l'ISO/CEI, cette norme évolue pour être enrichie et intégrée dans la norme SquaRE (*Software product Quality Requirement and Evaluation*, exigences et évaluation de la qualité du logiciel). Elle est composée de six caractéristiques générales qui définissent la qualité globale d'une application : la capacité fonctionnelle, la fiabilité, l'efficacité, la maintenabilité, la facilité d'usage et la portabilité. Chacune de ces caractéristiques est décomposée en sous-caractéristiques.

Cette norme semble être une bonne approche pour déterminer la qualité d'un logiciel dans son ensemble et fournir une vue globale satisfaisante. Cependant, la norme ne précise pas de manière explicite comment mesurer les caractéristiques qualité définies et comment les relier aux métriques de bas niveau. Il n'y a aucun continium entre

ces deux niveaux. Ainsi, le modèle reste trop abstrait : bien que constituant une base théorique solide, le fait de devoir l'adapter à chaque cas de figure sans avoir de guide précis pour le faire augmente d'autant sa difficulté de mise en œuvre.

1.2.2.2. *SquaRE*

La norme SquaRE [ISO 05] définie depuis 2005 est la norme qui succède au standard ISO 9126. Elle a été définie à partir de ISO 9126 et de la partie évaluation de la norme ISO 14598.

Le système SquaRE décrit deux modèles distincts. Un modèle de qualité lié à l'utilisation du logiciel et un modèle de qualité propre à la production logicielle. Nous nous intéresserons ici uniquement à ce dernier.

Suivant le même principe que la norme ISO 9126, le modèle est constitué de huit caractéristiques, décomposées en sous caractéristiques :

– adéquation fonctionnelle : degré à partir duquel un produit offre les fonctions répondant aux besoins exprimés et implicites dans des conditions d'utilisation spécifiées ;

– performances : performances par rapport aux ressources utilisées dans des conditions déterminées ;

– compatibilité : degré à partir duquel un produit peut échanger des informations avec d'autres produits et/ou remplir ses fonctions, tout en partageant les mêmes environnements matériel et logiciel ;

– facilité d'utilisation : degré à partir duquel un produit peut être utilisé pour atteindre les buts identifiés, avec efficacité, efficience et satisfaction, dans un contexte spécifié ;

– fiabilité : degré à partir duquel un produit exécute les fonctions spécifiées dans des conditions spécifiées pour une période de temps spécifiée ;

– sécurité : degré à partir duquel un produit protège les informations et données de manière à ce que les personnes ou autres produits aient un accès à ces derniers qui corresponde à leur niveau d'autorisation ;

– maintenabilité : degré d'efficacité et d'efficience à partir duquel un produit peut être modifié par les personnes adéquates ;

– portabilité : degré d'efficacité et d'efficience à partir duquel un produit peut être transféré depuis un environnement matériel ou logiciel vers un autre, ou d'un usage à un autre.

Issue de la norme ISO 9126, la norme SquaRE redéfinit de manière beaucoup plus judicieuse les caractéristiques qualité d'un logiciel. Le fait, par exemple, d'avoir distingué la partie sécurité comme étant désormais une caractéristique à part entière, ou encore d'avoir fait une distinction entre la portabilité et la compatibilité rendent le modèle plus pertinent. Cependant, là encore, ce modèle généraliste est difficile à mettre

en œuvre ; il n'existe aucun lien explicite entre les caractéristiques et les métriques. De plus, ces normes définissent des modèles de qualité généraux qui visent à qualifier un logiciel dans son ensemble. Pour ce faire, ces modèles qualifient à la fois des notions externes – adéquation fonctionnelle ou encore facilité d'utilisation – avec des notions internes de qualification du code source à proprement parlé – maintenabilité ou compatibilité –, ce qui rend l'application du modèle dans son intégralité souvent beaucoup trop complexe à mettre en œuvre.

1.2.2.3. *Le modèle de McCall : Facteurs-Critères-Métriques (FCM)*

McCall [MCC 76b] a défini un modèle appelé Facteurs-Critères-Métriques pour évaluer la qualité d'un système. Il a identifié cinquante facteurs et a sélectionné les onze principaux représentant une vision externe globale de la qualité. Ces facteurs sont caractérisés par vingt-trois critères qui représentent la vision interne de la qualité : le point de vue du programmeur. Bien que complet, ce modèle est difficile à mettre en œuvre du fait des 300 métriques qui le composent. Il est toutefois implémenté dans quelques outils commerciaux mais la correspondance entre les métriques et les critères manque de clarté. Ce modèle présente également une faiblesse importante : le manque de lisibilité. En effet, lorsqu'un critère obtient une faible note, il est difficile, voire impossible de relier cette note directement au problème qu'elle pointe, surtout lorsque le critère est composé de plusieurs métriques. Dans ce cas, il devient difficile de trouver comment remédier au problème existant.

1.2.2.4. *GQM (Goal-Questions-Metrics)*

Il s'agit d'une approche de la qualité logicielle qui a été promu par Basili [BAS 94]. Il définit la mesure de la qualité sur trois niveaux : le niveau conceptuel (*the goal level*), le niveau opérationnel (*the question level*), et le niveau quantitatif (*the metric level*).

Le niveau conceptuel fixe les objectifs de mesure à savoir, ce qui doit être mesuré, le niveau à atteindre en termes de qualité, les objectifs visés par l'entreprise.

Le niveau opérationnel définit pour sa part quelles sont les questions à se poser pour déterminer si les objectifs définis au niveau précédent sont atteints.

Le niveau quantitatif détermine quelles métriques doivent être utilisées pour mesurer le niveau précédent.

Même si ce modèle est largement diffusé dans l'industrie, il pose tout de même le problème de ne pas expliciter clairement comment intégrer les stratégies et les buts spécifiques aux entreprises dans le modèle. De plus, ce modèle ne sépare pas clairement les différents points de vue entre les managers et les développeurs et ne permet donc pas une lecture claire systématique et aisée des résultats obtenus.

1.2.2.5. *QMOOD*

Le modèle *Quality Model for Object-Oriented Design* (QMOOD) est également un modèle hiérarchique basé sur la norme ISO 9126. Il est composé de quatre niveaux : les attributs de la qualité du design, les propriétés du design orienté objet, les métriques du design orienté objet et les composants du design orienté objet. Ces attributs de haut niveau sont évalués en utilisant un ensemble de propriétés empiriquement identifiées et pondérées [BAN 02]. Ce modèle est conçu pour des applications orientées objet et ne peut s'appliquer pour d'autres paradigmes. Plus encore, il ne qualifie que la conception des programmes : il ne prend pas en compte la qualité de l'implémentation ou le respect des règles de programmation par exemple.

1.2.2.6. *Facteur-stratégie*

Marinescu et Raţiu [MAR 04b] sont partis de la question suivante : comment pouvons-nous travailler en partant des résultats ? Leur idée est de relier clairement les facteurs de qualité au code source en utilisant une stratégie de détection. Ils définissent cette stratégie de détection [MAR 04a] comme un mécanisme générique permettant d'analyser le code source en utilisant des métriques. Les métriques sont soumises à des mécanismes de filtrage et de composition. Ils présentent un nouveau modèle de qualité appelé facteur-stratégie à partir de cette stratégie. Ce modèle est pertinent pour mesurer les concepts orientés objet mais tout comme le modèle précédent il ne peut s'appliquer pour d'autres concepts et ne couvre pas l'intégralité des propriétés d'une application.

1.2.2.7. *SourceInventory*

Bakota et Guymóthy ont présenté un modèle appelé *SourceInventory* [BAK 08] qui collecte des mesures telles que des métriques et des informations de couverture de tests. Ce modèle fournit une aide pour interpréter les données collectées mais demeure toutefois un modèle de bas niveau qui ne fournit pas une vue globale et de haut niveau de la qualité.

1.2.2.8. *The Squale Quality Model*

Le modèle de qualité SQALE est un modèle hiérarchique qui en suit les principes : trois niveaux allant du plus général – les caractéristiques – au plus détaillé – les mesures des points de contrôle. Les caractéristiques de Sqale ont été déterminées à partir d'un modèle générique du cycle de vie d'un logiciel. Elles sont représentées selon un modèle en couche qui implique que chaque caractéristique doit être validée pour passer à la suivante, tout comme chaque étape de développement d'un logiciel doit être validée pour continuer. Les évaluations des caractéristiques sont basées sur des index de remédiation calculant la distance entre le code analysé et l'objectif qualité à atteindre. Calculé pour chaque composant du code, cet index représente l'effort de remédiation nécessaire pour corriger le composant mesuré. L'index d'un composant

se calcule par addition des index de ses constituants. De même, l'index d'une caractéristique se calcule à partir de la somme de ses sous-caractéristiques.

Ce modèle a été créé pour mesurer la qualité du code produit et l'évalue en termes d'effort de remédiation uniquement. Il est particulièrement adapté aux développeurs pour lesquels il a été conçu. En revanche, il ne tient pas compte de la qualité fonctionnelle du logiciel.

1.3. Evaluer la qualité à partir des mesures

Les modèles pyramidaux attribuent des notes déterminant le niveau de qualité d'un logiciel en se basant sur des métriques brutes. Par exemple, la norme ISO 9126 définit la sous-caractéritique « facilité de modification » comme « la capacité d'un logiciel à intégrer de nouvelles implémentations ». Pour mesurer cette propriété, les métriques telles que le nombre de lignes de code (SLOC), la complexité cyclomatique, le nombre de méthodes par classe, la profondeur d'héritage (DIT) [BRI 98, FEN 96, LOR 94, MAR 97], sont combinées de manière à déterminer à partir de toutes ces mesures une seule et unique note pour cette caractéristique.

Les mesures utilisées sont définies et mesurées au niveau des composants du logiciel : la métrique SLOC par exemple est calculée pour une méthode donnée ou encore la métrique DIT calculée pour une classe donnée. De plus, chaque métrique possède sa propre échelle de valeurs. Déterminer une note de haut niveau pour mesurer une exigence qualité demande donc de résoudre deux problèmes :

– composer des métriques entre elles et qui ne sont pas définies de manière similaire (par exemple SLOC et la complexité cyclomatique) ;

– agréger les résultats des métriques afin de déterminer une note globale pour une caractéristique donnée (par exemple pour la caractéristique facilité de modification), tout en conservant l'information livrée par chaque note brute.

1.3.1. *Exploiter les mesures*

Pour obtenir une note de haut niveau pertinente il faut donc composer et agréger des métriques. En théorie, ces deux étapes peuvent être menées dans n'importe quel ordre. En effet, la composition de métriques peut s'effectuer à différents niveaux : pour chaque composant à partir de chacune des métriques obtenues ou au niveau du projet à partir de métriques déjà agrégées.

