PROJET DE SESSION

Présenté à :

Alain April
Professeur
MGL804 Hivers 2011

Par:

Guy Bertrand
Code d'accès universel : AJ81310
Code permanent à l'ETS: BERG17016106

Table des matières

1) INTRODUCTION 4

2) LES STANDARDS ISO APPLICABLES 4

ISO 9126 4

ISO 14598 4

ISO 25000 4

3) LOGICIELS D'ÉVALUATION DE LA QUALITÉ LOGICIEL 5

« Définition de l'analyse statique 5

« Définition de l'analyse dynamique 5

Logiciels d'évaluation de la qualité : que font-ils? 5

a) règles de codage (pattern-based/rules-based static analysis) 5

b) arbre syntaxique du code (data flow static analysis) 6

c) métriques (code metrics calculation) 6

4) LES LOGICIELS D'ÉVALUATIONS 6

Checkstyle 6

PMD 7

Logiscope 7

Revue des capacités des 3 logiciels d'évaluations 8

Méthode d'utilisation/activation 8

5) LE CODE SOURCE UTILISÉ ET LA PREMIÈRE ANALYSE 10

Les résultats préliminaires 10

Avec CheckStyle et PMD 10

Logiscope 10

1ère Observations : 12

Un deuxième survol des résultats des analyses 13

Compétence et expertise requise pour bien utiliser ce genre d'outil 13

2ième Observation : 14

Ce genre d'outil sont-ils utilisés? 14

Pour 14

Contre 15

Limite des logiciels d'évaluation de la qualité 15

3ième Observation : 15

Conclusion..... *16*

Référence..... *16*

1) INTRODUCTION

Ce document est la deuxième partie de mon projet de session pour le cours MGL804 – Hivers 2011. La première partie consiste en une présentation PowerPoint, qui sera envoyé dans le même courriel avec ce document au professeur. Celle-ci a été présentée à mes confrères et consœurs de classes, ainsi que notre professeur Alain April jeudi le 3 mars, 2011.

Le travail de session choisi est :

> « #10 - Faites une analyse de la maintenabilité d'un logiciel à l'aide de logiciels d'évaluation de la qualité (Checkstyle, Logiscope, etc.). Suivez l'approche proposée en classe qui décrit les étapes à suivre pour effectuer une analyse reproductible, impartiale et objective. » (April & Abran, 2006)(p58)

2) LES STANDARDS ISO APPLICABLES

Les standards ISO applicables à la qualité et l'évaluation de la qualité logiciels sont les suivants :

ISO 9126 et 14598, et la norme plus récente qui englobe ceux-ci : ISO 25000.

ISO 9126

Définit la qualité d'un logiciel comme la combinaison de six caractéristiques de base.

1. *Capacité fonctionnelle*
2. *Fiabilité*
3. *Facilité d'utilisation*
4. *Rendement / Efficacité*
5. *Maintenabilité*
6. *Portabilité*

Ce standard donne un ensemble de mesure interne et externe pour la mesure de la qualité du logiciel

ISO 14598

Définit un processus d'évaluation de la qualité d'un logiciel

Ce standard donne un cadre pour la planification et l'exécution des processus de mesures utilisant la norme ISO 9126

ISO 25000

ISO "SQuaRE Software Quality Requirements and Evaluation"

Remplace et intègre les standards ISO 9126 et 14598.

- *Donne une référence pour l'évaluation de la qualité d'un logiciel (ISO 9126)*
- *Définit comment doit être évalué un logiciel selon cette référence (ISO 14598)*

Nota : sommaire définition (Laudrel)

3) LOGICIELS D'ÉVALUATION DE LA QUALITÉ LOGICIEL

Il existe deux types d'évaluation de la qualité logicielle :

1. *Statique*
2. *Dynamique*

« Définition de l'analyse statique *:*

En informatique, la notion d'analyse statique de programmes couvre une variété de méthodes utilisées pour obtenir des informations sur le comportement d'un programme lors de son exécution sans réellement l'exécuter. L'analyse statique est utilisée pour repérer des erreurs de programmation ou de conception, mais aussi pour déterminer la facilité ou la difficulté à maintenir le code. » (Analyse statique de programmes)

« Définition de l'analyse dynamique *:*

L'analyse dynamique de programmes est une analyse réalisée sur un programme informatique en l'exécutant sur un vrai processeur ou un processeur virtuel. Pour que l'analyse dynamique de programme produise des résultats intéressants, le programme cible doit être exécuté avec des entrées suffisamment variées. L'utilisation de techniques de test logiciel telles que la couverture de code aide à s'assurer qu'un ensemble adéquat des comportements possibles du programme a été observé. Il faut également s'assurer de limiter autant que possible les effets que l'instrumentation a sur l'exécution (y compris sur les propriétés temporelles) du programme cible. » (Analyse dynamique de programmes)

Pour cette évaluation, seulement l'analyse statique sera utilisée. Les logicielles d'analyses CheckStyle, PMD et Logiscope sont de type statique.

