

LOG8371 : Ingénierie de la qualité en logiciel

Qualité du code - Maintenabilité Hiver 2017

Fabio Petrillo
Chargé de Cours



This work is licensed under a Creative
Commons Attribution-NonCommercial-
ShareAlike 3.0 Unported License

Qui a écrit ce morceau de code?

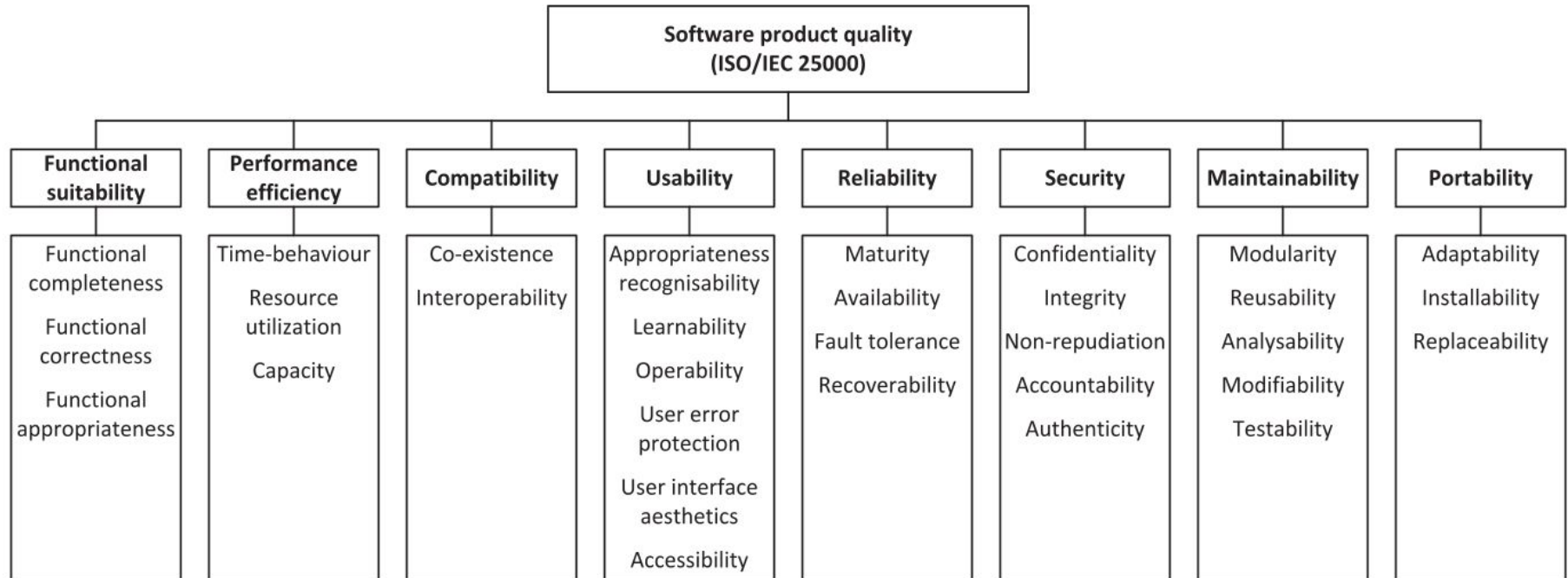
Je ne peux pas travailler comme ça !!

Qui a écrit ce morceau de code?

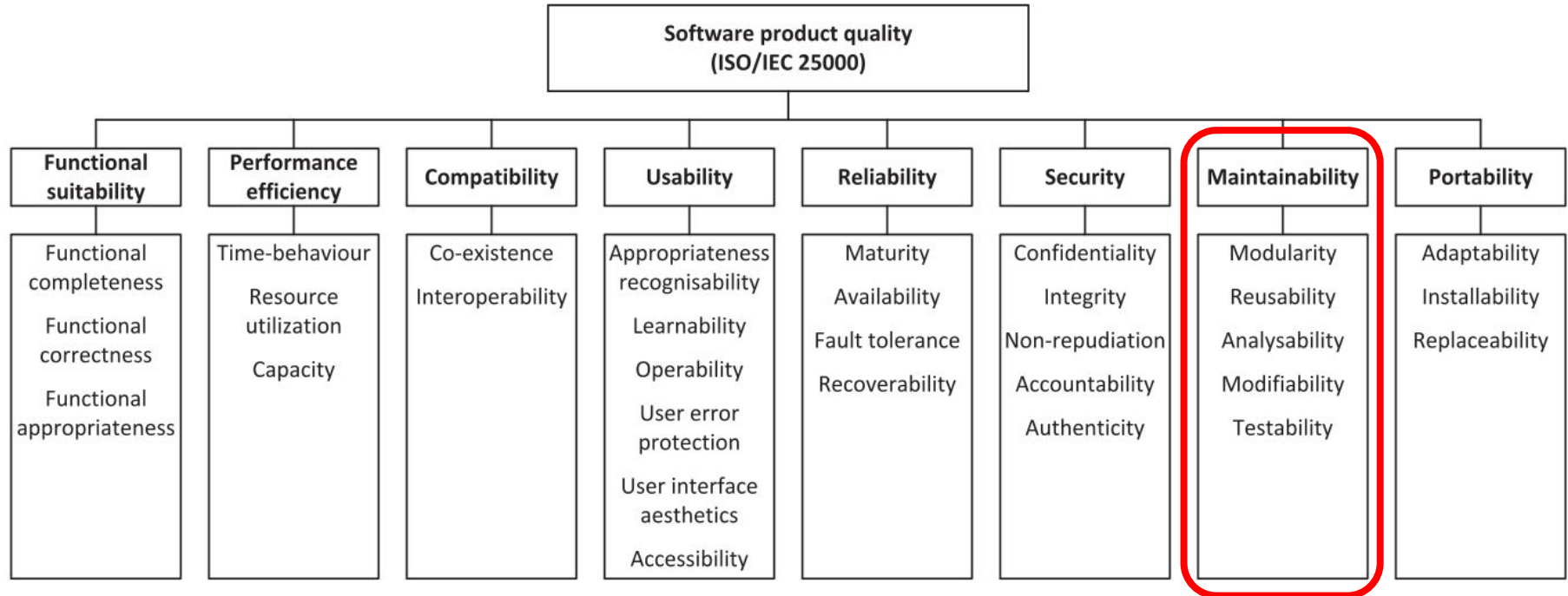
Je ne peux pas travailler comme ça !!

