Veille technologique

Qualité et test du code,

Test Driven Development

SOMMAIRE

INTRODUCTION

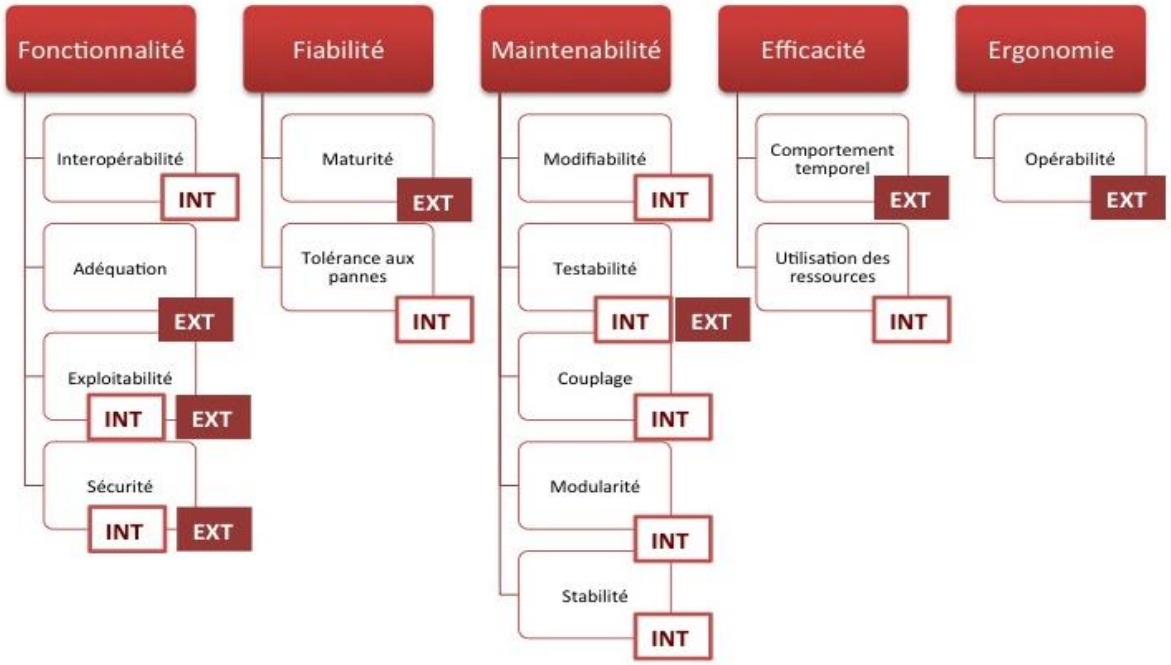
Des environnements de développement et d'exploitation de plus en plus hétérogènes ; des technologies et des langages issus de différents fournisseurs qui s'entremêlent ; des développements et une maintenance externalisés… Voilà de quoi rendre les relations entre donneurs d'ordre et fournisseurs de plus en plus complexes pour les sociétés qui conçoivent des logiciels ou externalisent leur développement. Elles ont en effet besoin d'un bon niveau de service incluant des mesures sur la fiabilité, la capacité de montée en charge, la sécurité et d'autres aspects de la qualité logicielle. De fait, les bénéfices et le retour sur investissement de la qualité logicielle ont été démontrés *(lire 01 Informatique n° 1978, p. 28)*. Un défaut coûte 100 fois moins cher à corriger lorsqu'il est repéré lors des spécifications que s'il est détecté en production. Sans compter le déficit d'image qu'il engendre pour la société. Afin d'améliorer la qualité, il existe des méthodes, des référentiels, et des outils. Outre les outils de test, une démarche d'amélioration de la qualité s'appuie sur des outils de mesure. C'est ici qu'intervient le Cisq, qui vient combler un besoin de standardisation au niveau de la mesure de la qualité logicielle.

QU'EST-CE QUE LE CISQ ?