Logiciels d'évaluation de la qualité : que font-ils?

Les logicielles d'évaluations de la qualité statique utilisent principalement les méthodes suivantes :

a) *Les règles de codage*
b) *L'arbre syntaxique du code*
c) *Plusieurs métriques*

a) règles de codage (pattern-based/rules-based static analysis)

L'objectif primaire des règles de codage est la détection et prévention d'erreurs. Ce genre d'analyse utilise plusieurs techniques, tel que :

- *Recherche de séquence de code qui ont le potentiel de causé des failles*
- *Normes pour éviter l'utilisation de certaines mauvaises pratiques*
- *Vérification du bon suivi des règles de codage (règle générale et de sa propre compagnie)*

- *La convention de codage spécifique à un langage ou environnement, tel que « The Java Language Specification » d'Oracle/Sun ou « Guide francophone des conventions de codage pour la programmation en langage Java » du site developpez.com*

Nota : une partie des items ci-haut traduit de l'anglais (Parasoft Corporation, 2008)

D'après le groupe « Open Web Application Security Project » (OWASP), approximativement 70% des problèmes de sécurité peuvent être évité avec utilisant l'analyse statique pour appliquer une bonne validation de l'entrée de donnée. (Parasoft Corporation, 2008) traduction

b) arbre syntaxique du code (data flow static analysis)

L'objective de l'analyse de l'arbre syntaxique du code est la détection d'erreurs en découvrant tous les chemins possibles dans le code. C'est une simulation de l'exécution du code sans l'exécuter et l'identification d'erreurs possibles si un chemin particulier est exécuté.

Traduction libre plusieurs sources (Parasoft Corporation, 2008) (IBM)

Cependant, l'analyse de l'arbre syntaxique du code n'est pas fiable à 100% et peut donner de faux résultat (aussi connue comme faux positif).

c) métriques (code metrics calculation)

Il existe plusieurs métriques. Le site Wikipédia offre cette liste (en anglais) (Software Metrics)

• *Balanced scorecard*	• *Bugs per line of code*
• *Code coverage*	• *Cohesion*
• *Comment density[1]*	• *Connascent software components*
• *Coupling*	• *Cyclomatic complexity (McCabe's complexity)*
• *Function point analysis*	• *Halstead Complexity*
• *Instruction path length*	• *Number of classes and interfaces*
• *Number of lines of code*	• *Number of lines of customer requirements*
• *Program execution time*	• *Program load time*
• *Program size (binary)*	• *Robert Cecil Martin's software package metrics*
• *Weighted Micro Function Points*	

4) LES LOGICIELS D'ÉVALUATIONS

Checkstyle

Le site web de Checkstyle offre ce sommaire de l'outil :

"Checkstyle is a development tool to help programmers write Java code that adheres to a coding standard. It automates the process of checking Java code to spare humans of this boring (but important) task. This makes it ideal for projects that want to enforce a coding standard." (Checkstyle)

L'outil Checkstyle vérifie que le Java et vient de source libre.

PMD

Le site web de PMD offre cette description pour celui-ci:

"PMD scans Java source code and looks for potential problems like:
-Possible bugs - empty try/catch/finally/switch statements
-Dead code - unused local variables, parameters and private methods
-Suboptimal code - wasteful String/StringBuffer usage
-Overcomplicated expressions - unnecessary if statements, for loops that could be while loops
-Duplicate code - copied/pasted code means copied/pasted bugs" (PMD)

PMD, comme Checkstyle, vérifie que le Java et vient de source libre

Logiscope

Logiscope est un produit commercial de la compagnie IBM. Le site web d'IBM offre cette description :

« IBM Rational Logiscope est un outil d'assurance qualité logicielle qui automatise les analyses de code ainsi que l'identification et la détection des modules exposés aux erreurs lors du test de logiciels. » (Logiscope)

Logiscope peut vérifier C, C++, Java et l' ADA. Ce logiciel est divisé en 4 composantes :

- *Rule Checker*

"RuleChecker: check that the source code complies with a set of defined coding standards and best practices leading to a satisfactory level of Maintainability, Reliability or Portability;" (IBM Logiscope Rule Checker and Quality Checker Getting Started PDF)

- *Quality Checker*

"QualityChecker: use source code metrics to locate complex, error prone modules, analyse graphic results to assess architecture of the application and detailed design of functions" (IBM Logiscope Rule Checker and Quality Checker Getting Started PDF)

- *Test Checker*

"Rational Logiscope TestChecker supports structure-based testing and test coverage analysis (cf. [DO178B]. It quantifies test completeness and shows uncovered source code paths. By uncovering bugs hidden in untested source code, Rational Logiscope CodeReducer improves program reliability.