Après quelques analyses, vous réalisez
qui a écrit que le code était **vous-même !!!**

ISO/IEC 25010 - Modèle de qualité pour les produits logiciels



ISO/IEC 25010 - Modèle de qualité pour les produits logiciels



**La maintenabilité est l'aspect de qualité
le plus important, parce que nous ne
pouvons pas atteindre les autres
aspects sans être capable de maintenir
notre code facilement!!!**

La maintenabilité [ISO 25010, §4.2.7]

*“Le degré d'**efficacité** et de **souplesse** avec lequel un produit ou un système peut être **modifié** par les responsables, Où les modifications peuvent inclure des corrections, des améliorations ou des adaptations au logiciel pour des changements dans l'environnement ainsi que dans les exigences et spécifications fonctionnelles. Elle comprend aussi l'installation de mises à jour et de mises à niveau. Elle peut être interprété comme une **capacité inhérente du produit ou du système afin de faciliter les activités de maintenance**, ou la qualité d'utilisation dont disposent les responsables pour maintenir le produit ou le système.”*

L'analyse[ISO 25010, §4.2.7.3]

“Le degré d'efficacité et de souplesse avec lequel il est possible d'évaluer l'impact d'un changement envisagé sur un produit ou un système dans une ou plusieurs de ses parties, ou pour diagnostiquer les défaillances ou les causes d'erreurs, ou pour identifier les parties à modifier”

La modifiabilité [ISO 25010, §4.2.7.4]

*“Le degré auquel un produit ou un système peuvent être efficacement et souplement modifiés **sans introduire de défauts** ou dégrader la qualité du produit”*

La testabilité [ISO 25010, §4.2.7.5]

“Le degré d'efficacité et de souplesse avec lequel **des critères de test peuvent être établis** pour un système, un produit ou un composant ainsi que des tests peuvent être **effectué** pour déterminer si ces critères ont été respectés.”

La modularité [ISO 25010, §4.2.7.1]

“Le degré auquel un système ou un programme informatique est composé de **composants discrets**, de sorte qu'un **changement sur un composant aura un impact minimal** sur d'autres.”

La réutilisabilité [ISO 25010, §4.2.7.2]

“Le degré auquel **un module** logiciel peut être utilisé dans **plus d'un système**, ou dans la **construction d'autres modules**.”

Comment déterminer les
caractéristiques de qualité
de la maintenabilité?

Comment déterminer les caractéristiques de qualité de la maintenabilité?

Mesurer un ensemble de **propriétés** d'un produit logiciel.

Les propriétés d'un produit logiciel

- **Volume: Taille** globale du code source du produit logiciel. La taille est déterminée à partir du **nombre de lignes** de code par langage de programmation.
- **Duplication**: Elle concerne l'apparition d'un **fragments de code source identiques** en **plus d'un endroit** du produit.
- **Complexité de l'unité**: Le degré de **complexité** dans les **unités** du code source. La notion d'unité correspond aux plus petites parties exécutables du code source, telles que des **méthodes ou des fonctions**.
- **Accouplement de module**: Le couplage entre les modules en fonction du nombre de **dépendances entrantes** pour les modules du code source. La notion de module correspond à un **groupement** d'unités liées.

Les Propriétés du produit logiciel (cont.)

- **Component balance:** it is the product of the **system breakdown**, which is a rating for the number of top-level components in the system, and the component size uniformity, which is a rating for the size distribution of those top-level components. The notion of top-level components corresponds to the first subdivision of the source code modules of a system into components, where a component is a grouping of source code modules.
- **Component independence:** Component independence is a rating for the percentage of code in modules that have **no incoming dependencies** from modules in other top-level components.

Les Propriétés du produit logiciel (cont.)

- **Balance des composants:** il est le produit de la **rupture du système**, qui est une estimation pour le nombre de composants du niveau supérieur dans le système, et l'uniformité de leurs taille, qui est une estimation pour la distribution de taille de ces composants du niveau supérieur. La notion de composants du niveau supérieur correspond à la première subdivision des modules de code source d'un système en composants, où un composant est un regroupement de modules de code source.
- **Indépendance des composants:** L'indépendance des composants est une estimation du pourcentage de code dans les modules qui n'ont aucune dépendance entrante à partir des modules des autres composants du niveau supérieur.

A Practical Model for Measuring Maintainability

– a preliminary report –

Ilja Heitlager
Software Improvement Group
The Netherlands
Email: i.heitlager@sig.nl

Tobias Kuipers
Software Improvement Group
The Netherlands
Email: t.kuipers@sig.nl

Joost Visser
Software Improvement Group
The Netherlands
Email: j.visser@sig.nl

Abstract—The amount of effort needed to maintain a software system is related to the technical quality of the source code of that system. The ISO 9126 model for software product quality recognizes maintainability as one of the 6 main characteristics of software product quality, with adaptability, changeability, stability, and testability as subcharacteristics of maintainability.

Remarkably, ISO 9126 does not provide a consensual set of measures for estimating maintainability on the basis of a system's source code. On the other hand, the Maintainability Index has been proposed to calculate a single number that expresses the maintainability of a system.

In this paper, we discuss several problems with the MI, and we identify a number of requirements to be fulfilled by a maintainability model to be usable in practice. We sketch a new maintainability model that alleviates most of these problems, and we discuss our experiences with using such a system for IT management consultancy activities.