Le SEI (Software Engineering Institute) et l'OMG (Object Management Group) ont annoncé un partenariat pour créer le Cisq (Consortium of IT Software Quality). Principal objectif de ce consortium : développer un standard pour les métriques de la qualité logicielle.

QUALITE LOGICIELLE | DEFINITION

La notion de qualité d'un logiciel est très large et assez vague. C'est pourquoi l'ISO (Organisation Internationale de Normalisation) en collaboration avec le SEI (Software Engineering Institute) a défini la norme ISO/9126 afin de structurer cette notion. Cette norme caractérise la qualité d'un logiciel par un ensemble d'attributs résumé sur le schéma suivant :



EXT signifie une qualité externe, c'est-à-dire visible de l'utilisateur final et déduite du comportement à l'exécution du logiciel.

INT une qualité interne, c'est-à-dire non visible et déduite uniquement des caractéristiques du logiciel.

Explication de ces cinq grands groupes d'indicateurs :

La fonctionnalité est la capacité qu'ont les fonctionnalités d'un logiciel à répondre aux exigences et besoins explicites ou implicites des usagers. Désigne ce que fait le logiciel pour remplir les besoins utilisateurs. Elle dépend de l'interopérabilité – capacité à interagir avec un ou plusieurs système – de la sécurité, de l'adéquation – présence de fonctions pour effectuer les tâches requises.

La fiabilité, c'est-à-dire la capacité d'un logiciel de rendre des résultats corrects quelles que soient les conditions d'exploitation. En font partie la tolérance aux pannes - la capacité

d'un logiciel de fonctionner même en étant handicapé par la panne d'un composant (logiciel ou matériel).

La maintenabilité, cet attribut désigne la capacité d'un produit logiciel à être modifié. Les modifications peuvent inclure des corrections, des améliorations ou des adaptations du logiciel à des changements dans l'environnement, les besoins et les spécifications fonctionnelles. Désigne l'effort nécessaire pour la modification du logiciel.

L'efficacité, c'est-à-dire le rapport entre la quantité de ressources utilisées (moyens matériels, temps, personnel), et la quantité de résultats délivrés. En font partie le temps de réponse, le débit et l'extensibilité - capacité à maintenir la performance même en cas d'utilisation intensive ;

La portabilité, c'est l'aptitude d'un logiciel de fonctionner dans un environnement matériel ou logiciel différent de son environnement initial. En font partie la facilité d'installation et de configuration dans le nouvel environnement.

Revenons à la fonctionnalité dit aussi l'utilisabilité, c'est un des points le plus important pour la qualité logicielle du point de vue du client.

La fonction	Utile et répond à un besoin.	Facile à utiliser et doit faciliter l'accomplissement de la tâche de l'utilisateur.	Utilisation satisfaisante pour l'utilisateur.
Le client	Permet à l'utilisateur de réaliser les tâches qu'il souhaite accomplir. L'utilisateur est efficace.	Permet à l'utilisateur de réaliser la tâche rapidement et sans erreurs. L'utilisateur est efficient.	Prend plaisir à l'utiliser.

LES BONNES PRATIQUES

Pour définir la qualité d'une fonction il faut se poser deux questions principales :

Comment accéder aux fonctions ?

- **Bonne visibilité des fonctions pour l'utilisateur :**
    - Emplacement sur l'écran : l'emplacement doit suivre le parcours naturel de la page
    - Nature (bouton, lien…)
    - Format (couleur, taille…)
    - et surtout la **terminologie** : chaque fonction est nommée en terme de tâche utilisateur et d'effets avec notamment un nommage **« verbe + complément »** afin de clarifier l'intention de l'action invoquée.
- **Hiérarchiser l'accès aux fonctions** : les plus visibles sont les plus fréquentes ou les premières effectuées
- **Cacher les fonctions avancées destinées aux utilisateurs experts** : tout ne peut pas être visible tout le temps

La fonction est elle facile à utiliser ?