Cependant, il est plus pertinent de composer les métriques au plus bas niveau. Prenons l'exemple de l'exigence qualité taux de commentaires qui cherche à déterminer si le code est suffisamment commenté. Calculer une note uniquement au plus haut niveau ne possède pas de réel intérêt. En effet, pour être significative, cette exigence

qualité doit également pouvoir se lire au niveau de chaque méthode : ceci permet de déterminer directement la méthode qui ne correspond pas aux exigences. De plus, une méthode déficiente peut se retrouver masquée par d'autres méthodes sur-commentées lorsque l'on exprime le résultat uniquement au niveau du projet. Ainsi, calculer les notes au niveau du projet mais également au niveau des composants permet de déterminer précisément les composants déficients et les améliorations à apporter au code pour en augmenter la qualité.

L'étape de composition des métriques implique de tenir compte des intervalles de mesures de celle-ci. Par exemple, calculer le taux de commentaires d'une méthode implique d'associer la métrique de complexité cyclomatique avec la métrique de nombre de ligne de commentaires pour traduire le fait qu'une méthode complexe doit être mieux commentée qu'une méthode plus simple. Ramener ces métriques dans un intervalle de valeurs commun doit également prendre en considération le fait de garder le sens des métriques les unes par rapport aux autres. Pour y parvenir, la composition des métriques doit être élaborée spécifiquement pour chaque pratique calculée.

L'étape d'agrégation des mesures doit, elle aussi, être effectuée de manière à ne pas perdre les informations fournies au niveau de chaque composant. Comment faire ressortir le fait qu'un élément ne répond pas aux exigences de qualité lorsque l'on se situe au niveau le plus haut ?

Utiliser des notes globales pour définir la qualité pose également un problème crucial pour les développeurs : comment retrouver les informations livrées par les données brutes à travers une seule note globale ? Comment traduire cette note en un problème concret de conception/développement ? Cet écueil empêche nombre de développeurs de s'intéresser à un modèle de qualité dans son ensemble et ils lui préfèrent encore souvent les métriques de code brutes.

Dans le reste de cette section nous détaillons en quoi les méthodes de composition et d'agrégation utilisées le plus couramment ne permettent pas de répondre de manière satisfaisante aux exigences. Puis nous regarderons comment l'agrégation de métriques est abordée aujourd'hui dans la littérature scientifique.

1.3.2. *Les pièges des calculs classiques*

Pour composer des métriques et obtenir une échelle de valeur commune, une technique classique consiste à transposer de manière discrète les valeurs dans un intervalle commun. Cependant, les résultats obtenus ne sont pas satisfaisants comme expliqué dans la section 1.3.2.1.

De même, la méthode la plus communément employée pour agréger des métriques consiste à calculer une moyenne, simple ou pondérée. Cependant, même si utiliser des

Valeurs normalisées	3	2	1	0
Interpretation	Excellent	Acceptable	Problématique	Mauvais
SLOC	≤ 35	$]35; 70]$	$]70; 160]$	> 160

Tableau 1.1 – Un exemple de transposition discrète

moyennes semble être la réponse la plus évidente pour déterminer une note à partir de plusieurs résultats, ce n'est pas sans problème, comme expliqué dans la section 1.3.2.2.

Pour ces raisons, le modèle Squale utilise une approche différente pour calculer ses notes, dans le but de conserver un maximum d'information et de donner du sens à ses notes de haut niveau.

1.3.2.1. *Normaliser un résultat*

Transposer des valeurs quelconques dans un intervalle donné consiste le plus souvent à appliquer des transformations discrètes sur ce jeu de valeurs. Le tableau 1.1 montre un exemple dans lequel les valeurs cibles normalisées sont toujours 0, 1, 2 ou 3.

Un tel système présente l'avantage d'être très simple à implémenter et facile à lire mais il n'est pas adapté à toutes les mesures. Il constitue certes le meilleur moyen de traduire une expertise humaine – telles que celles qui constituent les mesures manuelles du modèle Squale – mais ne convient absolument pas pour traduire des métriques de code telles que le nombre de lignes de code ou la complexité cyclomatique.

Transposer des notes continues en intervalle discret pose les problèmes suivants :

– les modifications sont masquées. Utiliser des formules discrètes introduit des paliers et des effets de seuil, ce qui masque certains détails et peut fausser l'interprétation des résultats. De plus, lorsque l'on souhaite surveiller l'évolution de la qualité dans le temps, ces valeurs discrètes masquent les fluctuations plus faibles, dans un sens comme dans un autre. Par exemple, si l'on prend les valeurs du tableau 1.1, pour un projet donné contenant des méthodes d'une moyenne de 150 lignes de code, chaque méthode a une valeur normalisée de 1. Si les développeurs réécrivent certaines méthodes pour les porter à un nombre de 80 lignes de code, la qualité du projet aura alors globalement augmenté ce qui ne sera pas traduit dans la note globale qui reste inchangée du fait des transpositions discrètes appliquées ;

– une mauvaise influence sur les décisions de réingénierie. Travailler sur les composants dont la note est proche d'une valeur seuil nécessite moins de charge de travail et permet d'augmenter plus facilement la note globale de la qualité. En revanche, les composants dont la note est plus éloignée d'une valeur seuil nécessitent beaucoup

plus d'efforts pour que le bénéfice soit visible d'un point de vue note. Pourtant, d'un point de vue purement qualitatif et indépendamment de la note, il est plus judicieux de se consacrer aux plus mauvais composants pour augmenter réellement la qualité et corriger les problèmes réels.

Pour éviter ces écueils, le modèle Squale utilise des fonctions continues pour transposer les valeurs brutes des métriques dans un intervalle prédéfini.

1.3.2.2. *Agréger des métriques*

Pour fournir une représentation de la qualité à un niveau élevé, le modèle ISO 9126 s'appuie sur des métriques, comme décrit dans sa partie 3 [ISO 03]. Cependant, cette description ne donne aucune indication précise quant à la manière d'agréger les différentes métriques citées. Une moyenne simple ou pondérée reste souvent le moyen le plus utilisé pour y parvenir. Et pourtant, les moyennes ne donnent pas entièrement satisfaction puisqu'elles perdent de l'information comme cela est souligné par Bieman et d'autres [BIE 96, SER 10, VAS 10].

1.3.2.2.1. Moyenne simple

La méthode employée pour calculer une note globale sans perdre les informations fournies par les notes individuelles des composants du projet cristallise souvent les points faibles des modèles de qualité. Calculer une simple moyenne n'est pas assez précis puisque cela ne permet pas de déterminer l'écart-type d'une population comme illustré ensuite.

Le tableau 1.2 présente le nombre de méthodes par classe pour deux projets. Dans cet exemple, la moyenne du nombre de méthodes est de 12,75 pour le projet 1 et de 12,25 pour le projet 2, ce qui pourrait amener à croire que le second projet est de meilleure qualité que le premier (puisque la moyenne est moins élevée). Mais cette moyenne masque le fait que le second projet possède une classe A qui est très clairement en dehors des normes. C'est pourquoi, bien que la note moyenne soit meilleure, le détail des notes montre que ce second projet est pourtant le moins bon. La moyenne, parce qu'elle lisse les résultats, ne représente pas toujours la réalité [VAS 10]. Pour donner une note globale qui ait du sens, un modèle doit prendre en compte ses plus mauvais composants et refléter les écarts entre eux. Dans l'exemple précédent, un indicateur de qualité approprié devrait pointer du doigt le mauvais résultat de la classe A en fournissant une note globale plus basse. Mesurer la qualité ne consiste pas uniquement à calculer de simples moyennes mais doit mettre en avant les mauvais composants et les faiblesses d'une application. Pour être utile, un modèle de qualité doit être un modèle d'évaluation mais également un guide pour augmenter la qualité. Un développeur doit pouvoir connaitre les composants à améliorer et un manager doit connaître les points faibles du projet. Une simple moyenne ne pointe pas les mauvais composants et même pire : elle les masque.

Classe	Projet 1 # méthodes	Projet 2 # méthodes
A	13	35
B	12	5
C	14	5
D	13	4
Moyenne	12,75	12,25

Tableau 1.2 – Les moyennes simples de deux projets

Pour remédier à ce défaut, on peut décider d'utiliser une moyenne pondérée. Cependant, cette méthode a aussi ses défauts comme indiqué ensuite.

1.3.2.2.2. Moyenne pondérée

L'idée principale derrière l'utilisation d'une moyenne pondérée est de mettre en avant les mauvais composants et de détecter s'il existe des composants critiques. Intuitivement, il s'agit d'utiliser l'agrégation de métriques comme une alarme : donner une mauvaise note globale lorsqu'un composant est mauvais. Le poids est appliqué aux notes individuelles et représente l'influence de la note comparée aux autres. Une première version du modèle Squale mettait ce principe en application : plus une note était mauvaise, plus elle avait un poids fort.

Pour illustrer le problème attenant à une telle démarche, considérons l'exemple suivant : le tableau 1.3 décrit les poids donnés pour la métrique SLOC dans la première version de Squale. Il indique que les mauvaises notes obtenues pour cette métrique étaient pondérées avec une valeur de 27 pour augmenter leur influence lors du calcul de la moyenne. Le tableau 1.4 contient les poids correspondants aux mesures de SLOC. Par exemple, les poids appliqués pour la méthode C sont différents du fait de la note différente obtenue. Ce que cet exemple met en valeur et qui prouve que la méthode utilisée ne convient pas est illustré grâce aux méthodes B et C. Alors que la valeur de la métrique SLOC diminue et donc que la qualité augmente, les poids appliqués aux valeurs produisent un effet totalement inverse sur le calcul de la note globale : celle-ci diminue ! Dans cet exemple, la moyenne pondérée passe de 222,75 à 259,53 pour la seconde version. Le résultat augmente alors même que la qualité du code est globalement améliorée. Cet exemple montre que l'utilisation d'une moyenne pondérée n'est pas la méthode adéquate et peut même s'avérer totalement inappropriée pour un modèle de qualité : la moyenne diminue la note attribuée alors même que la qualité du code est en augmentation ! Un modèle de qualité doit refléter tous les changements le plus finement possible et avec fiabilité.