Many software metrics have been proposed as indicators for software product quality [4], [5]. In particular, Oman *et al.* proposed the Maintainability Index (MI) [6], [7]: an attempt to objectively determine the maintainability of software systems based upon the status of the source code. The MI is based on measurements the authors performed on a number of systems and calibrating these results with the opinions of the engineers that maintained the systems. The results for the systems examined by Oman *et al.* were plotted, and a fitting function was derived. The resulting fitting function was then promoted to be the Maintainability Index producing function. Subsequently, a small number of improvements were made to the function.

We have used the Maintainability Index in our consultancy practice [8] over the last four years alongside a large

Standardized code quality benchmarking for improving software maintainability

Robert Baggen · José Pedro Correia · Katrin Schill · Joost Visser

Published online: 18 May 2011
© Springer Science+Business Media, LLC 2011

<https://link.springer.com/article/10.1007%2Fs11219-011-9144-9>

SIG/TÜViT
Evaluation Criteria
Trusted Product
Maintainability

Version 8.0

Relation des sous-caractéristiques et des propriétés d'un système

	Volume	Duplication	Unit size	Unit complexity	Unit interfacing	Module coupling	Component balance	Component independence
Analyzability	X	X	X				X	
Modifiability		X		X		X		
Testability	X			X				X
Modularity						X	X	X
Reusability			X		X			

Alors, comment construire des **logiciels
maintenables?**

O'REILLY®

Java Edition



Building Maintainable Software

TEN GUIDELINES FOR FUTURE-PROOF CODE

Joost Visser

Une proposition

10 guides pour vous aider à écrire du code source qui est facile à modifier.

“Après 15 ans de consultation sur la qualité des logiciels, comme des membres du Groupe d'amélioration des logiciels (SIG), Nous avons appris une ou deux choses sur la maintenabilité.”

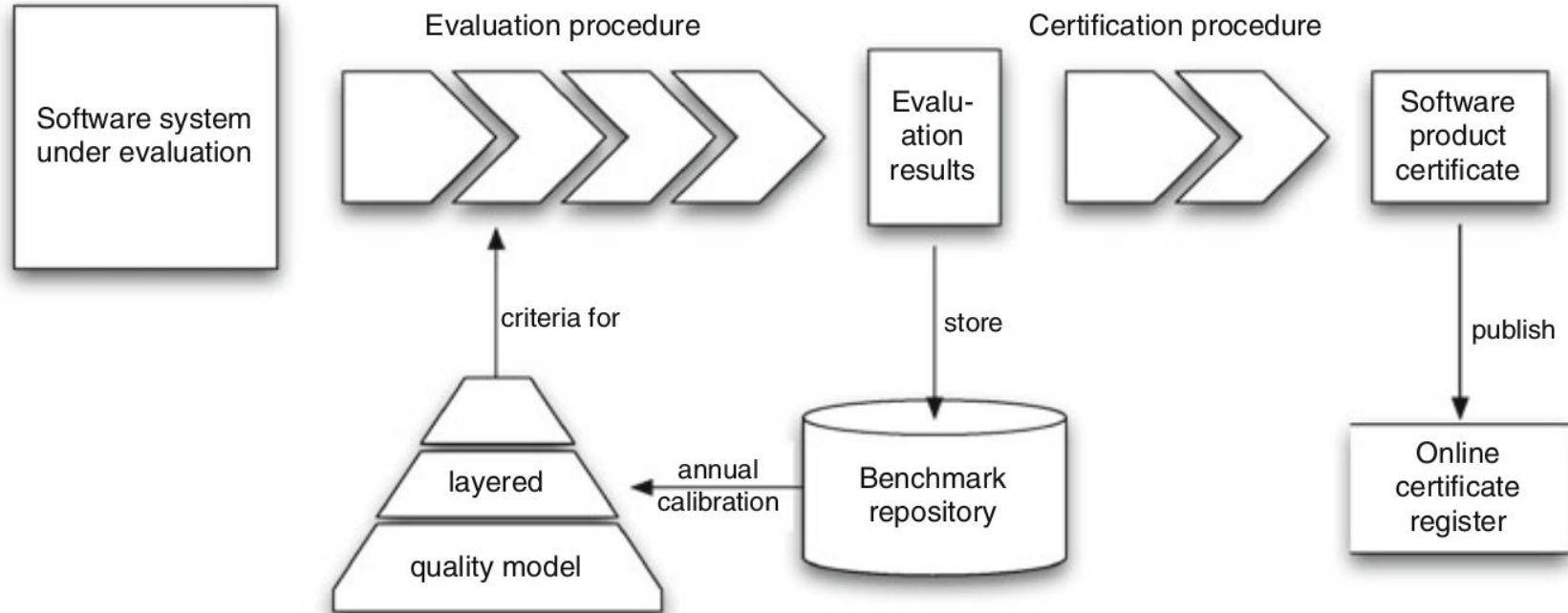
Les 10 principaux guides du groupe SIG

- Les avantages de maintenabilité est de respecter les **simples directives**.
- La maintenabilité n'est pas une réflexion secondaire, mais elle doit être abordée **dès le début** d'un projet de développement. **Chaque contribution individuelle compte**.
- Certaines violations sont **pires** que d'autres. Plus un système logiciel est conforme aux directives, plus il est facile à entretenir.