Rational Logiscope TestChecker is based on source code instrumentation techniques that are adaptable to your test environment constraints, both on host and target platforms." (IBM Logiscope Basic Concepts)

- *CodeReducer*

"CodeReducer is a code similarity search tool, that can satisfy different needs:
– Search for all similar pieces of code in a given set of files,
– Search for code similar to a reference code in a given set of files,
– Comparison of source code files,
– Search for differences between two versions of a set of source code files." (IBM Logiscope Code Reducer Identifying Code Similarities PDF)

Revue des capacités des 3 logiciels d'évaluations

	règles de codage	*arbre syntaxique du code*	*métrique*
CheckStyle	*Oui*	*Non*	*Non*
PMD	*Oui*	*Oui*	*Oui*
Logiscope	*Oui*	*Oui*	*Oui*

Table 1 - Sommaire des capacités des logiciels d'évaluations

Méthode d'utilisation/activation

CheckStyle et PMD ont 2 méthodes d'utilisation :

1) *Ils peuvent être intégrés dans un outil de développement intégré, tel qu'Eclipse*

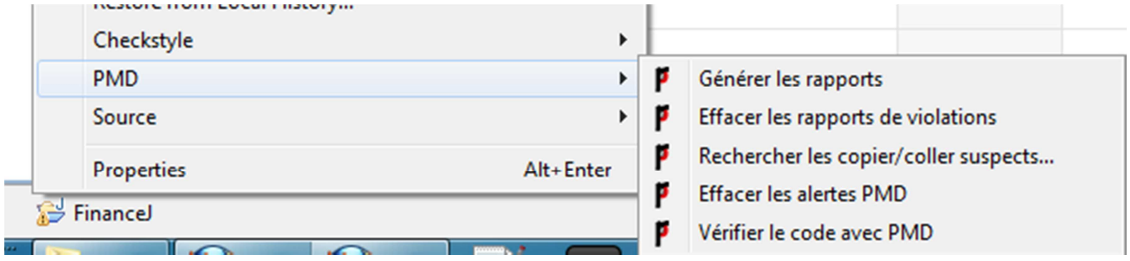

Figure 1 - Eclipse activation de CheckStyle et PMD

2) *Ils peuvent être exécute en mode « batch » pendant la compilation*
-exemple pour CheckStyle

```
java -D<property>=<value> \
   com.puppycrawl.tools.checkstyle.Main \
```

-c <configurationFile> \
[-f <format>] [-p <propertiesFile>] [-o <file>] \
[-r <dir>] file...

Le logiciel Logiscope a une interface usager spécifique à lui-même, mais peut aussi être intégré dans un processus de compile (batch).

"Logiscope batch

Logiscope batch is a tool designed to work with Logiscope in command line to:
• instrument the source code files specified in a Logiscope project: i.e. ".ttp" file,
• generate reports in HTML and/or CSV format automatically." (IBM)

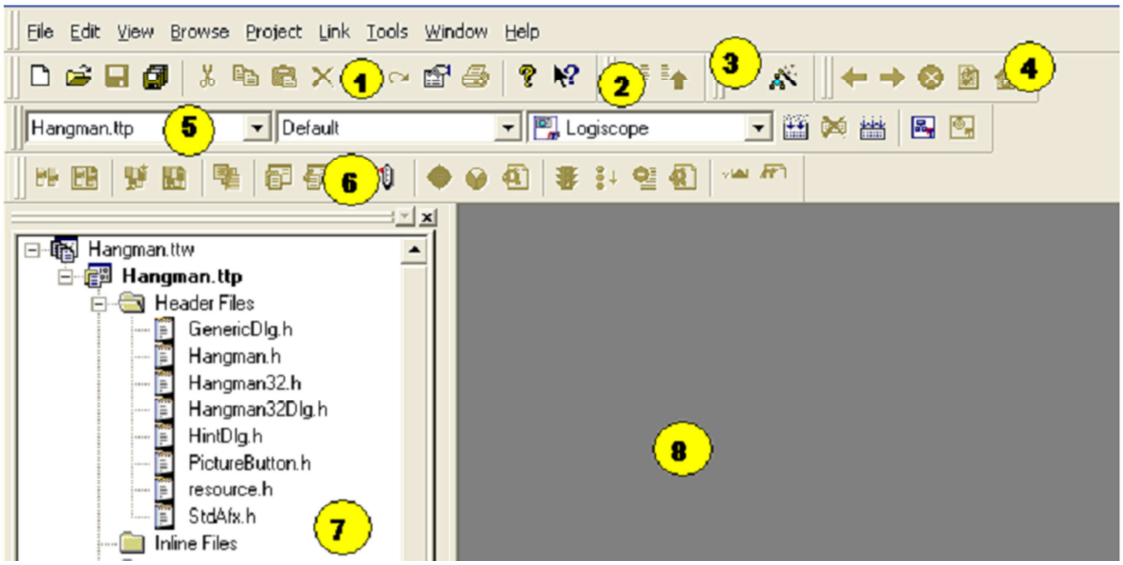

Figure 2 Logiscope interface usager

(IBM)

5) LE CODE SOURCE UTILISÉ ET LA PREMIÈRE ANALYSE

Trois codes source ont été utilisés pour ce projet :