Note	≤ 35	$]35;70]$	$]70;160]$	> 160
Poids	1	3	9	27

Tableau 1.3 – Exemple de poids appliqués sur SLOC

version 1

Méthodes	SLOC	Poids
A	30	1
B	50	3
C	70	9
D	300	27
Moyenne simple/pondérée	112,5	222,75

version 2

Méthodes	SLOC	Poids
A	25	1
B	30	1
C	50	3
D	300	27
Moyenne simple/pondérée	101,25	259,53

Tableau 1.4 – Les moyennes de deux projets

1.3.3. *L'agrégation des métriques dans la littérature scientifique*

En dehors de l'utilisation des moyennes simples ou pondérées, la littérature scientifique décrit de nouvelles méthodes d'agrégation telles que des fonctions d'écart médian ou de déviation [BAR 09, LAN 06b, PER 07]. Cependant, ces fonctions, de part leur similarité avec les moyennes tombent dans les mêmes travers dès lors qu'il existe dans les mesures des résultats trop dispersés, ce qui est souvent le cas dans les logiciels d'ingénierie [TUR 11].

Récemment, un courant a émergé qui vise à utiliser des techniques d'agrégation différentes des moyennes empruntées à l'économie : les indices d'inégalité. Vasilescu [VAS 10], Vasa [VAS 09] et Serebrenik [SER 10] ont étudié des fonctions telles que le coefficient de Gini ou l'index Theil. Ces indices ont une propriété de décomposition qui leur permettent d'expliquer également la diversité de répartition des résultats [COW 95].

Une telle propriété est utile par exemple lorsque l'on mesure la répartition des inégalités de taille entre les classes d'une application organisée en *packages*. On peut

ainsi essayer d'évaluer si les classes disproportionnées sont concentrées seulement dans quelques *packages* ou réparties sur l'ensemble du système. Serebrenik [SER 10] a pu ainsi démontrer que les inégalités mesurées dans les tailles de fichiers sous Linux sont plus dues aux répartitions par *packages* qu'aux différents langages de programmation utilisés.

Néanmoins, les indices d'inégalité économétriques sont basés sur un certain nombre d'hypothèses valables pour les valeurs économiques comme le revenu ou le bien-être, mais pas nécessairement pour les métriques logicielles. Par exemple, les indices d'inégalité ne font aucune différence entres des valeurs toutes faibles ou toutes fortes [COW 00]. Par exemple, un logiciel dont les valeurs de complexité des composants sont toutes élevées aura la même note qu'un autre ayant toutes ses valeurs très faibles, ce qui ne signifie pourtant pas la même chose en termes de maintenance logicielle ! Un modèle de qualité efficace doit permettre d'alarmer les développeurs et mettre en valeur les problèmes potentiels.

1.4. Le modèle Squale

Le modèle squale est basé sur les mêmes principes que le modèle facteurs-critères-métriques de McCall [MCC 76b] et la norme ISO 9126. Il s'agit de modèles hiérarchiques qui adoptent une démarche descendante. En effet, la norme ISO 9126 se définit tout d'abord par six caractéristiques principales – détaillées ensuite par des sous-caractéristiques – tout comme le modèle de McCall se définit tout d'abord par ses facteurs puis les critères qui les composent. Les métriques viennent ensuite caractériser ces différents niveaux. *A contrario*, le modèle Squale se base sur les mesures disponibles à un instant donné pour calculer les niveaux supérieurs du modèle et fournir l'image de la qualité qui en découle. Le modèle Squale part donc de données brutes qu'il agrège pour donner une mesure de la qualité à un niveau plus général. Cette approche donne du sens à des mesures brutes qui ne sont lisibles et compréhensibles en l'état uniquement par des techniciens ou des experts du domaine. Cette démarche permet également de s'assurer que les calculs effectués pour définir la qualité sont toujours basés sur des mesures concrètes et identifiables. De plus, adopter une telle démarche permet d'obtenir une image de la qualité au plus tôt et de suivre son évolution tout au long de son cycle de vie.

1.4.1. *Squale : un modèle sur quatre niveaux*

La figure 1.1 schématise le modèle Squale. S'appuyant sur les sources et les documents de l'application, le modèle est composé de quatre niveaux : mesures, pratiques, critères et facteurs. Un niveau supplémentaire aux modèles de McCall et de ISO 9126 a été introduit entre les mesures et les critères : les pratiques. Ce niveau, situé au-dessus

des mesures, permet d'agréger les données brutes sous forme de règles à respecter ou à éviter pour développer une application de qualité.

Le modèle Squale est donc composé de quatre niveaux scindés en deux parties (figure 1.2) :

– une partie de haut niveau :

- un facteur représente le plus haut niveau du modèle. Il donne une vue globale de la qualité et s'adresse essentiellement aux non techniciens. Un facteur spécifie une exigence qualité globale, fonctionnelle ou non, des clients et des utilisateurs ;

- un critère représente un principe conceptuel de qualité. Il détaille et précise le facteur dans lequel il est défini et s'adresse aux managers comme un niveau détaillé de la qualité ;

- une pratique représente un principe technique de qualité. Les pratiques s'adressent directement aux développeurs en termes de bonnes ou mauvaises méthodes de programmation et lui fournissent un guide à suivre pour développer une application qui respecte les critères de qualité définis pour ce projet. Les pratiques permettent donc de déterminer clairement les règles de qualité importantes pour l'entreprise et de définir sans ambiguïté les critères et facteurs qui composent les couches supérieures du modèle.

– une partie de bas niveau :

- une mesure est une donnée brute collectée à partir des informations du projet. Ces mesures sont utilisées pour déterminer la qualité du projet et mesurer les règles définies par les pratiques.

Les critères et les facteurs qui composent les deux niveaux supérieurs du modèle donnent une vue globale de la qualité et s'adressent plus spécifiquement aux managers. Les pratiques et les mesures, quant à elles, ciblent plus spécifiquement les techniciens. Ainsi, le modèle Squale offre deux grilles de lecture différentes de la qualité et s'adresse à la fois aux développeurs et aux managers.

Le niveau inférieur du modèle contient des mesures brutes qui peuvent prendre n'importe quelle valeur : un nombre de lignes de code ou encore une profondeur d'héritage, etc. Le niveau supérieur, composé des pratiques, critères et facteurs utilise un intervalle de notation commun, comme le préconise la norme ISO 9126 [ISO 01]. Cette dernière précise que l'échelle de mesures associées aux métriques utilisées pour évaluer les exigences qualité peut être divisée en différentes catégories, par exemple, deux catégories – satisfaisant ou non – ou encore quatre catégories – dépasse les exigences, atteint sa cible, acceptable *a minima*, non acceptable. Ainsi, pour faciliter l'interprétation d'une note donnée et permettre une comparaison plus aisée, le modèle Squale est composé de notes comprises dans l'intervalle continu $[0; 3]$, qui se comprend selon l'échelle de valeurs suivante :

– entre 0 et 1, le but n'est pas atteint ;

– entre 1 et 2, le but est atteint mais avec des réserves ;

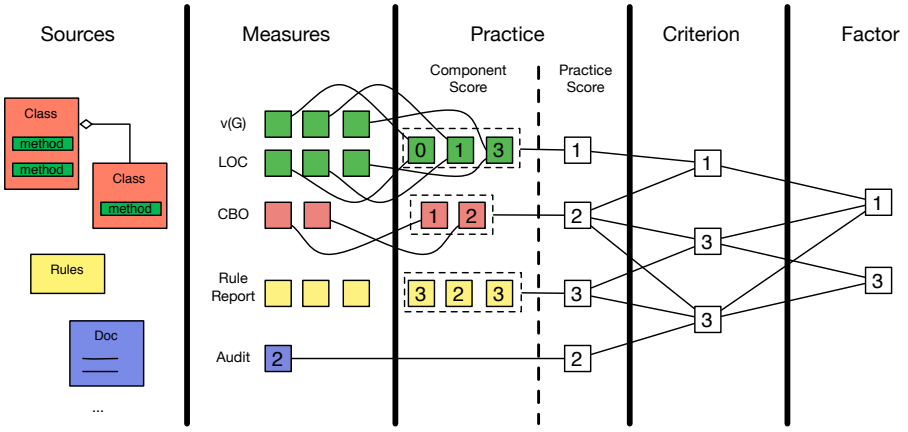

Figure 1.1 – Les données et les niveaux du modèle Squale

– entre 2 et 3, le but est atteint.

Le modèle Squale est donc composé d'une part de mesures de bas niveau et d'autre part de critères et facteurs de haut niveau (voir la figure 1.2). Chaque métrique livre une mesure dans sa propre échelle de valeurs tandis que les critères et les facteurs sont toujours notés dans l'intervalle [0 ;3]. L'étape de transformation des données brutes de bas niveau en notes de haut niveau s'effectue au niveau des pratiques. Elles constituent donc le point central du modèle Squale.

Les sections suivantes décrivent les quatre niveaux du modèles Squale, depuis le niveau inférieur des mesures jusqu'au niveau le plus élevé des facteurs.

1.4.1.1. *Mesures*

Une mesure est une information brute collectée au cours de l'analyse du projet évalué.

Le modèle Squale prend en compte deux principaux types de données pour mesurer la qualité d'un logiciel : les mesures automatiques calculées facilement et aussi souvent que nécessaire et les mesures manuelles auxquelles on attribue une longévité et qui ne sont redéterminées que lors de changements majeurs.

Les mesures automatiques se décomposent en trois grands groupes :

– les métriques [BRI 98, FEN 96, MAR 97] telles que le nombre de lignes de code [CHI 94], la profondeur d'héritage [LOR 94] ou la complexité cyclomatique [MCC 76a]. Ces métriques ont été sélectionnées pour ne garder que les plus pertinentes correspondant aux besoins de l'entreprise ;

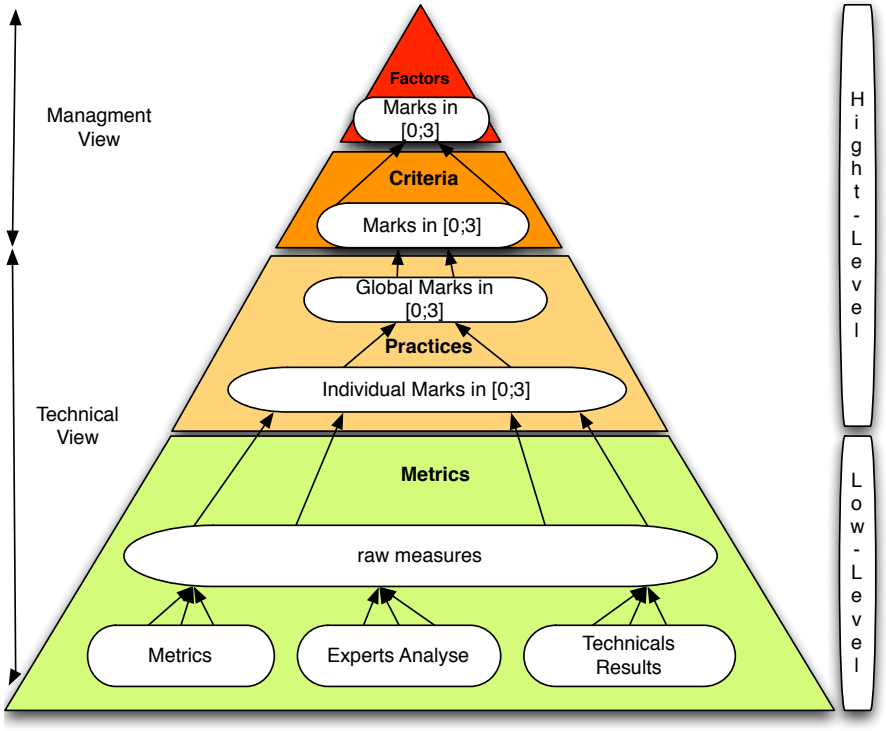

Figure 1.2 – La pyramide du modèle Squale.