- **Minimiser** le nombre d'actions à réaliser pour atteindre l'objectif (nombre de clics, nombre d'écrans, ...)
- **Le « feedback » est essentiel pour toutes les actions réalisées.** Quels retours système permettent de vérifier que les actions de l'utilisateur se sont bien réalisées ?
- **Optimiser la prise d'informations et de décision** en présentant des informations précises et brèves
- **Guider l'utilisateur** par des messages explicites (erreurs) et implicites (affichage, couleurs, ...)
- **Homogénéiser l'interface** : couleurs, format, emplacement des éléments,...
- **Maximiser la lisibilité** (bloc, données) et la terminologie employée (le nommage des commandes reflète leurs effets)

- **Faciliter le travail de l'utilisateur** (pré-saisie de certains champs, valeurs par défaut, auto-complétion)

- **Réversibilité** : toute décision de l'utilisateur doit être réversible, les erreurs doivent toujours être localisables, compréhensibles et corrigeables. Cela facilite la satisfaction de l'utilisateur.

Une des bonnes pratiques est l'utilisation de tests unitaires, ce sujet sera traité dans la partie des Test Driven Developement.

LES NORMES ISO

Toute mesure que l'on peut calculer à partir du code est un axe potentiel pour le Cisq. Ces mesures seront développées en utilisant les normes OMG de définition et de représentation des mesures logicielles. Le Cisq travaille également à coordonner les efforts avec les nouvelles normes ISO 25000, qui remplaceront la norme ISO 9126.

Les mesures de taille, telles que le nombre de lignes de code ou des métriques fonctionnelles.

Les mesures de qualité, comme la facilité de maintenance, la robustesse, la sécurité, la complexité ou encore, la performance du code.

Les mesures de défauts, d'erreurs et d'autres événements indésirables.

Les définitions d'anti-patterns architecturaux ou de mauvaises pratiques de codage qui peuvent être détectés dans le code.

Les mesures concernant les problèmes de qualité liés aux coûts informatiques.

La norme ISO 25000 :

« Cette norme offre un cadre pour l'intégration des évolutions des normes ISO 9126, la qualité logiciel, et ISO 14598, l'évaluation de la qualité logicielle.

L'architecture SquaRE préconise les 4 étapes suivantes :

1. *Fixer les exigences de qualité,*
2. *Etablir un modèle de qualité,*
3. *Fixer les métriques de la qualité*
4. *Conduire les évaluations. »*

Dans ce domaine, il n'existe aucune norme, aucun référentiel pour comparer et mettre en perspective les outils des différents éditeurs. On trouve bien les normes ISO telles que l'ISO/IEC TR 9126, qui décrit les caractéristiques internes et externes de la qualité et normalise son vocabulaire. Mais elle ne définit pas les mesures de qualité de manière suffisamment précise pour qu'elles soient calculées à partir du code. Il y a également des normes qui s'intéressent aux processus de la qualité (ISO/IEC TR 15504) ou au cycle de vie du test logiciel (ISO/IEC 29119). De son côté, le référentiel CMMi (Capability Maturity Model Integration) s'attache à la qualité des processus de management et de développement applicatif. Enfin, des organismes comme le CFTL (Comité français du test logiciel) et l'ISTQB (International Software Testing Qualification Board) proposent des recueils de bonnes pratiques, par exemple pour la constitution du plan de tests. Mais finalement, il s'avère impossible de comparer les mesures pour réaliser des bancs d'essai ou du contrôle qualité par des applications tierces.