1) *FinanceJ – code source utilisé pour les travaux pratiques du cours MGL804*
2) **uploadit-gwt* - logiciel propriétaire de l'employeur de l'étudiant écrit en Java âgé de 3 ans*
3) **IFOOSL* - logiciel propriétaire de l'employeur de l'étudiant écrit en Java âgé de plus de 10 ans écrit par des développeurs en C (procédurale) qui ont tenté leurs premières applications orienté objet*

Les résultats préliminaires

Avec CheckStyle et PMD

	Métriques du logiciel	**Avertissements**			
		Eclipse seulement	*Avec CheckStyle*	*Avec PMD*	*Total*
FinanceJ	*13 classes* *97 méthodes* *1709 lignes*	*44 avertissements*	*1022*	*89 erreurs* *428 avertissements* *99 autres/informations*	*1774*
Uploadit-gwt	*125 classes* *692 méthodes* *5,582 lignes*	*17 avertissements*	*4,544*	*67 erreurs* *1,268 avertissements* *289 autres/informations*	*6,148*
IFOOSL	*140 classes* *860 méthodes* *17,459 lignes*	*558 avertissements*	*23,819*	*378 erreurs* *4,292 avertissements* *856 autres/informations*	*28,787*

Table 2 - sommaire des résultats préliminaires CheckStyle et PMD

Si on regarde strictement le logiciel FinanceJ, il y a presque un avertissement par ligne de code . IFOOSL donne presque 2 avertissements par ligne de code. Le nombre d'avertissement est astronomique et décourageant.

Logiscope

Logiscope Quality Checker nous donne un format différent, c'est-à-dire un tableau avec un sommaire des valeurs, et plusieurs graphiques pour représenter différents aspect de la qualité du logiciel.

Voici les résultats pour FinanceJ :

Mnemonic	**Metric Name**	**Min**	**Max**	**Value**
ap_comf	*Application comment rate*	*0.20*	*+oo*	*0.09*
ap_eloc	*Application effective lines of code*	*0*	*200000*	*1728*
ap_func	*Number of application functions*	*0*	*3000*	*130*
ap_inhg_levl	*Depth of inheritance tree*	*1*	*5*	*2*
ap_scomm	*Number of lines of comments*	*-oo*	*+oo*	*216*
ap_sline	*Number of lines*	*0*	*300000*	*2471*
ap_sloc	*Number of lines of code*	*-oo*	*+oo*	*1979*
ap_ssbra	*Number of lines with lone braces*	*-oo*	*+oo*	*251*
ap_stat	*Number of statements*	*0*	*100000*	*1011*

ap_vg	*Sum of cyclomatic numbers of the application functions*	*0*	*6000*	*245*
ap_wmc	*Average complexity of functions*	*1.00*	*3.00*	*1.88*