– les conventions de développement (*rules checking*) telles que les règles de programmation, les règles syntaxiques. Il s'agit des règles mises en place par l'entreprise pour définir les standards d'écriture et de programmation ayant cours chez elle ;

– les métriques de tests qui permettent de savoir si l'application est testée correctement et si les couvertures de tests sont correctes.

Les mesures manuelles correspondent aux audits effectués par des experts. Ces mesures qualifient la documentation, le cahier des charges ou tout autre document nécessaire à la conception de l'application. Ces mesures englobent également la vérification du respect des contraintes définies au préalable : respect du cahier des charges par exemple.

Chaque mesure – automatique ou manuelle – est affectée à l'entité qu'elle mesure, c'est à dire la classe pour la profondeur d'héritage ou le projet pour la qualité du cahier des charges par exemple.

Les mesures possèdent des valeurs brutes qui varient selon leur propre échelle de valeurs. La métrique *SLOC*, qui détermine le nombre de lignes de code, possède en général une valeur n'excédant pas 1 000 tandis que la métriques *NOM* qui détermine le nombre de méthodes a une valeur inférieure à 100 en pratique, alors que la valeur de la métrique « instabilité » de Martin est toujours comprise entre 0 et 1.

En revanche, les mesures manuelles qui sont déterminées à partir des audits qualité sont directement évaluées dans l'intervalle $[0; 3]$. En effet, ces mesures sont reliées au projet dans son ensemble et définissent la note globale de la pratique qui est exprimée dans cet intervalle.

Actuellement, le modèle Squale dispose d'un nombre de mesures oscillant entre 50 et 100 selon le projet analysé, son étape de développement et les audits manuels effectués.

1.4.1.2. *Pratiques*

Une pratique s'assure du respect d'un principe technique issu du projet (par exemple les classes complexes doivent être plus commentées que les triviales). Elle s'adresse directement au développeur en termes de bonnes ou de mauvaises pratiques en respect avec la définition de la qualité attendue sur le projet. Les bonnes pratiques doivent être suivies et renforcées tandis que les mauvaises doivent être repérées et évitées. L'ensemble de ces pratiques définit des règles correspondant à l'optimum qualité visé.

Chaque pratique se définit par le principe technique qu'elle décrit, les différentes mesures qu'elle utilise pour le qualifier et les formules qu'elle utilise pour déterminer sa note. On distingue deux types de notes par pratique : la note individuelle donnée pour chaque composant du projet (chaque classe ou méthode par exemple) et la note globale obtenue par la pratique pour l'ensemble du projet. Ces calculs sont détaillés dans la section suivante.

Environ cinquante pratiques sont actuellement définies dans le modèle Squale mais la liste doit être établie en fonction des attentes des entreprises et des projets, au début de l'étude. Elle n'est donc pas fermée et peut être étendue selon les cas étudiés.

1.4.1.3. *Critères*

Un critère qualifie un principe de la qualité d'une application (sécurité, simplicité, ou modularité par exemple). Il s'adresse aux managers et leur propose une vue détaillée de la qualité du logiciel, sans toutefois être technique. Les critères définis dans le modèle Squale sont adaptés aux exigences des entreprises qui les utilisent. En particulier, l'instance du modèle actuellement utilisé par Air France-KLM et PSA Peugeot-Citroën est particulièrement adapté aux systèmes d'information.

Un critère agrège un certain nombre de pratiques. La note d'un critère correspond à une moyenne simple des pratiques qui la composent. Actuellement, le modèle Squale est composé d'une quinzaine de critères.

Par exemple, les pratiques suivantes : profondeur d'héritage, standards de documentation (respect des conventions de documentation), qualité de la documentation (audit manuel qui qualifie la documentation en respect avec les exigences du projet), spécialisation de la classe, taux de commentaires du code source (par méthode et par rapport à la complexité cyclomatique) définissent le critère compréhension pour l'instance du modèle utilisé par Air France-KLM et PSA Peugeot-Citroën.

1.4.1.4. *Facteurs*

Un facteur représente le niveau le plus élevé du modèle de qualité. Chaque facteur donne une vue globale de la qualité du projet pour un secteur précis (capacité fonctionnelle ou fiabilité par exemple) et reflète les exigences attendues par les clients.

Les facteurs agrègent les critères. La note d'un facteur correspond à la moyenne simple des notes des critères qu'il agrège.

Les six facteurs utilisés dans l'instance du modèle Squale déployée chez PSA Peugeot-Citroën et Air France-KLM sont inspirés des facteurs de la norme ISO 9126 mais redéfinis pour correspondre aux exigences et aux attentes de ces entreprises.

Par exemple, les critères suivants :

– compréhension,

– homogénéité,

– capacité d'intégration,

– simplicité,

définissent le facteur maintenabilité. Ce qui signifie qu'un système est plus facile à corriger – à maintenir – s'il est homogène (en respect avec les conventions de programmation et l'architecture générale), simple à comprendre et à modifier (une bonne documentation et une taille manipulable) et si son couplage est correct.

1.4.2. *Les notes calculées dans le modèle Squale*

Dans cette partie, nous allons présenter comment le modèle Squale résout les problèmes énoncés dans la section 1.3.2 et comment le modèle calcule les notes de haut niveau.

Notre modèle collecte différentes mesures et traduit celles-ci pour fournir une note globale de haut niveau comprise dans l'intervalle [0 ;3]. Pour calculer cette note, le

modèle Squale n'utilise pas de moyennes simples ou de transpositions discrètes mais des fonctions continues. Ces calculs sont effectués au niveau des pratiques, en deux étapes : la composition des métriques qui fournit un résultat par composants, puis l'agrégation des résultats obtenus qui détermine une note au niveau du projet. On distingue donc :

– les notes individuelles : la combinaison de métriques. Chaque élément (méthode, classe, *package*) concerné par une pratique donnée possède une note pour cette pratique. Cette note est calculée à partir des mesures effectuées sur cet élément. Par exemple, les deux métriques qui définissent la note de la pratique taux de commentaires, à savoir complexité cyclomatique et nombre de lignes de code, sont calculées pour chaque méthode du projet. Dans ce cas, la pratique taux de commentaires possède une note individuelle, calculée pour chaque méthode du projet à partir des métriques. Cette note prend en compte le poids d'une métrique par rapport à une autre et associe les métriques selon une formule définie. Cette note individuelle est donnée dans l'intervalle [0 ;3] pour permettre les comparaisons entre composants ;

– la note globale : l'agrégation des notes individuelles. Une note globale est ensuite attribuée pour une pratique donnée en se basant sur les notes individuelles obtenues pour chaque composant. Le modèle utilise une fonction continue pondérée pour calculer la note globale du projet pour une pratique donnée.

1.4.2.1. *Les notes individuelles*

Une note individuelle est calculée à partir des mesures obtenues pour un composant donné. Les mesures sont données dans des intervalles multiples tandis que la note individuelle est toujours comprise dans l'intervalle $[0; 3]$.

Pour transposer les mesures brutes en note individuelle, le modèle Squale utilise une formule continue afin d'éviter les effets de seuil et de traduire les variations de mesure au plus juste. Chaque pratique tient compte des spécificités des métriques utilisées et la fonction de combinaison des notes est déterminée pratique par pratique. Ces fonctions continues sont également ajustées aux besoins des entreprises qui utilisent le modèle. C'est pourquoi les fonctions utilisées se basent sur des valeurs seuils établies par des experts au sein des entreprises. A partir de ces valeurs, on définit une équation, linéaire ou non, qui interpole au mieux les valeurs intermédiaires.

L'exemple de la pratique taille de la méthode illustre ce mode calcul. Cette pratique est calculée au niveau méthodes, c'est à dire que les notes individuelles sont calculées pour chaque méthode et la note globale de la pratique est calculée avec une fonction pondérée et un poids moyen. Cette pratique se base sur le nombre de lignes de code pour être calculée. Elle est définie comme une pratique qui pointe les méthodes qui sont trop longues, et donc sans doute trop difficiles à maintenir et à comprendre.

Pour calculer la note individuelle pour cette pratique, des valeurs de référence ont été définies et sont données dans le tableau 1.5. A partir de ces valeurs, on définit la fonction de calcul de la note individuelle de la manière suivante :

$$IM = 2^{(70-sloc)/21}$$

Cette équation a été validée par les développeurs de Air France-KLM : les seuils sont définis à partir de leur prérequis. A noter que chaque instanciation du modèle Squale peut avoir ses propres valeurs.

SLOC	≤ 37	42	49	58	70	91	≥ 162
Pratique	3	2,5	2	1,5	1	0,5	0

Tableau 1.5 – Mesures/notes de références servant de base à la formule déterminant la note de la pratique

La figure 1.3 montre la courbe correspondant à cette fonction. Le seuil de 40 correspond à la note maximum, c'est-à-dire le seuil au-delà duquel la note est égale à 3. C'est la note maximum qui correspond au fait d'avoir atteint l'objectif souhaité. En dessous de ce seuil, les notes individuelles diminuent en suivant une courbe de forme exponentielle : les notes tendent très rapidement vers zéro.