Créé conjointement par le SEI et l'OMG, le Cisq sera constitué de sociétés issues du Global 2000 (classement des 2 000 plus grands industriels de l'informatique), ainsi que d'intégrateurs systèmes, SSII et éditeurs de logiciels. Ils travailleront ensemble pour élaborer un standard de mesure de la qualité logicielle et promouvoir un écosystème capable de supporter son déploiement. Le consortium devra mettre au point ce standard de façon suffisamment détaillée pour permettre une mesure automatisée de la qualité. Le Cisq aura quatre grands objectifs. Il devra **conseiller les entreprises et les chefs de gouvernement** en les alertant sur l'importance critique et stratégique de la qualité des applications informatiques. A lui de concevoir des attributs de mesures standardisées destinés à l'informatique et aux métiers pour évaluer la qualité logicielle et les risques des applications multi-tiers. Le consortium **proposera des méthodes pour utiliser les mesures de qualité** lorsqu'une entreprise s'apprête à négocier et gérer l'acquisition ou la maintenance de logiciels. Il aura également pour mission de **développer et de promouvoir un modèle de licence** pour les professionnels fournissant des services afin de mesurer la qualité de l'application informatique. Enfin, le Cisq mettra en place un **forum** en ligne destiné à l'industrie informatique de façon à répondre aux enjeux en matière de qualité des applications.

LES NORMES DE CONDAGE ET DE NOMMAGE

Le but de ces normes est apparut suite à un constat simple :

- 80% du temps passé dans le cycle de vie d'un logiciel intervient pendant le cycle de maintenance
- Il est très rare qu'un code soit maintenu par son auteur original
- De bonnes conventions adoptées par tous permettent à un développeur de s'y retrouver plus facilement dans un code inconnu

Quelques exemples de ces normes

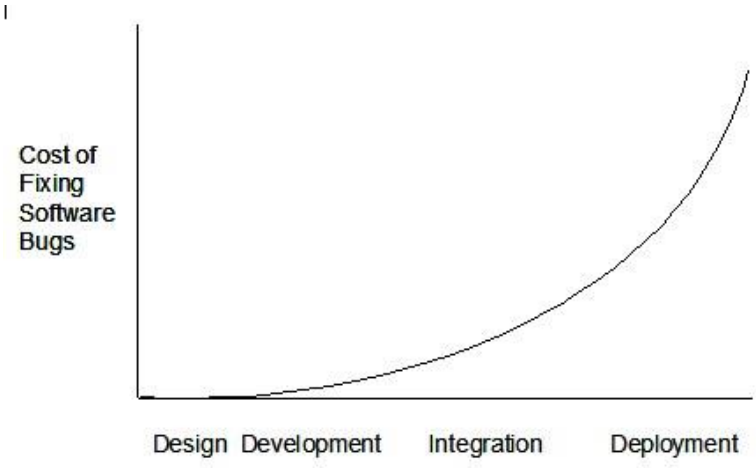
- Normes générales d'organisation du code :
  - La taille d'un fichier ne doit pas excéder 1000 lignes
  - La longueur d'une méthode doit faire au plus 100 lignes
  - L'entête documentaire des classes et méthode permettent de spécifier le comportement de l'applicatif
  - Environ 20% du code est du commentaire

- Normes générales de codage
  - Les packages : tout en minuscule, utiliser [a-z][0-9] et le point, utiliser un root comme (com, gov, mil, net…)
  - Les classes : première lettre en majuscule, utiliser [a-z][A-Z] et [0-9], première lettre de chaque mot en majuscule, nom simple et descriptif
  - Les variables : comme pour les classes sauf première lettre en minuscule
  - Les constantes : tout en majuscules, séparer les mots par des "_", utiliser [a-z][A-Z] et [0-9], nom simple et descriptif

- Conventions relatives à la lisibilité
  - Indentation du code
  - Taille des lignes : 180 caractères maximum

Ces normes sont présentes pour que le code soit interprétables plus facilement par les développeurs mais il est parfois nécessaire de sortir de ces normes.

Les projets étant de plus en plus complexes et les équipes de développement réparties partout dans le monde, il devient de plus en plus important d'en suivre la qualité afin de corriger les problèmes le plus tôt possible. Car plus un bug est découvert tard dans le cycle de réalisation du logiciel, plus sa correction est coûteuse.