Table 3 - résultat Logiscope Qualité pour FinanceJ

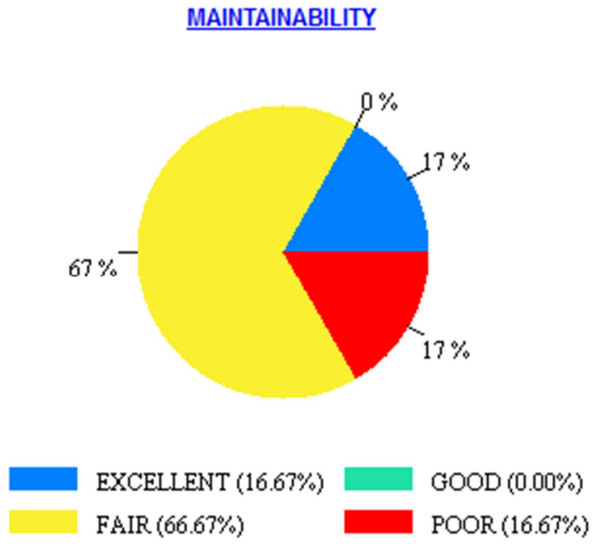

Figure 3 Logiscope Maintenabilité FinanceJ

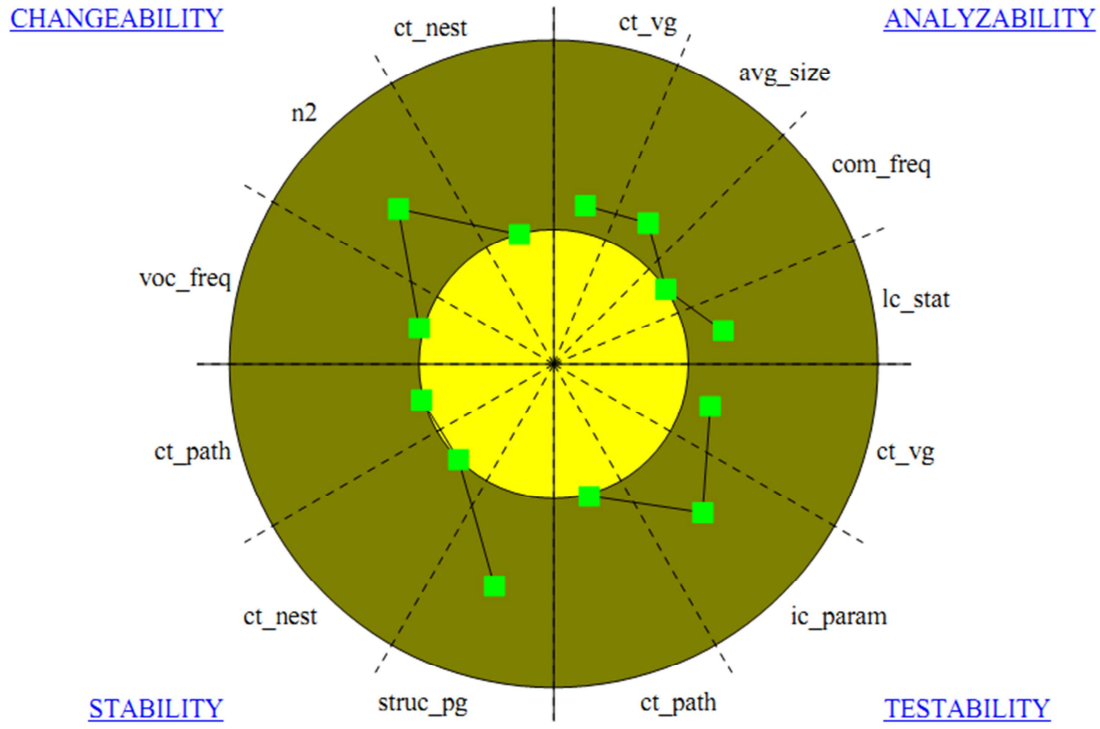

Figure 4 - graphique Kiveat pour FinanceJ

Les résultats de Logiscope ne sont guère plus encourageants. Plus de 82% du code est considéré comme médiocre ou mauvais. Le graphique de Kiveat pour FinanceJ démontre aussi que plusieurs aspects laissent à désirer. Les rapports Logiscope de qualité et violation des règles pour FinanceJ seront inclus dans le courriel envoyé au professeur avec ce document.

NOTE : LES DIFFÉRENTS MORCEAUX DE LA TARTE DU GRAPHIQUE KIVEAT REPRÉSENTE DES MÉTRIQUES QUE LOGISCOPE CALCULE. EXEMPLE : STRUCT-PG = « NUMBER OF VIOLATIONS OF STRUCTURED PROGRAMMING ».

1ère Observations :

- *Faire une analyse sur un logiciel existant va donner une quantité décourageante d'avertissements et erreurs. Donc il faut commencer à utiliser ce genre d'outil dès la conception du logiciel.*
- *Il faut aussi étudier les résultats. Ils y en a qui sont très faciles à corriger, et donneront des résultats positifs très rapidement. Un exemple est une des règles de Checkstyle, qui spécifie qu'il ne devrait pas avoir d'espace à la fin d'une ligne. Il suffit d'une ligne de commande (en Unix) pour corriger cette situation sur toutes les lignes dans tous les codes source. (> 500 améliorations dans FinanceJ)*
- *Quelqu'un avec peu d'expertise en gestion de développement pourrait avoir l'idée de corriger tous ces erreurs avant de continuer. L'approche « Big Bang » ne fonctionne jamais, car c'est simplement une tâche beaucoup trop longue et c'est un effort qui sera certainement abandonné. Il faut se prendre par petit pas, possiblement en encourageant les programmeurs de corriger une classe, ou plusieurs méthodes par jour ou en même temps qu'ils font des « check-in » de code source.*

De plus, ce genre de correction est ennuyant pour un programmeur d'expérience. La correction d'avertissement de syntaxe tel que « espace à la fin de la ligne » se situe dans la zone bleue du graphique ci-dessous.