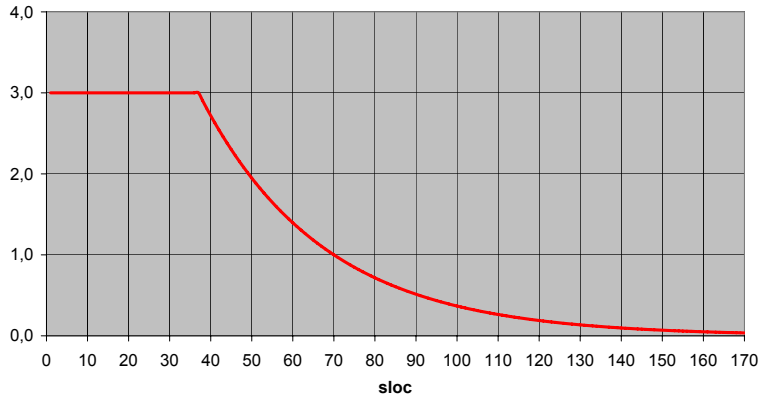

Figure 1.3 – Courbe de la fonction pondérée de la pratique taille de la méthode

1.4.2.2. *Les exceptions aux agrégations continues*

Dans le modèle Squale, certaines notes sont toutefois traduites de manière discrète dans l'intervalle [0 ;3] : il s'agit des mesures manuelles.

Par exemple, la pratique spécifications fonctionnelles applique cette notation :

– la note est de 0 s'il n'y a pas de spécifications fonctionnelle ;

– la note est de 1 ou 2 s'il existe des spécifications mais pas totalement satisfaisantes ;

– la note est de 3 si les spécifications fonctionnelles existent et sont conformes aux exigences du client.

Cette note est déterminée par un expert et délivre deux types d'informations : l'existence et la qualité des spécifications fonctionnelles. Dans les cas similaires, une notation discrète avec une description précise des notes limite la subjectivité de l'expert lors de l'attribution de la note.

1.4.2.3. *Les notes globales*

Pour une pratique donnée, chaque composant du projet visé par celle-ci obtient une note individuelle située dans l'intervalle $[0; 3]$. Ces notes sont ensuite agrégées ensemble pour obtenir la note globale de la pratique (c'est à dire pour le projet) : la note obtenue au niveau projet pour une pratique donnée.

Les fonctions utilisées pour calculer cette note globale permettent de faire ressortir les mauvaises notes pour mettre en avant les points faibles. Pour cela, un système de pondération a été introduit et permet de définir un seuil de tolérance aux mauvaises notes individuelles. Ce seuil est fixé en fonction des exigences des entreprises et permet de renforcer les pratiques importantes et critiques : les pondérations utilisées pour ces dernières sont plus sévères que pour les autres, faisant baisser la note globale plus rapidement :

– un poids fort est appliqué quand la tolérance aux erreurs est extrêmement faible. Dans ce cas, il suffit de peu de mauvaises notes individuelles pour obtenir une mauvaise note globale – comprise dans l'intervalle $[0; 1]$;

– un poids moyen est utilisé quand la tolérance aux erreurs se situe dans la norme. La note globale chute dans l'intervalle $[0; 1]$ uniquement s'il existe un nombre moyen de mauvaises notes individuelles ;

– un poids faible est appliqué en cas de tolérance très grande aux erreurs. La note globale ne chute alors que s'il existe un grand nombre de notes individuelles mauvaises.

Le calcul de la note globale s'effectue en plusieurs étapes comme l'illustre la figure 1.4. Les trois notes individuelles de départ ont pour valeur $0,5$, $1,5$, et 3 :

– dans un premier temps une fonction de pondération est appliquée à chaque note individuelle :
$g(IM) = \lambda^{-IM}$
où IM – *Individual Mark* – correspond à la note individuelle d'un composant pour une pratique donnée et λ correspond à une constante qui définit le poids appliqué –

lourd, moyen ou léger. λ est plus élevé pour un poids fort tandis qu'il diminue pour un poids faible. Dans la figure 1.4, les notes pondérées sont représentées sur l'axe des y avec un poids moyen appliqué soit $\lambda = 9$. Cette transformation transpose également les notes individuelles dans un nouvel espace où les notes les plus basses ont plus de poids que les plus élevées ;

– dans un second temps, la moyenne de ces notes pondérées à l'étape précédente est calculée (*weighted average* sur l'axe des y). Le point *mark average* représente la moyenne simple des notes initiales ;

– enfin, sur cette moyenne est appliquée la fonction opposée :
$g^{-1}(Wavg(IMs)) = -log_{\lambda}(Wavg(IMs))$
qui permet de retransposer la note obtenue dans l'espace de notation initial de $[0; 3]$: *weighted mark* sur l'axe des x.

Si nous regroupons toutes ces étapes sous la forme d'une unique fonction de calcul nous obtenons :

$$mark = -log_{\lambda}\left(\frac{\sum_{1}^{n} \lambda^{-IM_n}}{n}\right)$$

où λ varie selon le poids appliqué et IM_n correspond à la note individuelle pour le composant n.

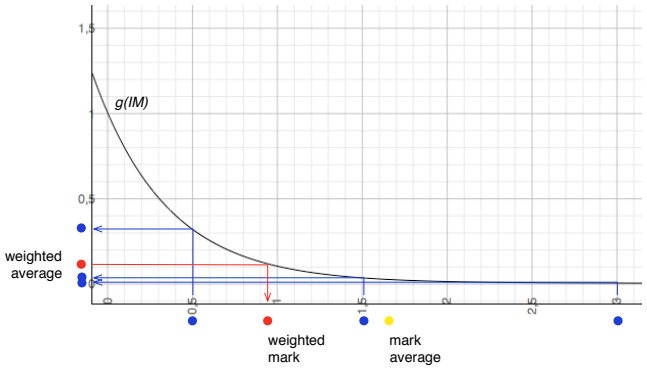

Figure 1.4 – Principe de pondération : les notes individuelles sont diminuées lorsqu'elles sont transposées dans l'espace pondéré.

1.4.3. *L'exemple d'une pratique*

Prenons la pratique taux de commentaires. Elle a pour propriétés :

– définition : qualifie le taux de commentaires des lignes de code d'une méthode. Cette pratique ne prend pas en compte les javadoc par exemple. Elle vérifie l'assertion

suivante : plus une méthode est complexe et plus elle doit être commentée. La complexité d'une méthode est calculée en fonction du nombre de lignes de code mais également de sa complexité cyclomatique. La fonction de calcul des notes individuelles a été définie avec des seuils qui tiennent compte de l'encapsulation – accesseurs et mutateurs ne sont pas pris en compte dans le calcul – et qui visent à obtenir des taux de commentaires allant de 10 % à 30 % en fonction de la complexité de la méthode ;

– composant : méthodes ;

– métriques : V(g) pour complexité cyclomatique, NCLOC pour nombre de lignes de commentaires et SLOC pour nombre de lignes de code source ;

– note individuelle (IM) : si $v(G) >= 5$ ou $sloc > 30$:
alors :
$IM = (ncloc) * 9/(ncloc + sloc)/(1 - 10^{(-v(G)/15)})$

– poids : faible.

Le tableau 1.6 donne les valeurs seuils de cette pratique, à partir desquelles a été déterminée la formule de calcul des notes individuelles.

Note de la pratique	**Métrique v(G)**	**Taux de commentaires**
< 0,5	>= 5	1 %
< 1	>= 5	5 %
>= 1 and <= 2	<= 15	10 %
< 1	> 15	
3	<= 14	30 %
2,8 or 2,9	<= 26	
2,7	>= 27	
3	–	50 %

Tableau 1.6 – Mesures/notes de référence servant de base à la formule déterminant la note de la pratique taux de commentaires

Pour un projet donné, chaque méthode du projet obtient ainsi une note individuelle dans l'intervalle $[0; 3]$ (sauf pour les méthodes d'encapsulation qui ne sont pas prises en compte). L'ensemble de ces notes est ensuite agrégé selon la formule définie précédemment pour obtenir une note globale de la pratique taux de commentaires pour l'ensemble du projet.

Le tableau 1.7 donne un exemple de notes attribuées à dix méthodes en fonction des résultats des métriques SLOC, v(G), NCLOC. La note individuelle varie en fonction du nombre de lignes de commentaires par rapport au nombre de lignes de code mais également en fonction de la complexité de la méthode. Les méthodes G et H, bien qu'ayant les mêmes nombres de lignes de commentaires et de code n'obtiennent pas la

même note : la méthode H ayant une complexité moindre, elle obtient de fait une note individuelle plus élevée, par exemple. Ceci est dû aux formules de calcul des notes individuelles dans le modèle Squale qui tiennent comptent de plusieurs paramètres.

Le tableau 1.7 donne également les résultats d'agrégation des notes individuelles selon les principes expliqués précédemment. La note globale varie en fonction du poids attribué : plus le poids est fort et plus l'exigence est élevée, ce qui se traduit par une note globale qui diminue fortement en fonction du poids. La moyenne simple quant à elle ne traduit pas le fait que certaines méthodes sont en dessous des exigences requises (les méthodes A et E par exemple) et donne une résultat globalement satisfaisant.

Méthode	A	B	C	D	E	F	G	H	I	J
Métrique v(G)	6	7	3	11	6	8	5	3	11	8
Métrique NCLOC	3	10	10	35	3	30	20	20	10	30
Métrique SLOC	33	39	40	50	50	50	150	150	50	113
Note individuelle	1,246	2,789	3	3	0,846	3	1,976	2,869	1,840	2, 670

Note globale avec un poids faible	1,969
Note globale avec un poids moyen	1,657
Note globale avec un poids fort	1,442
Moyenne simple	2,324

Tableau 1.7 – Série 1 de notes pour la pratique taux de commentaires

Méthode	A	B	C	D	E	F	G	H	I	J
Métrique v(G)	6	7	3	11	6	8	5	3	11	8
Métrique NCLOC	3	10	10	35	**10**	30	20	20	10	30
Métrique SLOC	33	39	40	50	50	50	150	150	50	113
Note individuelle	1,246	2,789	3	3	**2.492**	3	1,976	2,869	1,840	2, 670

Note globale avec un poids faible	2,277
Note globale avec un poids moyen	2,050
Note globale avec un poids fort	1,856
Moyenne simple	2,488

Tableau 1.8 – Série 2 de notes pour la pratique taux de commentaires

Le tableau 1.8 reprend les mêmes valeurs que le tableau précédent à une valeur près : le nombre de lignes de commentaires de la méthode E. La note individuelle de cette méthode s'en trouve alors fortement augmentée : elle passe d'une note insuffisante à une note acceptable selon le barème d'interprétation des notes du modèle

Squale (voir section 1.4.1). Les notes globales s'interprétant de la même façon, la figure 1.5 permet de constater que dans la série 1 toutes les notes globales sont inacceptables alors que pour la série 2, les notes globales de poids faible et moyen sont devenues acceptables. L'utilisation de poids renforce l'écart entre les résultats des deux séries. Les poids servent donc à ajuster le niveau d'exigence des entreprises pour chaque pratique.