Une politique de qualité doit être mise en place et comporter au moins ces 4 étapes :

- Définition d'un objectif
- Choix d'une méthode
- Outillage de cette méthode
- Mise en place d'un processus de suivi de qualité

Notre objectif sera de valider la définition de la qualité suivante.

On peut définir la qualité d'un code source par ces critères :

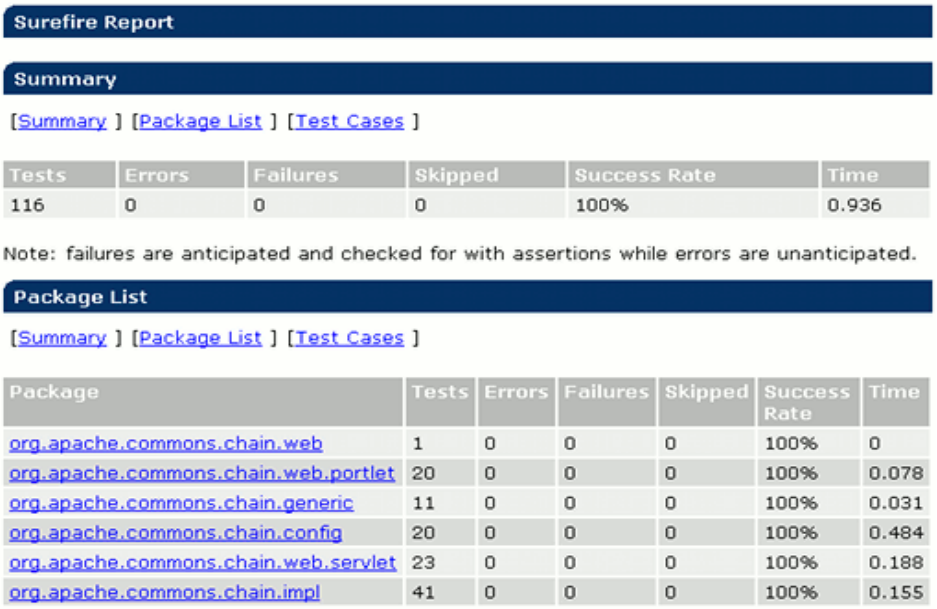
- Un code documenté
- Un code respectant les standards (de son entreprise, de son projet, ...)
- Un code sans bug
- Un code maintenable, évolutif et simple
- Un code performant
- Un code testé

Maven2 est un outil qui permet grâce à la définition du projet dans le fichier pom.xml de gérer tout le cycle de vie du projet (compilation, test, packaging, installation...).

Grâce à plusieurs plugins, l'outil nous indiquera divers point de qualité du projet.

Plugin 1 : Surefire

Ce plugin permet d'avoir les rapports d'exécution de tests unitaires.



Surefire Report

Summary

[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
116	0	0	0	100%	0.936

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

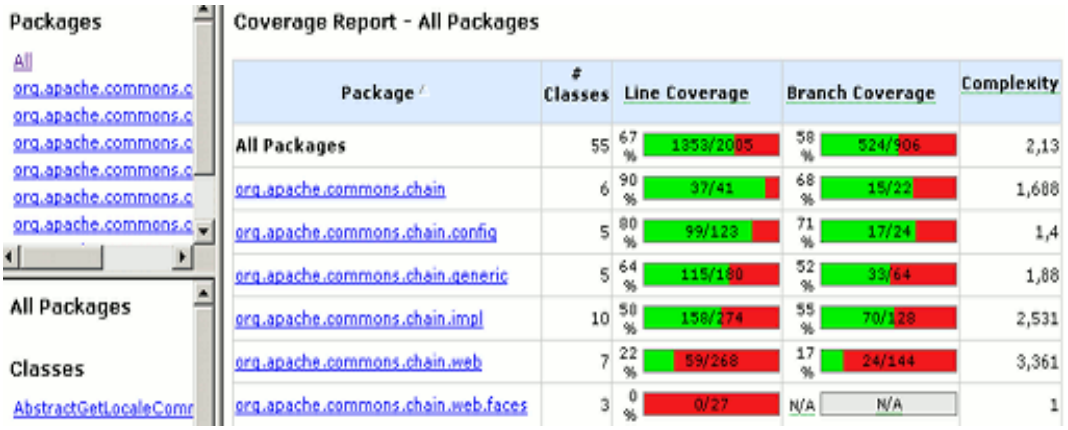
Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
org.apache.commons.chain.web	1	0	0	0	100%	0
org.apache.commons.chain.web.portlet	20	0	0	0	100%	0.078
org.apache.commons.chain.generic	11	0	0	0	100%	0.031
org.apache.commons.chain.config	20	0	0	0	100%	0.484
org.apache.commons.chain.web.servlet	23	0	0	0	100%	0.188
org.apache.commons.chain.impl	41	0	0	0	100%	0.155