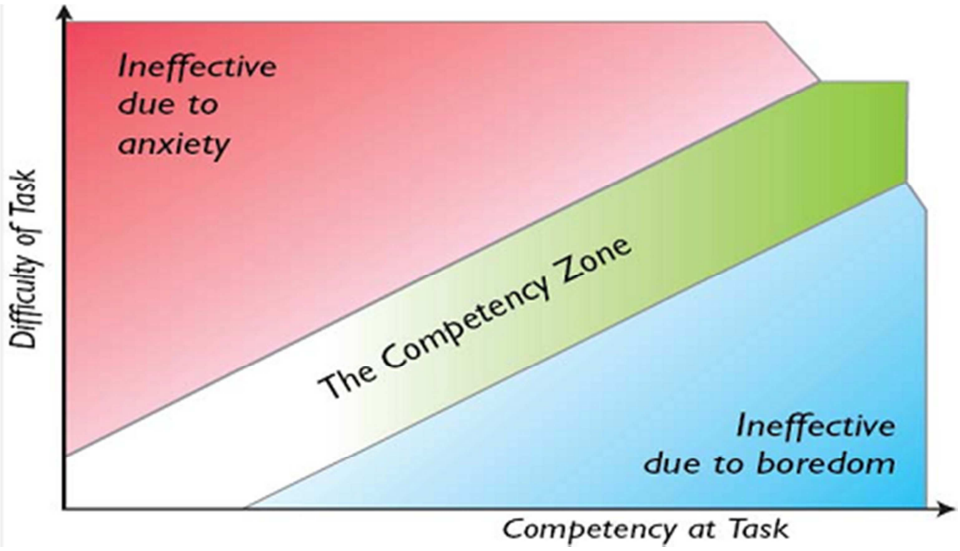

Figure 5 - Psychology of Engineering of Developers

(Armour, 206) (Csikszentmihalyi, 1990) (Huizinga & ECS)

Cependant, si ces outils sont utilisés dès le début, le nombre d'erreur sera au minimum et même inexistant. Ceci contribue à conserver les programmeurs dans leur zone de compétence.

Un deuxième survol des résultats des analyses

Chaque outil est configurable. On peut choisir d'exécuter tous les tests, ou être plus sélectif.

De plus, chaque test peut être configuré ou ajuster pour donner moins de résultat ou qu'une portion de ceux-ci.

Si on regarde le tableau (Table 3 - résultat Logiscope Qualité pour FinanceJ), les valeurs permises pour « nombre de ligne » est de 0 à 300 000. Un projet de 300 000 lignes est monstrueux et devrait être divisé en plusieurs projets séparé. Il est donc préférable de modifier la limite supérieure à un niveau qui est plus acceptable pour l'environnement/groupe/compagnie.

De plus, on a l'impression que les différents outils de vérification donnent des avertissements différents pour la même ligne de code. C'est normal, car il ne vérifie pas nécessairement la même chose ou de la même façon. Il existe beaucoup d'autres outils de vérification de la qualité du code, chacun ayant sa façon de faire.

Expérience personnel: Une application écrite en C qui est compilé avec un paramètre « +w2 display all compiler warnings » donnent des résultats très différents sur chaque plateforme (HP-UX, Solaris, AIX, Windows et Linux). Aucun des compilateurs ne fait défaut. Ils sont corrects pour leurs plateformes.

Compétence et expertise requise pour bien utiliser ce genre d'outil

Il est facile de commencer à configurer ces logiciels, surtout pour réduire le nombre d'avertissements initiaux, mais pour bien utiliser ces outils, il faut aussi connaitre le code source, et aussi avoir une expertise avancé dans la programmation.

Ceci est plutôt important pour étudier et valider les métriques reliés à la complexité, tel que la complexité Cyclomatique (McCabe's complexity) et la complexité d'Halstead. Même si le résultat de ces 2 métriques pour notre code pourrait être haut, il faut étudier, évalué et bien connaitre cette section de code pour conclure si c'est un problème à corriger ou non.

Il ne faut pas négliger les efforts requis pour corriger certains problèmes. Dans la majorité des logiciels, ils existent des défauts qui sont contournés (« work-around ») ou ne rencontre pas les normes de complexité. Ce genre de problème est trop couteux et requiert trop de temps à corriger. Le logiciel fonctionne bien, répond en majorité aux exigences des usagers et sont, plus ou moins, impossible à corriger ou améliorer.