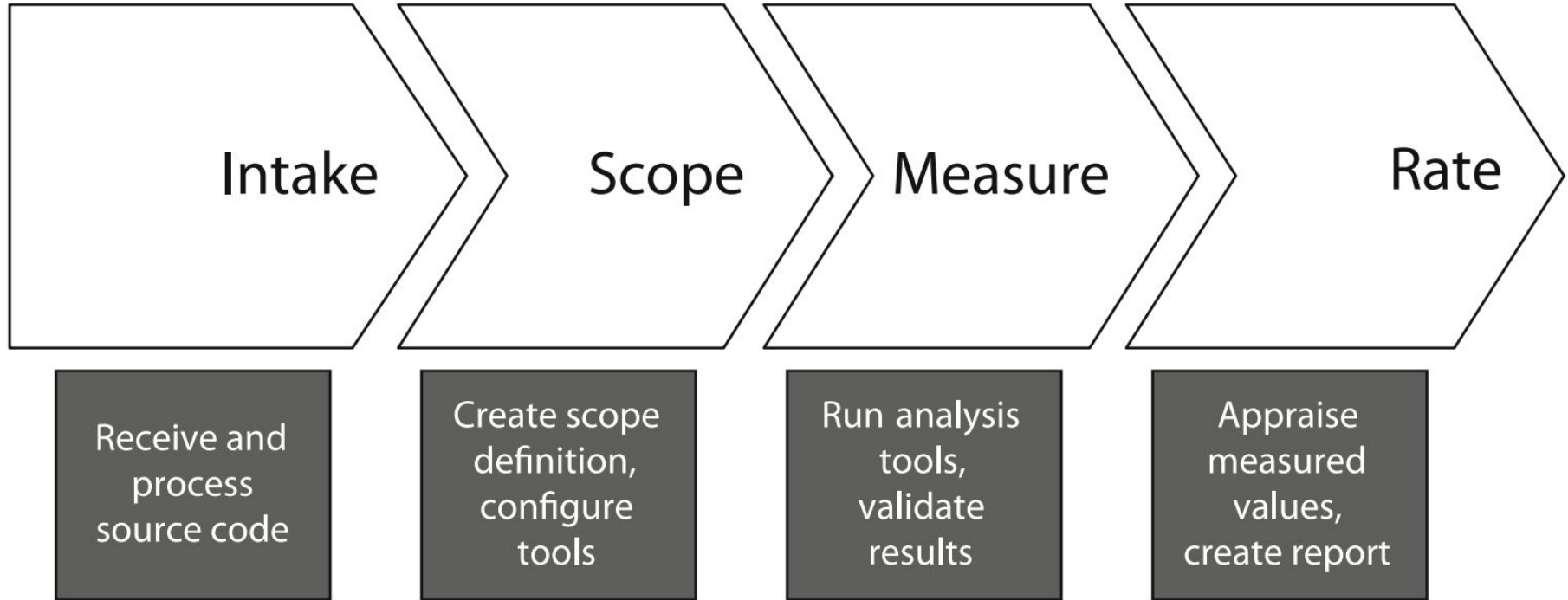
Capacité de maintenabilité

- La maintenabilité est une caractéristique de qualité mesurable
- Différents degrés pour pouvoir maintenir un système
- SIG divise les systèmes dans le Benchmark par des **étoiles d'évaluation**, Allant de **1 étoile (difficile)** à entretenir) à 5 étoiles (**facile** à entretenir).
- Benchmark - Résultats de plusieurs centaines d'évaluations de systèmes standard
- Distribution -> 1 à 5 étoiles est 5%-30%-30%-30%-5%
- Des preuves empiriques montrant que la **résolution des problèmes** et les améliorations sont **deux fois plus rapides** dans les systèmes à **4 étoiles** que dans les systèmes à **2 étoiles**.

Benchmarking de la qualité des produits logiciels



Procédure d'évaluation du code source

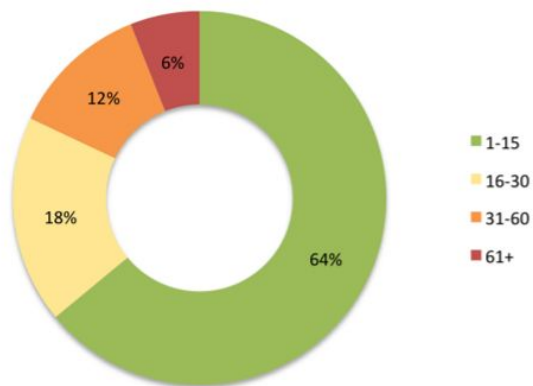


Taux de maintenabilité SIG

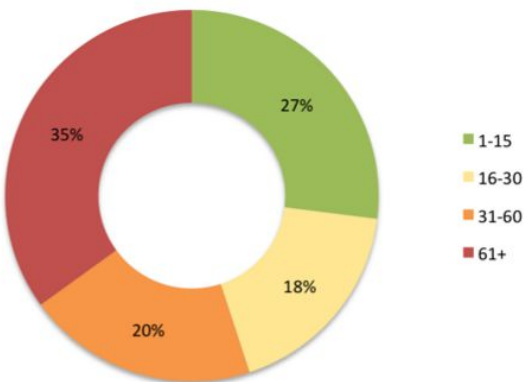
Rating	Maintainability
5 stars	Top 5% of the systems in the benchmark
4 stars	Next 30% of the systems in the benchmark (above-average systems)
3 stars	Next 30% of the systems in the benchmark (average systems)
2 stars	Next 30% of the systems in the benchmark (below-average systems)
1 star	Bottom 5% least maintainable systems

Capacité d'entretien

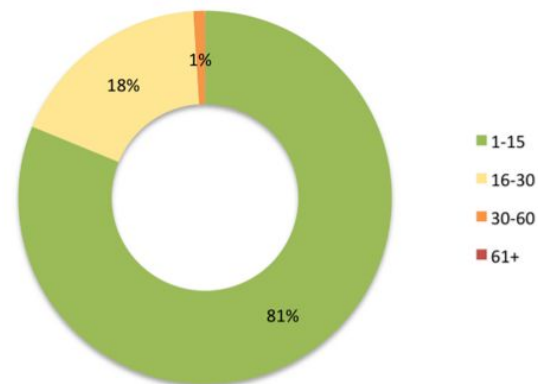
Unit size quality profile of Jenkins



Unit size quality profile of an anonymous ★★☆☆☆ system



Unit size quality profile of an anonymous ★★★★★ system



Un regroupement générique de concepts en Java

Generic name	Generic definition	In Java
Unit	Smallest grouping of lines that can be executed independently	Method or constructor
Module	Smallest grouping of units	Top-level class, interface, or enum
Component	Top-level division of a system as defined by its software architecture	(Not defined by the language)
System	The entire codebase under study	(Not defined by the language)