Plugin 2 : Cobertura

Cobertura permet de connaître la couverture du code par les tests unitaires. Cette information est importante car, même avec un nombre élevé de test unitaire, on n'a pas forcement une bonne couverture et certaines parties importantes peuvent se retrouver peu ou pas testées.



Packages

All

org.apache.commons.c

org.apache.commons.c

org.apache.commons.c

org.apache.commons.c

org.apache.commons.c

org.apache.commons.c

All Packages

Classes

AbstractGetLocaleComr

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	55	67 % 1353/2005	58 % 524/906	2,13
org.apache.commons.chain	6	90 % 37/41	68 % 15/22	1,688
org.apache.commons.chain.config	5	80 % 99/123	71 % 17/24	1,4
org.apache.commons.chain.generic	5	64 % 115/180	52 % 33/64	1,88
org.apache.commons.chain.impl	10	58 % 158/274	55 % 70/128	2,531
org.apache.commons.chain.web	7	22 % 59/268	17 % 24/144	3,361
org.apache.commons.chain.web.faces	3	0 % 0/27	N/A N/A	1

Plugin 3 : PMD/CPD

PMD va analyser le code afin de trouver des problèmes potentiels tel que :

- Bugs possibles,
- Code mort,
- Code non optimal,
- Expressions trop compliquées,
- Problèmes de sécurité,
- Problèmes de couplage entre objet/package,

Plugin 4 : CheckStyle

CheckStyle permet de contrôler le respect des conventions de codage et d'avoir quelques métriques. Ce contrôle est fait avec 126 règles.

Il existe encore bien d'autre plugin que je ne citerais pas ici pour des raisons de redondance.

TEST DRIVEN DEVELOPMENT (TDD)

Ce terme désigne une technique de développement qui entremêle la programmation, l'écriture de tests unitaires et l'activité de remaniement. Elle propose les règles suivantes:

- Créer **un seul** test unitaire décrivant un aspect du programme
- S'assurer, en l'exécutant, que ce test échoue pour les bonnes raisons
- Ecrire **juste assez** de code, le plus simple possible, pour que ce test passe
- **Remanier** le code autant que nécessaire pour se conformer aux critères de simplicité
- Recommencer, en **accumulant** les tests au fur et à mesure

Lors de l'exécution des tests une "barre verte" signifie que l'ensemble des tests unitaires accumulés passent avec succès, "barre rouge" signifie qu'au moins un test est en échec.

Les bénéfices d'une telle pratique :

- De façon informelle, de nombreux retours d'expérience d'équipes utilisant TDD font état d'une réduction très significative du nombre de défauts, en contrepartie d'un surcoût modéré du développement *initial*

- Ces mêmes retours suggèrent que ce surcoût est compensé par l'élimination d'une partie importante de l'effort de mise au point en fin de projet

- Bien que les travaux scientifiques à ce sujet restent circonspects, de nombreux vétérans de cette pratique y voient un facteur d'amélioration de la conception objet de leur code, et plus généralement de la qualité technique: on entend ainsi que la pratique du TDD a pour résultat du code possédant une meilleure cohésion et un plus faible couplage