ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COUR MGL804
DE LA MAITRISE EN GÉNIE LOGICIEL

ANALYSE DE LA MAINTENABILITÉ D'UN PROJET

PAR
Olivier, MIRANDETTE

MONTRÉAL, 27-04-2014

REMERCIEMENTS

Mon employeur Guavus pour me permettre d'utiliser deux de nos logiciels pour l'analyse.

SELECTION ET APPLICATION D'UN MODÈLE DE MESURE DE LA MAINTENABILITÉ

Olivier, MIRANDETTE

RÉSUMÉ

Dans ce rapport, je présente le processus de sélection d'une méthodologie pour mesurer et comparer la maintenabilité d'un logiciel. L'utilisation d'une méthodologie simple, standard et reproductible permet de connaitre la situation présente, de suivre le progrès et de la communiquer à la direction. La méthodologie choisie sera testée sur deux logiciels.

TABLE DES MATIÈRES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CC	Complexité Cyclomatique
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
MI	Maintenability Index

INTRODUCTION

Une application va être passer la majorité de sa vie en maintenance. D'après le site Wikipedia, le développement d'une application représente moins de 20% du cycle de vie d'une application [1]. La durée des taches de maintenance est grandement impacté par la qualité du logiciel maintenu. Ce concept est appelé dette technique et est normalement calculée en utilisant un logiciel d'évaluation de la qualité du code comme Checkstyle, SonarQube et Logiscope. Ces logiciels produisent généralement un rapport détaillé et très technique avec comme auditoire des développeurs. Cependant, le résultat est difficilement présentable à un membre de la direction ou de l'exécutif et il est complexe de comparer des logiciels entre eux.

Lors de ce travail, je vais présenter le processus de sélection ainsi que la méthodologie choisit pour produire une analyse de la maintenabilité d'un logiciel.

CHAPITRE 1

CRITÈRES DE RECHERCHE

1.1 Critères de recherche : Simple et rapide à exécuter

Pour choisir le meilleur modèle possible, il faut au préalable définir les critères de succès. En surface, le modèle doit être applicable dans une entreprise. Pour avoir une bonne chance d'être adopté et maintenu, le modèle doit être peu couteux en temps. Ce type d'activité est normalement exécutée par un chef d'équipe. Un type ressource qui a déjà beaucoup de responsabilités et peu de temps libre. L'automatisation du calcul des résultats est donc un critère important.

1.2 Critères de recherche : Standard

Pour établir une crédibilité et facilité la discussion, il est important d'utiliser une terminologie, catégories et métriques provenant d'un standard ou d'une norme. De plus, il est préférable de suivre une méthodologie établie que de créer une nouvelle.

1.3 Critères de recherche : Simplifie la comparaison entre deux versions différentes

Le modèle doit permettre de comparer facilement deux versions d'un même logiciel et de comprendre si il y a eu amélioration ou régression. Il est important pour un chef d'équipe de savoir le plus rapidement possible si des changements récents ont eux un impact sur la maintenabilité d'une application. Pour permettre la comparaison, Le résultat doit être simple à comprendre et doit permettre de trouver rapidement la cause du problème ou la raison de l'amélioration.

1.4 Critères de recherche : Simplifie la comparaison entre deux logiciels différents

Le modèle doit permettre de comparer facilement deux logiciels différents. Cette activité a pour but de comparer un nouveau logiciel avec des logiciels connus par l'équipe. Avec de l'expérience sur un logiciel, l'équipe sait comment difficile un logiciel est à maintenir. Comme toute activité en logiciel, il est normalement trop couteux d'atteindre la perfection. En comparant un logiciel avec des logiciels connus pour être dur à maintenir et d'autres faciles à maintenir, il devient possible de déterminer un but à atteindre. De plus, il devient possible pour un directeur de comparer la performance de plusieurs équipes et de prendre des décisions en conséquence. Pour permettre la comparaison, le modèle doit avoir au final un résultat simple qui ne contient que quelque métrique.

1.5 Critères de recherche : Simple à communiquer à la direction

Pour être utile, toute activité technique dans une entreprise doit soit améliorer la qualité d'un logiciel, améliorer la vitesse de livraison ou communiquer un statut à la direction. L'activité de mesure du code permet d'améliorer la qualité d'un logiciel et de rendre un logiciel plus maintenable, donc plus rapide à modifier et livrer. Cependant, communiquer à la direction la qualité du code à aussi son avantage. Dans la situation qu'une équipe de maintenance hérite d'un logiciel peu maintenable, il est préférable pour le chef d'équipe de communiquer à la direction la situation. En la communicant, le chef d'équipe peut établir des attentes réalistes. Pour rendre cette communication efficace, le résultat du modèle doit être simple et peu technique.