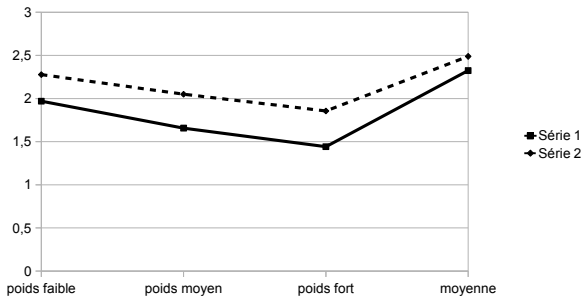

Figure 1.5 – Les notes globales obtenues pour les deux séries des tableaux 1.7 et 1.8

1.5. Remédiation et plans d'action

Appliquer des modèles de qualité à des logiciels ne constitue qu'une première étape. En effet, les entreprises qui mesurent la qualité de leur logiciel ont majoritairement pour but soit d'augmenter celle-ci, soit d'en éviter la détérioration au fil du temps. La constitution d'un plan de remédiation constitue donc l'étape suivante dans le modèle Squale : construire un outil d'aide à la décision à partir d'une vue qualitative.

En effet, les modifications, que ce soient des corrections de bogues ou des améliorations de fonctionnalités peuvent entraîner une détérioration de la qualité. De même, ces opérations de maintenance peuvent être l'occasion d'augmenter la qualité du logiciel. Pour ce faire, les chefs de projet vont devoir prendre des décisions et orienter les développements judicieusement. Le plan de remédiation a pour but d'aider à cette prise de décision. Il ne s'agit pas de se substituer aux décisions humaines mais d'aider dans les choix à prendre.

Identifier les modifications et calculer leur coût par composant reste un défi. Plusieurs travaux ont déjà tenté de couvrir l'un des deux aspects de cette problématique. Nous citerons les plus connus et les plus intéressants :

– les modèles d'estimation des coûts comme *Cocomo* (bien qu'ils s'attachent uniquement à l'estimation de coûts et non pas à l'identification des modifications). De plus, ces modèles ne prennent pas en compte les éventuelles conséquences des modifications d'un composant [KEM 87] ;

– les modèles de refactorisation et les actions de maintenance [BUC 05, MEN 04] qui s'attachent à déterminer les modifications à apporter à un logiciel mais qui ne prennent pas en compte les coûts de maintenance comme critère de décision ;

– les modèles de prédiction des efforts de maintenance [BEN 99, J.E 97, MUN 98] ;

– les modèles d'impact de changements [BOH 96].

Les plans de remédiation présentent de multiples caractéristiques qui régissent leur portée et leur application au sein d'une entreprise : il faut savoir prendre en compte à la fois le code impacté mais également les équipes qui devront travailler dessus ou encore prendre en compte l'opportunité de certaines modifications. La démarche visant à augmenter la qualité n'échappe pas à ces principes et doit toujours les prendre en compte pour les utiliser à bon escient. Il est important de connaître les caractéristiques des tâches abordées par le plan, son aspect organisationnel par rapport aux équipes de développement et au cycle de développement du projet par exemple.

1.5.1. *Tâches abordées par Squale*

Le but du plan de remédiation du modèle Squale est de définir une stratégie globale d'augmentation de la qualité à partir de calculs de coûts unitaires effectués sur les composants du projet. L'effort de remédiation unitaire est défini comme étant la quantité de travail à fournir pour obtenir une meilleure qualité de l'entité analysée.

Le socle du plan de remédiation de Squale est constitué par l'idée que l'augmentation de la qualité doit être incrémentale. Le plan se base autant sur les composants défectueux que sur les violations des pratiques du modèle. Le but du plan est de fournir un guide pour améliorer la qualité au plus faible coût possible.

Les tâches abordées par le plan de remédiation de Squale sont caractérisées comme suit :

– la transgression : une instance qui n'a pas atteint l'objectif visé par la pratique c'est à dire que ce soit un composant ou une règle de transgression ;

– la tâche de remédiation : la description des actions à mener pour résoudre la transgression d'une pratique.

– le coût de remédiation : le coût est caractérisé par le travail à accomplir pour appliquer une tâche de remédiation.

1.5.1.1. *La transgression : de l'utilisation des pratiques*

Le plan de remédiation de Squale utilise les caractéristiques du modèle de qualité sur lequel il s'appuie. Il repose donc essentiellement sur les pratiques du modèle et sur leur notes – individuelles ou globales.

Une transgression est donc définie comme étant un objectif non atteint défini par une pratique donnée. Ainsi, une transgression se détermine en fonction d'une pratique et des composants ciblés par cette pratique.

Par exemple, si on prend la pratique nombre de méthodes, celle-ci définit un nombre de méthodes par classe à respecter. Une classe qui transgresse cette pratique possède un nombre de méthodes inadéquat et donc une mauvaise note individuelle pour cette pratique. Une transgression est donc directement liée aux notes individuelles des pratiques.

Dans le cas des pratiques de convention de programmation, celles-ci étant définies pour le projet en entier, une transgression correspond alors à une règle non respectée pour l'ensemble du projet.

1.5.1.2. *La tâche de remédiation : la portée d'une action*

Une tâche de remédiation est une unité conceptuelle décrivant les actions à mener pour résoudre une transgression. Dans le contexte du modèle Squale, deux questions guident l'évaluation des efforts de remédiation :

1) Dans quel contexte et caractéristiques du système l'effort de remédiation se situe-t-il ?

2) Quel est le but souhaité pour cette action de remédiation ?

Le contexte évalue le composant visé par la pratique aussi bien que les composants reliés qui seront sans doute également impactés par cette action. Il se compose de deux paramètres qui définissent le contexte : la complexité c'est à dire est-il facile de modifier le composant selon l'action voulue ; l'impact c'est à dire combien de composants vont être touchés par cette action.

Le but représente le niveau de qualité recherché. Dans le contexte du modèle Squale, la qualité est notée dans l'intervalle [0 ; 3] pour permettre une interprétation précise du niveau de qualité. Il est donc naturel d'utiliser ces notes pour définir les objectifs. Le but se base donc sur les énoncés suivants :

– une note au dessus de 2 est satisfaisante,

– une note comprise entre 1 et 2 devrait être améliorer si possible,

– une note inférieure à 1 doit être une priorité pour le modèle de remédiation.

Au final, la remédiation est un compromis entre les buts à atteindre (en termes de notes) et le contexte qui permet de déterminer les ressources à mettre en oeuvre pour fournir l'effort souhaité.

Nous identifions trois principes fondamentaux pour évaluer le travail nécessaire à une tâche de remédiation :

– la portée de la pratique et de la remédiation : évalue la cible de la remédiation au sein du projet. Le tableau 1.9 décrit les différentes portées de la remédiation en fonction des pratiques visées.

– le degré d'automatisation du travail de remédiation : Certaines tâches de remédiation peuvent être automatisées en effectuant quelques corrections préalables. C'est le cas par exemple des pratiques visant la conformité avec certains standards de programmation. Dans ces cas, il est plus judicieux de cibler les transgressions plutôt que les composants. En effet, la transgression ne sera corrigée qu'une fois tous les composants mis en conformité et homogènes.

– l'expertise nécessaire à la rémédiation : Tandis que certaines règles décrites par les pratiques sont faciles à suivre, d'autre nécessite un niveau d'expertise plus élevé. Par exemple, les tâches liées à l'architecture et au design, ou à la sécurité ont besoin d'un expert dans leur domaine respectif.

A cela s'ajoute quatre caractéristiques supplémentaires qui dépendent du contexte (du type de projet et des exigences et standards appliqués par l'entreprise) :

– les propriétés et l'organisation du code : un plan de remédiation nécessite de prendre en compte l'organisation du code aussi bien que ses propriétés. Certains composants peuvent se situer en dehors du champ d'action du plan. Il est parfois indispensable de bien connaitre les composants et leur historique de développement pour déterminer si certaines transgressions sont nécessaires et doivent être conservées.

– la criticité d'une pratique : toutes les pratiques ne sont pas de même importance tout au long du cycle de vie d'une application. Par exemple, les pratiques de test deviennent primordiales lorsque l'application est en phase de tests. De plus, en fonction de certaines contraintes, un chef de projet peut décider que certaines pratiques deviennent prioritaires par rapport à d'autres, ce qui entraîne alors une modification des priorités dans le plan de remédiation.

– la criticité du composant : une connaissance en interne du projet peut conduire à décider de rendre prioritaires certains composants lorsqu'ils sont les éléments centraux correspondant au stade courant de développement du projet. D'un autre côté, certains autres composants peuvent être déclarés hors du champ d'action s'ils fonctionnent malgré leur défauts et qu'il n'est pas prévu de les modifier.

– les caractéristiques du composant : selon la pratique, les caractéristiques tels que la taille, la complexité, le couplage peuvent avoir un impact important sur les corrections à effectuer. Certaines pratiques prennent explicitement ces corrélations en

Pratique	Portée	Portée de la remédiation
Audit	Projet	Selection des composants prioritaires identifiés lors de l'audit
Metrique Modèle Test	Composant	le composant et ses dépendances (package, classe, méthode...)
Conventions	Règles transgression	Composant responsable de la transgression

Tableau 1.9 – Portée des pratique et de la remédiation

considération : par exemple, le fait qu'il faille fournir plus de tests pour les méthodes plus complexes.

1.5.1.3. *Coût de remédiation : de la mesure de l'effort*

1.5.1.3.1. Mesure de l'effort

Le plan de remédiation repose sur les transgressions des pratiques du modèle de qualité de Squale et définit les tâches de remédiation comme les actions à mener pour ramener la note d'une pratique au niveau souhaité. La formule qui permet de calculer l'effort de remédiation doit donc prendre en compte les propriétés des pratiques. Les pratiques reposant sur des mesures de code, il est donc indispensable de prendre en compte ces mesures pour le calcul de l'effort de remédiation. Ceci permet de relier directement les efforts de remédiation aux notes de la qualité : propriété importante qui permet de mesurer le niveau de travail à fournir pour parvenir au but souhaité.

Prenons la pratique taux de commentaires. Les transgressions pour cette pratique correspondent aux méthodes qui ne sont pas assez commentées. Calculer l'effort de remédiation pour ces méthodes consiste alors à calculer le coût que représente l'ajout de commentaires dans le code. La métrique NCLOC est donc directement reliée à la mesure de l'effort de remédiation.