Les 10 consignes de SIG

- **Écrivez de courtes unités de code:** les courtes unités (c'est-à-dire les méthodes et les constructeurs) sont plus faciles à analyser, à tester et à réutiliser.
- **Écrire des unités de code simples:** Les unités avec un petit nombre de points de décision sont plus faciles à analyser et à tester.
- **Écrire le code une fois:** la duplication du code source doit être toujours évitée, étant donné que des modifications devront être apportées dans chaque copie. La duplication est également une source de régression de bogues.
- **Gardez de petites interfaces d'unité:** les unités (méthodes et constructeurs) avec moins de paramètres sont plus faciles à tester et à réutiliser.
- **Séparez les préoccupations en modules:** les modules (classes) qui sont lâchement couplés sont plus faciles à modifier et conduisent à un système plus modulaire.

Les 10 consignes de SIG (cont.)

- **Composants de l'architecture faiblement couplés:** les composants de niveau supérieur d'un système qui sont plus faiblement couplés sont plus faciles à être modifiés et conduisent à un système plus modulaire.
- **Maintenir les composants de l'architecture équilibrée:** Une architecture bien équilibrée, qui n'a ni beaucoup ni trop peu de composants, de taille uniforme, elle est la plus modulaire et qui permet une modification facile par la séparation des préoccupations.
- **Gardez votre codebase petit:** un grand système est difficile à maintenir, car plus de code doit être analysé, modifié et testé. En outre, la productivité de la maintenance par ligne de code est plus faible dans un système de grande taille que dans un petit système.

Les 10 consignes de SIG (cont.)

- **Automatiser les pipelines de développement et des tests:** les tests automatisés (c'est-à-dire les tests qui peuvent être exécutés sans intervention manuelle) permettent une rétroaction quasi instantanée sur l'efficacité des modifications. Les test manuels ne sont pas à l'échelle (scalable).
- **Écrire du code propre:** avoir des artefacts non pertinents tels que TODOs et le code mort dans votre codebase rend plus difficile pour les nouveaux membres de l'équipe à devenir productifs. Par conséquent, il rend la maintenance moins efficace.

Ecrire des unités de code courtes

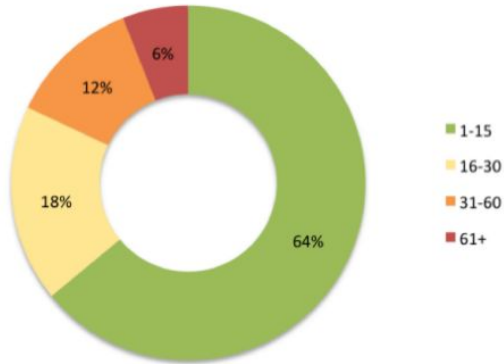
- Limitez la longueur des unités de code (méthodes) à **15 lignes de code**.
- Les petites unités sont
 - facile à comprendre
 - facile à tester
 - facile à réutiliser

Seuils minimal pour un classement de taille d'unité 4-étoiles

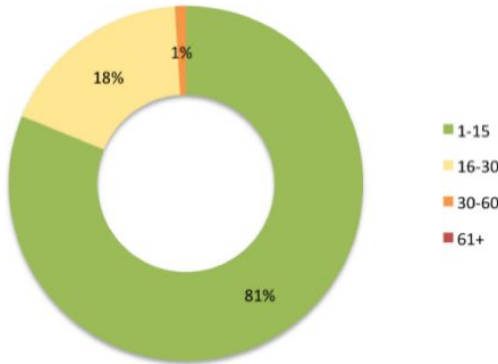
Lines of code in methods with ...	Percentage allowed for 4 stars for unit size
... more than 60 lines of code	At most 6.9%
... more than 30 lines of code	At most 22.3%
... more than 15 lines of code	At most 43.7%
... at most 15 lines of code	At least 56.3%

Trois profils de qualité pour la taille de l'unité

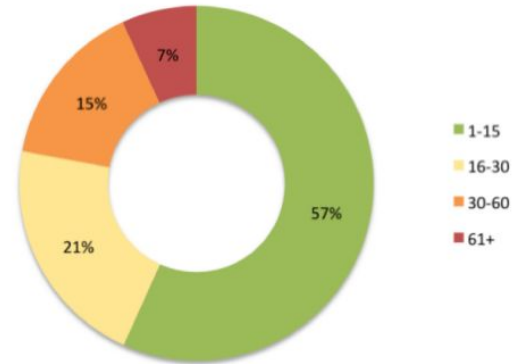
Unit size quality profile of Jenkins



Unit size quality profile of an anonymous ★★★★★ system



Unit size quality profile cut-offs for a ★★★★★ system



Écrire des unités de code simples

- Limiter le nombre de **points de branchement par unité à 4 (ou limiter la complexité de McCabe à 5)**
- Un point de branche est une instruction dans laquelle l'exécution peut prendre plus d'une direction en fonction d'une condition (ex. les instructions *if* et *switch*).
- faire ceci par diviser les unités complexes en plus simples ainsi que éviter les unités complexes ensemble.
- Cela améliore la maintenabilité car le fait de maintenir un petit nombre de points de branche rend les unités plus faciles à modifier et à tester.
- Nous devons **limiter** la complexité!
- Les unités simples facilitent les tests

Sample code

```
statement 1;  
If ( condition 1 ) {  
    statement 2;  
} else {  
    statement 3;  
}  
statement 4;  
for( condition 2 ) {  
    statement 5;  
}  
statement 6;
```