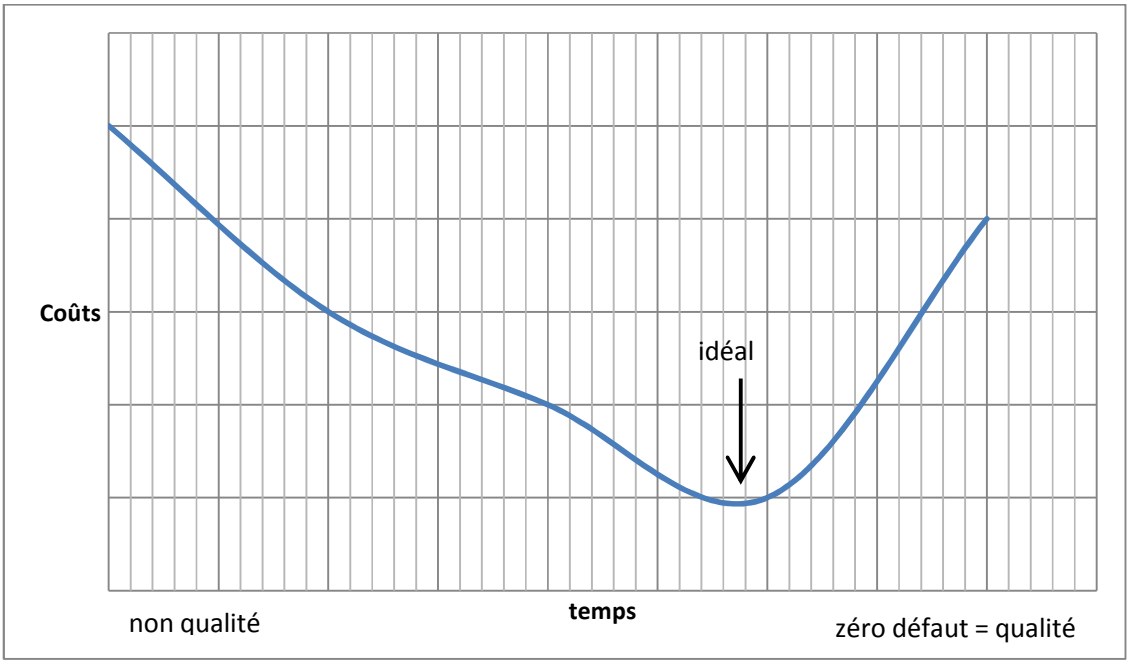

Figure 6 - qualité, effort et temps

Cependant, l'expertise requise pour prendre cette décision est grande. Un programmeur débutant n'a pas la capacité de faire cette analyse.

En étudiant la document du logiciel Logiscope, on remarque l'identification du rôle nommé « Logiscopian Quality Engineer ». On remarque aussi qu'IBM offre de la formation pour Logiscope (Formation Logiscope) et celle-ci est de «Niveau de compétence Avancé».

2ième Observation :

L'utilisation de ces outils devrait être faite de préférence par des programmeurs d'expérience avec une expertise pour bien utilisé les outils et d'être capable d'interpréter les résultats.

Ce genre d'outil sont-ils utilisés?

À titre d'intérêt, une petite recherche a été faite sur l'Internet sur l'utilisation de ce genre d'outil. Le résultat était surprenant : il y a autant de commentaire pour ce genre d'outil qu'il y en a contre.

Voici un échantillon :

Pour

- *Les « vraies programmeurs Macho » n'utilisent pas d'outils de vérification*
- *La configuration des outils est trop complexe et longue*
- *"It is time consuming if conducted manually"*
- *"Automated tools do not support all programming languages"*

- *“Automated tools produce false positives and false negatives”*
- *“There are not enough trained personnel to thoroughly conduct static code analysis”*
- *“Automated tools can provide a false sense of security that everything is being addressed”*
- *“Automated tools only as good as the rules they are using to scan with”*
- *“It does not find vulnerabilities introduced in the runtime environment”*
- *“The build fails if a violation is found. So yeah, it can be annoying. My beef is having to "switch gears" and get out of the flow of whatever I was doing. It's worth it if I find something of concern. Not so much if I don't”*
- *“nice sanity check to consult once you have a bug and can't figure it out”*

Contre

- *les outils peuvent trouver un bogue suptile qui prendrait une éternité à trouver par un humain*
- *Permets de réduire les besoins de main-d'oeuvre*
- *“at least 60% of the run time bugs were caught before compilation”*
- *“Current static analysis tools are capable of performing deep interprocedural analyses, an can automatically identify for example procedure pre- and post-conditions. This can be a great help for later code reviews as well.”*
- *“It can find weaknesses in the code at the exact location.”*
- *“It can be conducted by trained software assurance developers who fully understand the code”*
- *“It allows a quicker turn around for fixes”*
- *“Automated tools can scan the entire code base relatively fast”*
- *“Automated tools can provide mitigation recommendations, reducing the research time”*
- *“It permits weaknesses to be found earlier in the development life cycle, reducing the cost to fix”*
- *“Static code analysis usually uncovers bad style and buggy programming habits. After a while, you will learn to write code that does not flag analysis errors. There is less time "wasted" on fixing issues, and you will learn to write better code”*

(Does static code analysis save time?) (Parasoft Corporation, 2008) (Static vs dynamic code analysis: advantages and disavantages) (What are the real benefits of static code analysis)

Limite des logiciels d’évaluation de la qualité

D’après le Dr.Kolawa, même si on utilise très bien les outils de d’analyse statique de code, il y a des erreurs qui ne pourront pas être découvert. Selon lui, “Most bugs are related to poorly-implemented requirements, missing requirements, or confused users, and cannot be identified without involving human intelligence” (Parasoft Corporation, 2008)

De plus, le “Software Engineering Institute” mentionne que “ 70% of tools purchased by the organizations in the surveys are never used, other than perhaps in initial trial” (Process Overview)

3ième Observation :

L’intégration des outils d’analyse statique de code dans le processus journalier de développement est critique. Les programmeurs doivent se servir de ces outils régulièrement, et/ou un processus

d'automatisation doit être appliqué dans les procédures du département et doivent être visible/annoncer.

Comme démontre les analyses faites pour projet, faire une analyse à la fin du cycle de développement donne des résultants décourageants. Il faut faire ces analyses régulièrement et rigoureusement.