Cependant, les formules des pratiques ne sont pas nécessairement écrites pour évaluer la complexité de la situation mais pour détecter des problèmes. Prenons par exemple la pratique nombre de méthodes dont la note est calculée à partir des métriques NOM et $v(G)$. Si la formule de calcul de la pratique est effectivement judicieuse pour détecter les classes qui transgressent cette pratique, en revanche, les métriques utilisées ne peuvent à elles seules évaluer l'effort de remédiation. En effet, la remédiation consiste à diviser la classe de manière à ce que les méthodes soient distribuées entre deux classes plus petites. Cependant, évaluer la difficulté de diviser une classe en deux se fait plus facilement grâce à des métriques de cohésion (plus la

méthode est cohésive et plus il est difficile de la séparer en deux) qu'avec les métriques NOM ou $v(G)$.

De plus, lors de la mesure de l'effort, d'autres paramètres peuvent également influencer cette mesure. Prenons à nouveau l'exemple de la pratique nombre de méthodes. Diviser une classe en deux est souvent liée à des décisions de conception : séparer une classe oblige à penser au nombre de nouvelles classes à créer et à leur interaction par exemple.

Ainsi, la mesure de l'effort de remédiation dépend de la transgression et de l'impact de la remédiation. A chaque pratique correspond une formule d'estimation de coût.

1.5.1.3.2. Coût de remédiation

Le coût est caractérisé par le travail à accomplir pour appliquer une tâche de remédiation. Le coût de remédiation (noté W) pour une pratique et pour un composant donné est basé sur la formule empirique des pratiques : l'objectif est de donner une évaluation du coût, basé sur la complexité du cas et l'impact les modifications. Ceci implique une estimation du nombre de corrections nécessaires et du poids des actions de refactorisation.

Quatre types de paramètres sont utilisés pour calculer ce coût :

– complexité : évaluation des caractéristiques du composant visé qui doit être soumis à la remédiation.

– impact : le nombre de composants touchés par la remédiation (qui doivent également être modifiés pour parvenir au but)

– but à atteindre : la note attendue après la remédiation.

– l'unité du coût de remédiation (URC) : l'unité de l'effort qui dépend des changements à appliquer.

Notons que ces quatre paramètres ne sont pas nécessairement indépendants les uns des autres. Par exemple, évaluer la complexité du cas dépend du but à atteindre. Plus la note attendue est élevée, plus le cas est complexe à résoudre.

De plus, les pratiques identifient souvent des situations complexes pour lesquelles il n'existe pas un seul mais de multiples problèmes. Pour cette raison, fixer un coût de remédiation devient alors beaucoup plus difficile. Il faut alors parvenir à déterminer le niveau d'expertise requis pour effectuer ces corrections. C'est pourquoi les coûts exprimés ne peuvent être que des estimations qui doivent être affinées par des experts en fonction de la situation.

1.5.1.3.3. Coût unitaire d'une remédiation

Quand on modifie une partie du code source, plusieurs coûts sont à prendre en compte. De plus, on ne peut évaluer ce coût en dehors du contexte et des nécessités de l'application. Par exemple, il faut prendre en compte :

– la nécessité de maintenir une compatibilité "arrière" ;

– le nombre de tests qu'il faudra à nouveau effectuer ;

– la phase du projet. Plus les corrections à faire sont effectuées tôt au cours des phases de développement et plus elles sont faciles à mettre en oeuvre.

1.5.2. *Aspect organisationnel*

Chaque entreprise a sa propre politique d'application de la qualité et utilise la remédiation à différents niveaux, depuis le technicien de base jusqu'au chef de projet. Un modèle de qualité et de remédiation doit donc s'adresser à tous et prendre en compte toutes les équipes du projet. Il doit pouvoir doit être adaptable et répartir les tâches recensées entre les différentes équipes :

– le développeur peut orienter son travail quotidien sans ressentir comme un fardeau les processus liées à la qualité.

– le chef de projet peut utiliser cet outil pour évaluer la situation et orienter les efforts à fournir. Il peut s'appuyer sur le plan de remédiation pour augmenter la qualité globale du projet.

– l'équipe qui s'occupe de la qualité doit pouvoir adapter le modèle en fonction des besoins et des standards de l'entreprise.

L'aspect organisationnel du plan de remédiation détermine également l'ordre dans lequel les tâches à effectuer seront présentées, depuis les tâches prioritaires jusqu'aux tâches les moins importantes.

1.5.2.1. *Choisir sa stratégie*

Le modèle actuellement utilisé par Squale vise à réduire les risques en ciblant les mauvaises notes des pratiques et en priorisant les remédiations qui vont augmenter ces notes au moindre coût. Le modèle est donc orienté pratique en donnant la priorité aux pratiques par rapport aux composants du projet.

1.5.2.1.1. Stratégies de base

Indépendamment des critères de haut niveau comme le coût, la rentabilité, ou la criticité, il existe deux stratégies de base pour augmenter la qualité d'une pratique :

– d'améliorer légèrement les marques de nombreux composants ;

– améliorer les notes des pires éléments.

Les poids appliqués aux notes individuelles des pratiques (voir Section 1.4.2.3) déterminent la stratégie qui devra être adoptée : utiliser des poids forts obligent à s'intéresser aux plus mauvais composants et donc à améliorer les pires éléments, tandis que l'utilisation des poids faibles entraine la stratégie inverse. La stratégie dépend également de la complexité et de l'automatisation de la tâche de remédiation, comme cela est décrit dans la section 1.5.1.2. les corrections visant les pratiques de convention de nommage peuvent se faire par l'intermédiaire d'un script qui automatisera les corrections et qui s'effectuera pour l'ensemble du projet tandis que les corrections nécessitant une plus grande expertise sont d'avantage susceptibles d'être abordés en se concentrant uniquement sur quelques composants défectueux.

1.5.2.1.2. Objectifs

A partir de ces constatations, les chef de projet vont alors avoir le choix entre deux stratégies pour remédier aux problèmes liés à la qualité :

– parvenir à une réduction importante des risques en ciblant les éléments et les pratiques critiques. Nous appelons ce choix le modèle de risque.

– obtenir la meilleure rentabilité dans les processus de qualité c'est à dire le meilleur compromis entre l'augmentation de la qualité et les coûts de remédiation liés. Nous appelons ce choix le modèle de rentabilité.

1.5.2.2. *Stratégie utilisée par Squale : Mixed priority-cost strategy*

La stratégie mise en place dans le modèle actuel de Squale suit quatre étapes :

1) les pratiques avec les plus basses notes sont prioritaires ;

2) les composants qui échouent pour une pratique doivent être corrigés avant les autres (on améliore d'abord les plus mauvais composants) ;

3) certaines pratiques sont plus faciles à corriger que d'autres ;

4) la charge de travail nécessaire pour corriger une pratique est fonction de la pratique elle-même et du nombre de transgressions de la pratique.

1.5.2.2.1. Priorisation des pratiques en fonction de leur note

Le modèle Squale classe ses pratiques en trois catégories – "refusée", "acceptée avec des reserves" et "accepté" – en fonction des notes attribuées. Le plan de remédiation utilise ces notes pour déterminer l'ordre de priorité des pratiques : les pratiques "refusées" obtiennent la plus grande priorité, suivies des pratiques "acceptées avec réserves". Les pratiques "acceptées" sont en dehors du cadre de la remédiation mais peuvent toutefois faire l'objet d'amélioration futures.

1.5.2.2.2. Sélection des composants en fonction des pratiques

Une fois les pratiques sélectionnées en fonction de leur note, seuls les composants qui ont obtenu une mauvaise note individuelle – une note inférieure ou égale à la note

globale – apparaissent dans le plan de remédiation. Les composants ayant obtenu une note correcte peuvent éventuellement faire l'objet d'amélioration mais pas de correction.

1.5.2.2.3. Coefficient de correction par pratique

Un coefficient de correction est déterminé pour chaque pratique, en collaboration avec l'entreprise. Celui-ci évalue les efforts relatifs nécessaires pour corriger une transgression pour une pratique donnée, par rapport aux autres pratiques. Il sert à traduire des faits tels que, par exemple, la transgression de normes de formatage du code est plus facile à corriger qu'un problème de conception dû au couplage. Ce coefficient agit un peu comme un poids qui traduit la quantité d'effort en fonction de la pratique. Ce coefficient est déterminé de manière empirique en fonction des spécificités de l'entreprise.

1.5.2.2.4. Charge de travail par pratique

étant donné que le coefficient de correction pour une pratique est une constante, la charge de travail pour corriger toutes les transgressions de la pratique est définie comme étant le produit du coefficient par le nombre de transgressions. Pour les pratiques basées sur des métriques, le nombre de transgressions est le nombre de composants sélectionnés. Pour les pratiques basées sur du rule cheking (convention de nommage), il s'agit directement du nombre de transgressions détecté par la règle. Cependant, ce calcul implique que le coefficient de correction ait été déterminé avec le plus grand soin. C'est pour cette raison que ce dernier nécessite des ajustements dans les premières phases d'utilisation du plan de remédiation. En effet, le coefficient possède une marge d'erreur qui multipliée aux composants traités entraîne une augmentation significative de la marge d'erreur globale.

1.5.2.2.5. Stratégie de remédiation

L'idée est de parcourir chaque niveau défini par Squale (depuis "refusé" à "accepté") pour pour chacun de sélectionner les pratiques en commençant par celles qui ont la plus petite charge de travail.

1.5.2.2.6. Limitations

Il y a plusieurs limites à cette stratégie :

1) Le coefficient est une constante indépendante du composant qui transgresse la pratique, ce qui signifie que le coût de remédiation ne prend en compte les caractéristiques du composant visé. Pour la pratique de couverture de tests, l'effort pour écrire des tests pour des méthodes complexes (ce qui est requis par la pratique) est lié łeur complexié et croit avec celle-ci ce qui n'est pas pris en compte par une simple constante.

2) Certaines pratiques sont contradictoires, corriger une pratique peut dégrader d'autres pratiques.

3) Bien que cette stratégie trie les pratiques par notes et effort, elle ne peut déterminer de manière optimum ou et quand les notes doivent-être améliorées. Pour cette raison, la charge de travail peut ne pas être optimale.

4) Bien que la tâche de remédiation soit fixée en fonction d'un but à atteindre, les tâches proposées ne garantissent pas d'atteindre ce but. Les corrections proposées amélioreront la note de manière incontestable mais ne parviendront pas forcément à obtenir la note maximale.

5) Les composants impliqués dans plusieurs transgressions ne sont pas détectés. Il est donc possible qu'ils soient une cible de travail répétée.