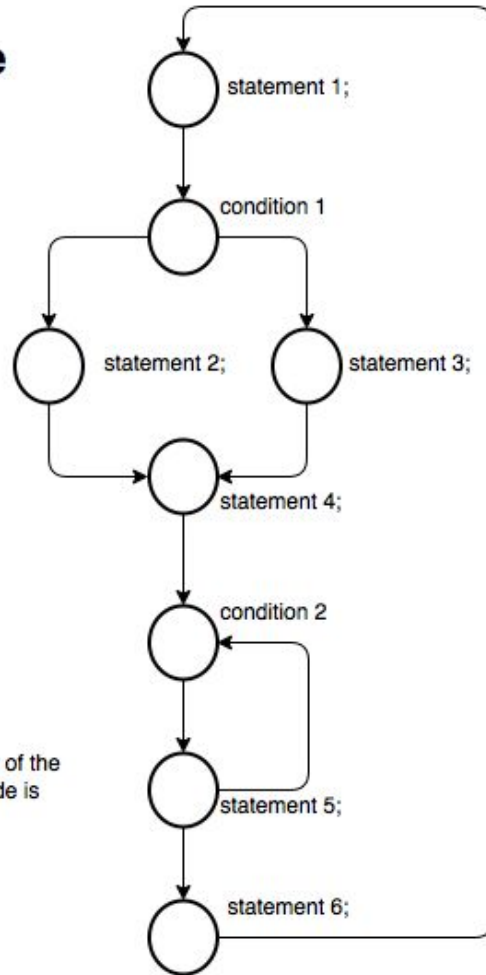
Complexity

The cyclomatic complexity of the graph representing the code is

$$v(G) = E - N + 2$$

$$= 9 - 8 + 2$$

$$= 3$$

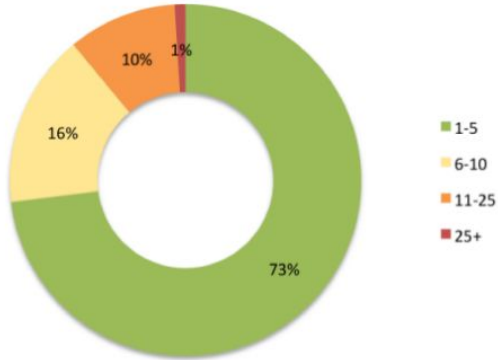


Seuils minimal pour classement de complexité d'unité 4-étoiles

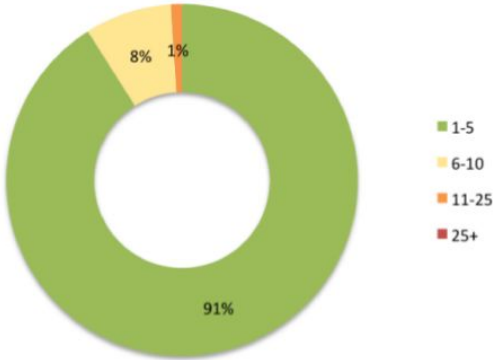
Lines of code in methods with ...	Percentage allowed for 4 stars for unit complexity
... a McCabe above 25	At most 1.5%
... a McCabe above 10	At most 10.0%
... a McCabe above 5	At most 25.2%
... a McCabe of at most 5	At least 74.8%

Trois profils de qualité pour la complexité de l'unité

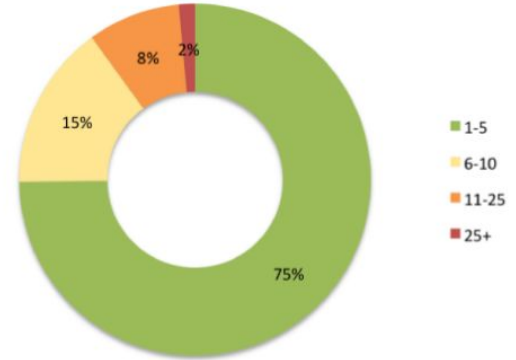
Unit complexity quality profile in Jenkins



Unit complexity quality profile of an anonymous ★★★★★ system



Unit complexity quality profile cut-offs for a ★★★★★ system



Écrire le code une fois

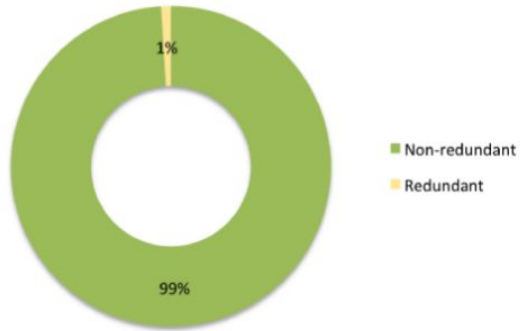
- Ne pas copier le code.
- Faites ceci en écrivant un code réutilisable, générique et/ou en appelant des méthodes existantes.
- Cela améliore la maintenabilité parce que si le code est copié, les bogues doivent être corrigés à plusieurs endroits, ce qui est inefficace et sujet à erreurs.
- Le problème fondamental de la duplication n'est pas de savoir s'il existe une autre copie du code que vous analysez, combien d'exemplaires existent et où ils se trouvent.
- Le code dupliqué contient un bogue, le même bug apparaît plusieurs fois.
- **Ne** réutilisez **jamais** le code en **copiant et collant** des fragments de code existants!