Conclusion

Pour bien utilisez les outils d'analyse statique de code :

- *Assurez-vous d'intégrer les outils dans vos environnements de développement intégré (IDE) et dans un processus automatique avec visibilité*
- *Planifier une courbe d'apprentissage pour augmenter vos connaissances et compétences dans l'utilisation, la configuration et l'interprétation des résultats des outils.*
- *Il faut inclure des étapes de mesure de la qualité du code dans vos projets. Il faut les réévaluer régulièrement.*

Donc est-il possible d'utiliser un logiciel d'analyse statique de code pour évaluer la maintenabilité d'un logiciel? Oui. Et certain des logiciels, tel que Logiscope, offre un calcul sur ceci.

Cependant, les résultats doivent être étudiés par un programmeur d'expérience, qui connait aussi l'utilisation de ces logiciels d'analyses. (« Logiscopian Quality Engineer »).

Pour ceux qui font cette analyse avec une base de code source existant, et se retrouve avec des milliers d'avertissement, le philosophe chinois Lao-Tseu offre ceci :

> ***Un voyage de mille lieues commence toujours par un premier pas.***

LAO-TSEU (V. 570-V. 490 AV. J.-C. OU V. IVE SIÈCLE AV. J.-C.)
PHILOSOPHE CHINOIS CONSIDÉRÉ PAR LA TRADITION COMME LE FONDATEUR DU TAOÏSME.

Référence

Illustrations / Figure

Figure 1 - Eclipse activation de CheckStyle et PMD 8
Figure 2 Logiscope interface usager 9
Figure 3 Logiscope Maintenabilité FinanceJ 11
Figure 4 - graphique Kiveat pour FinanceJ 11
Figure 5 - Psychology of Engineering of Developers 12
Figure 6 - qualité, effort et temps 14

Tableaux

Table 1 - Sommaire des capacités des logiciels d'évaluations 8
Table 2 - sommaire des résultats préliminaires CheckStyle et PMD 10
Table 3 - résultat Logiscope Qualité pour FinanceJ 11

Travaux cités

Analyse dynamique de programmes. *(s.d.). Récupéré sur Wikipedia: http://fr.wikipedia.org/wiki/Analyse_dynamique_de_programmes*

Analyse statique de programmes. *(s.d.). Récupéré sur Wikipedia: http://fr.wikipedia.org/wiki/Analyse_statique_de_programmes*

April, A., & Abran, A. (2006). Améliorer le maintenance du logiciel. *Loze-Dion.*

Armour, P. G. (206). "The business of Software", Communications of the ACM, June 2006, Vo.49, No. 6.

Checkstyle. *(s.d.). Récupéré sur Checkstyle: http://checkstyle.sourceforge.net/*

Csikszentmihalyi, M. (1990). "Flow: The Psychology of the Optimal Experience". *Harper & Row.*

Does static code analysis save time? *(s.d.). Récupéré sur Javalobby: http://www.javalobby.org/java/forums/t105457.html*

ETIEVANT, H. (2004). Guide francophone des conventions de codage pour la programmation en langage Java. *Récupéré sur developpez.com Club des professionnels de l'informatique: www.cyberzoide.net*

Formation Logiscope. *(s.d.). Récupéré sur IBM: http://www-304.ibm.com/jct03001c/services/learning/ites.wss/ca/fr?pageType=course_description&courseCode=QT110CE*

Huizinga, D., & ECS. (s.d.). Putting the Engineering in Software Engineering.

IBM. (s.d.). Logiscope.

IBM Logiscope Basic Concepts. (s.d.).

IBM Logiscope Code Reducer Identifying Code Similarities PDF. (s.d.).

IBM Logiscope Rule Checker and Quality Checker Getting Started PDF. (s.d.).

Laudrel, S. (s.d.). Processus et standards. *Récupéré sur Assurance qualité: http://assurancequalite.voila.net/QA_general_fr.pdf*

Logiscope. *(s.d.). Récupéré sur IBM: http://www-142.ibm.com/software/products/fr/fr/ratilogi/*

Logiscope. *(s.d.). Récupéré sur IBM France: http://www.ibm.com/software/products/fr/fr/ratilogi/*

Parasoft Corporation. (2008, 08 07). An Interview with Adam Kolawa, Ph.D. *Récupéré sur The Code Project: http://www.codeproject.com/KB/interviews/Code_Review.aspx*

PMD. *(s.d.). Récupéré sur PMD: http://pmd.sourceforge.net/*

Process Overview. *(s.d.). Récupéré sur Software Engineering Institute SEI: http://www.sei.cmu.edu/intro/documents/intro-slides/process-overview.pdf*

Software Metrics. *(s.d.). Récupéré sur Wikipedia: http://en.wikipedia.org/wiki/Software_metric*

Static vs dynamic code analysis: advantages and disavantages. *(s.d.). Récupéré sur GCN The online authority for government IT professionals: http://gcn.com/articles/2009/02/09/static-vs-dynamic-code-analysis.aspx*

What are the real benefits of static code analysis. *(s.d.). Récupéré sur Programmers Stack Exchange: http://programmers.stackexchange.com/questions/27682/what-are-the-real-benefits-of-static-code-analysis*