1.6. Conclusion

Mesurer la qualité d'un logiciel est plus complexe qu'il n'y parait. Il faut tout d'abord définir clairement ce que sous-entend le mot qualité pour chaque entreprise, chaque équipe et chaque logiciel mesuré. Une fois les objectifs qualité clairement établis, encore faut-il savoir comment les mesurer. Il ne suffit pas simplement de collecter des résultats de métriques mais surtout de donner du sens à ces différentes mesures.

Les modèles de type ISO 9126 servent de cadre en établissant différentes catégories, en classant différents concepts, mais restent toutefois trop imprécis ou trop difficiles à mettre concrètement en oeuvre pour donner une totale satisfaction.

Un autre défi à relever consiste à déterminer une note globale pour une caractéristique donnée à partir de métriques brutes. Il faut pouvoir combiner ces métriques de manière à traduire les informations qu'elles délivrent pour fournir une lecture aisée de la qualité. De simples transpositions ou de simples moyennes ne permettent pas d'y parvenir de manière totalement satisfaisante.

Pour ces raisons, le modèle de qualité Squale a introduit un nouveau concept : les pratiques. Celles-ci définissent des normes de qualité et des méthodes de programmation. Chaque pratique qualifie une technique de programmation et permet également la combinaison des métriques brutes en notes de plus haut niveau facilement compréhensibles.

En plus de fournir une vue de la qualité à un moment donné, le modèle Squale propose également des pistes de remédiation de la qualité en utilisant les pratiques comme point de référence. Bien que nécessitant encore des améliorations notamment en ce qui concerne ses formules d'agrégation de coût de l'effort, le plan de remédiation permet de dégager des pistes d'amélioration de la qualité en mettant en avant les plus mauvais composants qui composent les plus mauvaises pratiques. Il s'agit de fournir une aide à la décision quant aux choix à effectuer pour combiner maintenance applicative et augmentation de la qualité : mettre en corrélation les composants devraient

être améliorés d'un point de vue qualitatif avec ceux qui doivent être modifiés dans le cadre d'une opération de maintenance.

Le modèle Squale est donc un modèle complet qui fournit une vue détaillée mais également une vue d'ensemble de la qualité. Il donne également des pistes pour augmenter le niveau de la qualité grâce à son plan de remédiation. Cependant, ce plan reste une indication théorique et demeure encore trop imprécis pour se substituer totalement à l'avis d'un expert. De plus, le plan doit également être amélioré pour définir comment fournir une estimation beaucoup plus précise de la charge de travail. De même, les conditions et le contexte de travail au sein d'une entreprise sont pris en compte dans une certaine mesure, lors de la mise en place des pratiques. Il serait pourtant judicieux de les prendre en compte plus finement lors du paramétrage du plan de remédiation. Il faudrait par exemple inclure dans les calculs à quel moment on intervient dans le cycle de vie du logiciel, ou encore prendre en compte le fait que les équipes de développement travaillent par exemple à la correction d'un bogue sur une partie du logiciel, ou encore à l'implémentation d'une nouvelle propriété. Il s'agit de mutualiser les efforts : inclure la remédiation dans les développements en cours pour diminuer les coûts.

1.7. Bibliographie

[ABR 04] ABRAN A., BOURQUE P., DUPUIS R., TRIPP L., Guide to the software engineering body of knowledge (ironman version), Rapport, IEEE Computer Society, 2004.

[BAK 08] BAKOTA T., BESZÉDES A., FERENĆ R., GYIMÓTHY T., « Continuous software quality supervision using SourceInventory and Columbus », *Companion of the 30th international conference on Software engineering*, ICSE Companion '08, New York, NY, Etats-Unis, ACM, p. 931–932, 2008.

[BAL 09] BALMAS F., BERGEL A., DENIER S., DUCASSE S., LAVAL J., MORDAL-MANET K., ABDEEN H., BELLINGARD F., Software metric for Java and C++ practices (Squale Deliverable 1.), Rapport, INRIA Lille Nord Europe, 2009.

[BAN 02] BANSIYA J., DAVIS C., « A Hierarchical Model for Object-Oriented Design Quality Assessment », *IEEE Transactions on Software Engineering*, vol. 28, n°1, p. 4–17, janvier 2002.

[BAR 09] BARKMANN H., LINCKE R., LÖWE W., « Quantitative evaluation of software quality metrics in open-source projects », *Advanced Information Networking and Applications (WAINA'09). International Conference on*, IEEE, p. 1067–1072, 2009.

[BAS 94] BASILI V. R., CALDIERA G., ROMBACH H. D., « The Goal Question Metric Approach », *Encyclopedia of Software Engineering*, Wiley, 1994.

[BEN 99] BENGTSSON P., BOSCH J., « Architecture Level Prediction of Software Maintenance », *European Conference on Software Maintenance and Reengineering (CSMR'99)*, p. 139–147, 1999.

[BIE 96] BIEMAN J. M., « Metric Development for Object-Oriented Software », 1996.

[BOH 96] BOHNER S. A., ARNOLD R. S., *Software Change Impact Analysis*, IEEE Computer Society Press, 1996.

[BRI 98] BRIAND L. C., DALY J. W., WÜST J. K., « A Unified Framework for Cohesion Measurement in Object-Oriented Systems », *Empirical Software Engineering : An International Journal*, vol. 3, n°1, p. 65–117, Kluwer Academic Publishers, 1998.

[BUC 05] BUCKLEY J., MENS T., ZENGER M., RASHID A., KNIESEL G., « Towards a Taxonomy of Software Change », *Journal on Software Maintenance and Evolution : Research and Practice*, p. 309–332, 2005.

[CHI 94] CHIDAMBER S. R., KEMERER C. F., « A Metrics Suite for Object Oriented Design », *IEEE Transactions on Software Engineering*, vol. 20, n°6, p. 476–493, juin 1994.

[COW 95] COWELL F. A., JENKINS S. P., « How Much Inequality Can We Explain ? A Methodology and an Application to the United States », *Economic Journal*, vol. 105, n°429, p. 421-30, March 1995.

[COW 00] COWELL F. A., « Measurement of inequality », ATKINSON A. B., BOURGUIGNON F., Eds., *Handbook of Income Distribution*, vol. 1, p. 87–166, Elsevier, 2000.

[ESM 06] ESMOND B. MARVRAY NASA GODDARD SPACE FLIGHT CENTER S. Q., « Procedure for Developing and Implementing Software Quality Programs », last update 2006.

[FEN 96] FENTON N., PFLEEGER S. L., *Software Metrics : A Rigorous and Practical Approach*, International Thomson Computer Press, London, UK, second édition, 1996, 06-8147-I*, envoye a l'inria lille le 19 aout.

[ISO 01] ISO/IEC, « ISO/IEC 9126-1 Software engineering -Product quality- part 1 : Quality model », 2001.

[ISO 03] ISO/IEC, « ISO/IEC 9126-3 Software engineering -Product quality- part 3 : Internal metrics », 2003.

[ISO 05] ISO/IEC, « ISO/IEC 25000 :2005 Software engineering -Software product QUAlity Requirement and Evaluation », 2005.

[J.E 97] J.E. H., J.P. C., « A quantitative comparison of perfective and corrective software maintenance », *Journal of Software Maintenance : Research and Practice*, p. 281-297, 1997.

[JON 08] JONES C., *Applied Software Measurement : Global Analysis of Productivity and Quality*, McGraw-Hill, Inc., Hightstown, NJ, Etats-Unis, 2008.

[KEM 87] KEMERER C. F., « An Empirical Validation of Software Cost Estimation Models », *Communications of the ACM*, 1987.

[LAN 06a] LANZA M., MARINESCU R., *Object-Oriented Metrics in Practice*, Springer-Verlag, 2006.

[LAN 06b] LANZA M., MARINESCU R., *Object-oriented metrics in practice : using software metrics to characterize, evaluate, and improve the design of object-oriented systems*, Springer, 2006.

[LOR 94] LORENZ M., KIDD J., *Object-Oriented Software Metrics : A Practical Guide*, Prentice-Hall, 1994.

[MAR 97] MARTIN R. C., « Stability », 1997, www.objectmentor.com.

[MAR 04a] MARINESCU R., « Detection Strategies : Metrics-Based Rules for Detecting Design Flaws », *20th IEEE International Conference on Software Maintenance (ICSM'04)*, Los Alamitos CA, IEEE Computer Society Press, p. 350–359, 2004.

[MAR 04b] MARINESCU R., RAŢIU D., « Quantifying the Quality of Object-Oriented Design : the Factor-Strategy Model », *Proceedings 11th Working Conference on Reverse Engineering (WCRE'04)*, Los Alamitos CA, IEEE Computer Society Press, p. 192–201, 2004.

[MCC 76a] MCCABE T., « A Measure of Complexity », *IEEE Transactions on Software Engineering*, vol. 2, n°4, p. 308–320, décembre 1976.

[MCC 76b] MCCALL J., RICHARDS P., WALTERS G., *Factors in Software Quality*, NTIS Springfield, 1976.

[MEN 04] MENS T., TOURWÉ T., « A Survey of Software Refactoring », *IEEE Transaction on Software Engineering*, vol. 30, n°2, p. 126–139, IEEE Press, 2004.

[MUN 98] MUNSON J., ELBAUM S., « Code Churn : A Measure for Estimating the Impact of Code Change », *ICSM'98*, p. 24-34, 1998.

[PER 07] PEREPLETCHIKOV M., RYAN C., FRAMPTON K., TARI Z., « Coupling metrics for predicting maintainability in service-oriented designs », *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, IEEE, p. 329–340, 2007.

[SER 10] SEREBRENIK A., VAN DEN BRAND M., « Theil index for aggregation of software metrics values », *Proceedings of ICSM 2010*, 2010.

[TUR 11] TURNU I., CONCAS G., MARCHESI M., PINNA S., TONELLI R., « A modified Yule process to model the evolution of some object-oriented system properties », *Inf. Sci.*, vol. 181, p. 883–902, Elsevier Science Inc., février 2011.

[VAS 09] VASA R., LUMPE M., BRANCH P., NIERSTRASZ O., « Comparative Analysis of Evolving Software Systems Using the Gini Coefficient », *Proceedings of the 25th International Conference on Software Maintenance (ICSM 2009)*, Los Alamitos, CA, Etats-Unis, IEEE Computer Society, p. 179–188, 2009.

[VAS 10] VASILESCU B., SEREBRENIK A., VAN DEN BRAND M., « Comparative Study of Software Metrics' Aggregation Techniques », *Proceedings of the International Worskhop Benevol 2010*, 2010.