Seuils minimal pour classement de duplication d'unité 4-étoiles

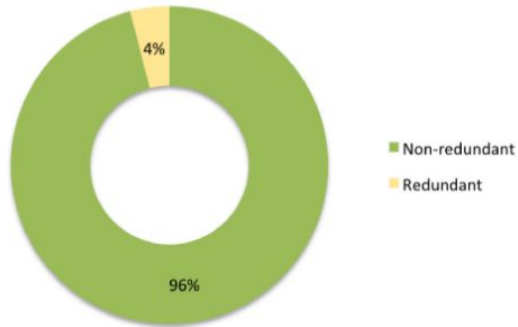
Lines of code categorized as ...	Percentage allowed for 4 stars
... nonredundant	At least 95.4%
... redundant	At most 4.6%

Trois profils de qualité de duplication de code

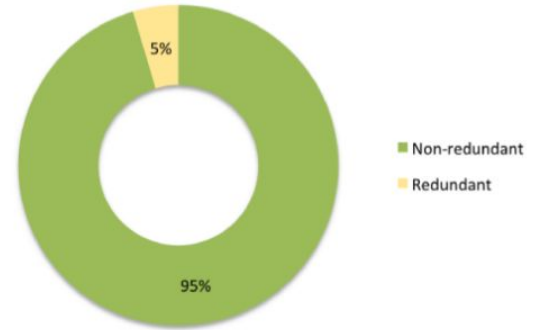
Code duplication quality profile in Jenkins



Code duplication quality profile of an anonymous ★★★★★ system



Code duplication quality profile cut-offs for a ★★★★★ system



Conserver les interfaces d'unité petites

- Limiter le nombre de paramètres au maximum 4 par unité.
- Effectuez cette opération en extrayant des paramètres dans des objets.
- Cela améliore la maintenabilité car le fait de maintenir un nombre faible de paramètres rend les unités plus compréhensible et à réutilisable.

operation //

* *The height of this square (in pixels).*

*/

```
private void render(Square square, Graphics g, int x, int y, int w, int h) {  
    square.getSprite().draw(g, x, y, w, h);  
    for (Unit unit : square.getOccupants()) {  
        unit.getSprite().draw(g, x, y, w, h);  
    }  
}
```

operation

** The height of this square (in pixels).
/

```
private void render(Square square, Graphics g, int x, int y, int w, int h) {  
    square.getSprite().draw(g, x, y, w, h);  
    for (Unit unit : square.getOccupants()) {  
        unit.getSprite().draw(g, x, y, w, h);  
    }  
}
```

operation

** The position and dimension for rendering the square.
/

```
private void render(Square square, Graphics g, Rectangle r) {  
    Point position = r.getPosition();  
    square.getSprite().draw(g, position.x, position.y, r.getWidth(),  
        r.getHeight());  
    for (Unit unit : square.getOccupants()) {  
        unit.getSprite().draw(g, position.x, position.y, r.getWidth(),  
            r.getHeight());  
    }  
}
```

```
* @param r  
* The position and dimension for rendering the square.  
*/
```

```
private void render(Square square, Graphics g, Rectangle r) {  
    Point position = r.getPosition();  
    square.getSprite().draw(g, position.x, position.y, r.getWidth(),  
        r.getHeight());  
    for (Unit unit : square.getOccupants()) {  
        unit.getSprite().draw(g, position.x, position.y, r.getWidth(),  
            r.getHeight());  
    }  
}
```

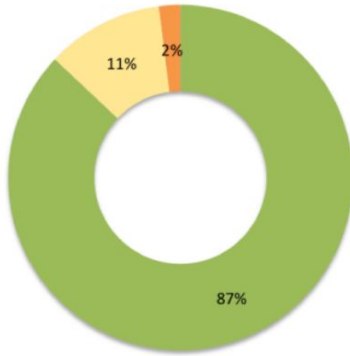
```
private void render(Square square, Graphics g, Rectangle r) {  
    Point position = r.getPosition();  
    square.getSprite().draw(g, r);  
    for (Unit unit : square.getOccupants()) {  
        unit.getSprite().draw(g, r);  
    }  
}
```

Seuils minimal pour un classement d'unité de 4-étoiles

Lines of code in methods with ...	Percentage allowed for 4 stars for unit interfacing
... more than seven parameters	At most 0.7%
... five or more parameters	At most 2.7%
... three or more parameters	At most 13.8%
... at most two parameters	At least 86.2%

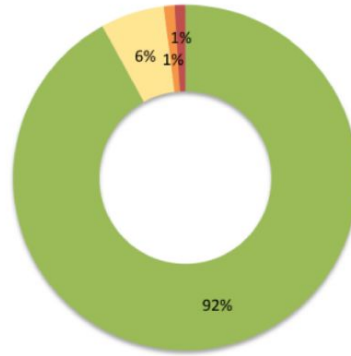
Trois profils de qualité pour l'interface de l'unité