CHAPITRE 2

COMPARAISON DE DEUX MODÈLE

2.1 Index de maintenabilité

Dans la publication « Using Metrics to Evaluate Software System Maintainability», Coleman *et al.* [2] proposent un index unique, pour calculer la maintenabilité d'un logiciel.
Cet index est calculé en utilisant la formule :
MI = 171–5.2 * ln(Métriques d'Halstead)– 0.23*Complexité Cyclomatique * 16.ln(ligne de code).

Un logiciel avec un index inférieur à 65 est considéré comme étant peu maintenable et un logiciel avec un index supérieur à 85 est considéré comme étant très maintenable. Un index entre les deux valeurs est considéré comme étant maintenable.

2.1.1 Simple et rapide à exécuter

Le calcul de l'index de maintenance est supporté par trois solutions commerciales, lattix, testwell et le plugin SQALE pour SonarQube. L'absence de solution ouverte ou non payante limite l'utilisation de ce modèle pour plusieurs compagnies. Pour les utilisateurs de .NET, la suite commerciale visual studio inclus par défaut une variante de l'index [3].

2.1.2 Standard

L'index de maintenabilité a créée par Oman et Hagemeister à l'université de l'Ohio avec la contribution de Hewlett Packard. L'index est reconnu et a été modifié par l'organiste gouvernemental "software enginerering institute". Avec le support de la communauté universitaire, de grande entreprise et d'organiste gouvernemental, on peut dire sans grand risque que l'index est un standard dans l'industrie.

2.1.3 Simplifie la comparaison entre deux versions différentes

L'index est simple à comprendre avec un seul résultat. L'analyse de la cause est simple avec seulement 3 métriques l'analyse qui influencent le résultat. Cependant, l'index présenté sans les métriques ne donne pas beaucoup d'information. Il est facile de savoir si une version est plus maintenable ou moins maintenable que la précédente, mais beaucoup plus difficile de comprendre pourquoi l'index a changé.

2.1.4 Simplifie la comparaison entre deux logiciels différents

Comme la comparaison entre deux versions différentes, l'index permet de savoir quel logiciel est le plus maintenable, mais rend très difficiles de savoir pourquoi.

2.1.5 Simple à communiquer à la direction

L'index est très simple à présenter à la direction. Avec un seul résultat et trois valeurs possibles, il est très simple de produire un statut par module et de produire un présentation sur le statut de la maintenabilité.

2.2 Modèle de Maintenabilité du SIG

Dans la publication, « A Practical Model for Measuring Maintainability » de Heitlager *et al.* [4] proposent un modèle alternatif pour mesurer la maintenabilité d'un logiciel. Le modèle diffère de l'index de maintenabilité sur plusieurs points. Le plus évidant est la présentation d'un résultat pour chacune des 4 caractéristiques de la maintenabilité du modèle ISO 9126. Les 4 caractéristiques sont la facilité d'analyse, la facilité de modification, la stabilité et la testabilité. Le résultat pour chaque caractéristique n'est pas une valeur numérique, mais entre 5 valeurs possibles. Les valeurs sont --, -, O, +, ++.

Pour calculer une valeur pour chaque caractéristique, le modèle utilise 5 métriques qui sont le nombre de ligne de code, nombre de ligne de code par unité, la complexité cyclomatique, la couverture du code et la duplication du code.

Une autre différence est comment chaque métrique est utilisée. Dans l'index de maintenabilité la moyenne est utilisée, tandis que le modèle du SIG utilise une distribution par méthode. Chaque méthode a un poids égale au nombre de ligne de code. Par exemple, en utilisant seulement la moyenne, un grand nombre de méthode très simple peuvent fausser le résultat. Dans le modèle du SIG, il est impossible d'avoir en haut de - pour la complexité si plus de 0% du code est dans une méthode très complexe.

Ce n'est pas le but de ce rapport de décrire en détail les formules et critères pour chaque métrique. Veuillez-vous référer à la publication de Heitlager *et al.* [4] pour plus détails.

Le modèle du SIG a été mis-à-jour pour suivre la norme ISO 25010. Cependant pour le manque de support dans les logiciels, cette analyse va être fait sur la première version.

2.2.1 Simple et rapide à exécuter

Le modèle est supporté par le plugin SIGMM pour SonarQube. Le plugin est complètement gratuit et demande une version de SonarQube supérieur à 3.2 et inferieur à 4.2. Il est très facile à générer des résultats sur un logiciel complexe. Malheureusement, le plugin est déprécié dans la version 4.2.

2.2.2 Standard

Le modèle est moins populaire comparé à l'index de maintenabilité, cependant il est plus proche de la norme ISO 9126 et avec la nouvelle version à la norme ISO 25010.

2.2.3 Simplifie la comparaison entre deux versions différentes

Il est plus simple de comparer deux versions différentes d'un logiciel. Avec 4 caractéristiques du standard ISO, le résultat est plus simple et parlant et ce même sans les valeurs des métriques.

2.2.4 Simplifie la comparaison entre deux logiciels différents

Comme la comparaison entre deux versions différentes, le modèle du SIG facilite la comparaison entre deux logiciels différents. Deux logiciels avec un index de maintenabilité identique peuvent avoir un résultat complètement différent avec le modèle du SIG.

2.2.5 Simple à communiquer à la direction

Malgré sa complexité plus élevée que l'index de maintenabilité, le modèle du SIG est assez simple à expliquer. Une moyenne des 4 caractéristiques peut être utilisée pour un sommaire ou on peut garder la granularité pour démontrer un avancement sur une caractéristique précise. Pour la présentation, un code de couleur peut être utiliser pour remplacer les résultats sous forme de plus et moins.

2.3 Choix

Le modèle du SIG a été choisit du principalement à sa simplicité d'exécution sans utiliser un logiciel commercial. De plus, sa facilité d'analyse a été prise en compte dans le choix. Sans l'existence du module du SIG MM dans SonarQube, le choix aurait été l'index de maintenabilité. La méthode utilisée par le SIG MM est beaucoup plus complexe à calculer.

CHAPITRE 3

APPLICATION DU MODÈLE

3.1 Méthodologie de l'application du modèle

Pour tester le modèle du SIG, nous avons choisit deux logiciels avec des caractéristiques différentes. Un logiciel mature avec des algorithmes complexes et une application de production de rapport développer en mode preuve de concept. Les deux applications ont été développées en java.

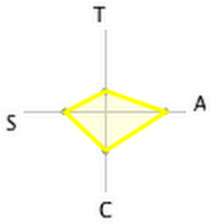
L'opinion générale est que le premier logiciel est très maintenable en général et contient certains modules très complexes qui pourraient être amélioré. Le deuxième logiciel est reconnue comme étant très peu maintenable.

Si le modèle est précis, nous devrions avoir deux résultats complètement différents pour les deux logiciels. De plus, le résultat pour le premier logiciel devrait démontrer que le maintenabilité du logiciel n'est pas égale dans tous les modules.

3.2 Résultat pour le logiciel #1

SIG Maintain. Model

(A)nalysability	+
(C)hangeability	0
(S)tability	0
(T)estability	-



Lines of code
14,604
19,790 lines
6,292 statements
268 files

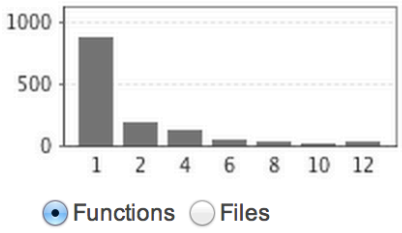
Classes
308
43 packages
1,354 functions
78 accessors

Duplications
4.3%
849 lines
57 blocks
25 files

Unit Tests Coverage
74.9%
76.7% line coverage
69.8% branch coverage

Unit test success
97.9%
11 failures
2 errors
626 tests
7:28 min

Complexity
2.4 /function
10.6 /class
12.2 /file
Total: 3,266



3.2.1 Nombre de ligne de code

Avec 6300 lignes de code java, le logiciel est considéré comme étant un petit logiciel et donc très maintenable ce qui nous confère une note de ++.

3.2.2 Nombre de ligne de code par unité

Malheureusement, SonarQube ne donne aucune vue rapide sur le nombre de ligne de code par unité. Il n'y a aucune alerte pour une méthode trop grande ou un histogramme sur la distribution du nombre de méthode par nombre de ligne de code. Au plus nous avons droit à un agrégat des résultats. Avec plus de 1300 méthodes, nous avons une moyenne de ~5 lignes de code par méthode. Cette moyenne est excellente, cependant une ou des méthodes très longue pourraient affecter le résultat. En analysant le résultat sur les caractéristiques, il semble que le résultat pour cette métrique soit -.

3.2.3 Duplication du code

Un taux de duplication de 4.3 est considéré comme bon, donc nous avons un résultat de +.

3.2.4 Complexité

Le logiciel a une excellente moyenne de 2.4CC par méthode. Cependant, une seul méthode avec 85CC nous donne le résultat -. Comme discuté dans la section 2, une seule méthode très complexe nous limite à ce résultat.

3.2.5 Couverture des tests

Une couverture des tests de 74.5% est considérée comme étant moyenne et nous donne un résultat O.

3.2.6 Résultat

	Volume	Complexité	Duplication	Ligne de code par méthode	Couverture des tests	Résultat
la facilité d'analyse	++		+	-	o	+
la facilité de modification		-	+			o
la stabilité					o	o
la testabilité		-		-	o	-

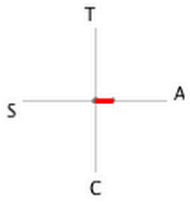
3.2.7 Analyse du résultat

Le résultat attendu était légèrement supérieur à celui produit par le modèle du SIG. L'impact de quelques méthodes très complexes est très présent surtout pour la métrique de complexité. Une méthode avec une complexité de 10CC est normalement trop complexe et le modèle ne pardonne pas si une méthode est en haut de 20CC.

3.3 Résultat du logiciel #2

SIG Maintain. Model

(A)nalysability	-
(C)hangeability	--
(S)tability	--
(T)estability	--



Lines of code
22,110
25,854 lines
12,789 statements
20 files

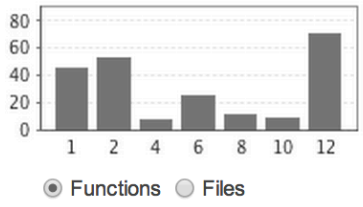
Classes
20
3 packages
219 functions
18 accessors

Duplications
50.4%
13,035 lines
795 blocks
15 files

Unit Tests Coverage
-

Unit test success
0 tests

Complexity
21.8 /function
238.4 /class
238.4 /file
Total: 4,767



Documentation
52.6% docu. API
76 public API
36 undocu. API

Comments
7.3%
1,730 lines

3.3.1 Nombre de ligne de code

Avec 12300 lignes de code java, le logiciel est considéré comme étant un petit logiciel et donc très maintenable ce qui nous confère une note de ++.

3.3.2 Nombre de ligne de code par unité

Dans l'article "A Practical Model for Measuring Maintainability" de Heitlager *et Al*. il y a aucune mention de combien de ligne de code une méthode doit avoir. Cependant, une moyenne de 58 lignes de code par méthode est de loin beaucoup trop élevée. Nous avons donc comme résultat --.

3.3.3 Duplication du code

Le taux de duplication de code est de 50.4%. Ce taux est de loin trop élevé, donc nous avons comme résultat --. Pour avoir une note de -, il faudrait réduire le taux à 20%.

3.3.4 Complexité

Avec une moyenne de 21.8CC par méthode, nous avons une note de -- automatiquement. Pour atteindre un meilleur résultat, il faudrait donc réécrire l'application au complète.

3.3.5 Couverture des tests

Il n'y a aucun test unitaire dans l'application, ce qui nous donne un résultat de --.

3.3.6 Résultat

	Volume	Complexité	Duplication	Ligne de code par méthode	Couverture des tests	Résultat
la facilité d'analyse	++		--	--	--	-
la facilité de modification		--	--			--
la stabilité					--	--
la testabilité		--		--	--	--

3.3.7 Analyse du résultat

Comme attendu, le modèle démontre que le logiciel est peu maintenable. En utilisant les métriques dans notre analyse, on arrive à la conclusion que l'application doit être complètement réécrite et que la dette technique est beaucoup trop élevée. Le résultat de cette analyse a été présenté à la direction et a été très bien reçu. Le développement de nouvelle fonctionnalité a été réduit de 50% et l'effort a été réinvestit dans la réécriture de module. La production de ce rapport ce fait maintenant sur une base hebdomadaire.

CONCLUSION

Le modèle du SIG répond au critère de recherche initiale. Le modèle est facile à utiliser, à analyser et à communiquer. Il fait maintenant partit du processus de planification. Cependant, le modèle est loin d'être parfait. Une seule méthode complexe peut donner un mauvais résultat. Cette méthodologie est meilleure qu'une simple moyenne, cependant elle est très punitive. Pour l'automatisation, la nouvelle version du modèle n'a pas encore été automatisée. La version implémentée par SonarQube n'est plus supportée dans le futur. Sans une automatisation, le modèle est beaucoup trop complexe a calculé à la main.

Nous allons chez Guavus Canada continuer a utilisé le modèle dans un future proche, cependant nous allons continuer la recherche pour ne pas être pris au dépourvu si le plugin de SonarQube ne fonctionne plus.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

[1] http://en.wikipedia.org/wiki/Software_maintenance

[2] Coleman, D.; Lowther, B.; and Oman, P.; Using Metrics to Evaluate Software System Maintainability, IEEE Computer, Vol. 27(8), pp. 44-49, Aug. 1994

[3] http://blogs.msdn.com/b/zainnab/archive/2011/05/26/code-metrics-maintainability-index.aspx

[4] Heitlager I., Kuipers T., and Visser J. (2007) "A Practical Model for Measuring Maintainability", 6th Int'l Conf. on the Quality of Information and Communications Technology (QUATIC '07), IEEE Computer Society Press.