Unit interfacing quality profile for Jenkins



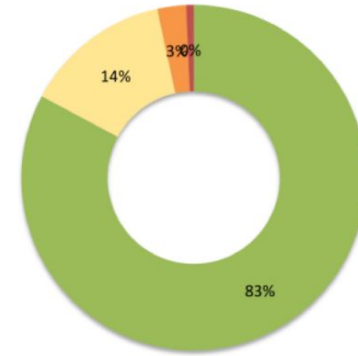
■ 1-2
■ 3-4
■ 5-6
■ 7+

Unit interfacing quality profile of an anonymous ★★★★★ system



■ 1-2
■ 3-4
■ 5-6
■ 7+

Unit interfacing quality profile cut-offs for a ★★★★★ system



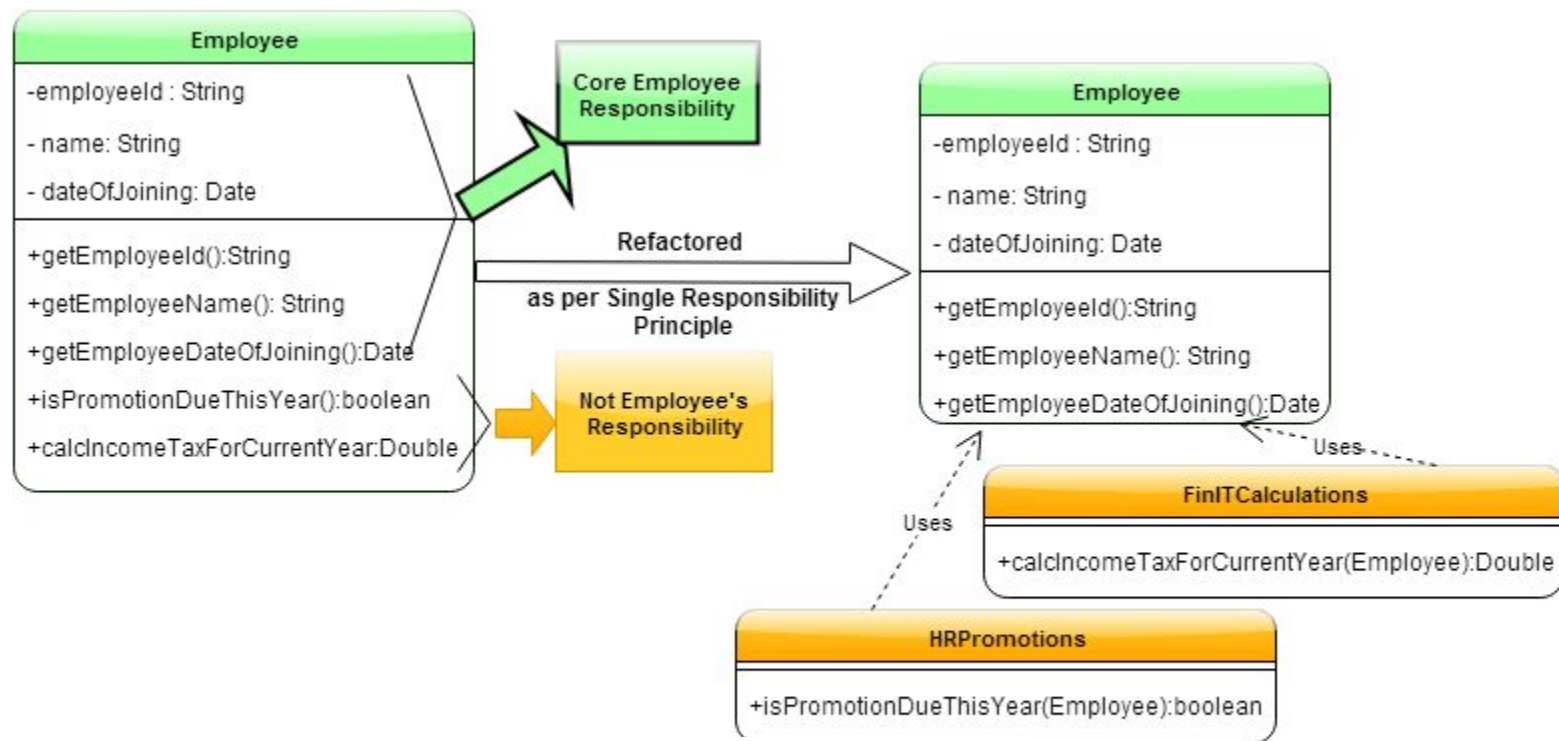
■ 1-2
■ 3-4
■ 5-6
■ 7+

Préoccupations séparées dans les modules

- Évitez les **gros modules** afin d'obtenir un **faible couplage** entre eux.
- Mesurer à l'aide du nombre d'**appels arrivant** d'autres classes (**fan-in**) de toutes les méthodes de la classe.
- Faire ceci par assigner des responsabilités à des **modules séparés** et **masquer les détails d'implémentation** des interfaces.
- Le **couplage** signifie que **deux parties d'un système** sont en quelque sorte connectées quand des changements sont nécessaires. Quand nous changeons **A**, nous devons changer **B**.
- Le problème avec ces classes est qu'ils deviennent un **hotspot de maintenance**.
- Appliquer un principe de responsabilité unique.

Principe de responsabilité unique

- Les principes SOLID de Robert C. Martin
- Chaque module/classe devrait avoir la responsabilité sur une seule partie de la fonctionnalité fournie par un logiciel.
- La responsabilité doit être entièrement encapsulée par la classe.
- Tous ses services devraient être étroitement alignés sur cette responsabilité.
- **Une classe doit avoir une seule raison de changer.**
- Principe de cohésion.



Comment s'inscrire

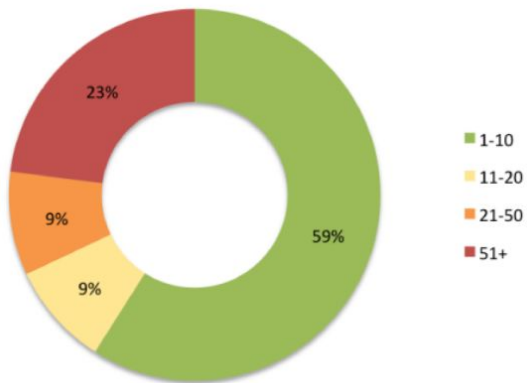
- **Diviser les classes** pour séparer les préoccupations
- **Masquer** les implémentations spécialisées derrière les **interfaces**
- Remplacez le code personnalisé **Third-Party Libraries/Frameworks**

Catégories de risque de couplage de modules

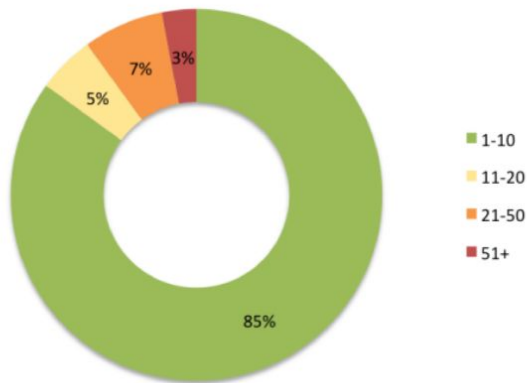
Fan-in of modules in the category	Percentage allowed for 4 stars
51+	At most 6.6%
21–50	At most 13.8%
11–20	At most 21.6%
1–10	No constraint

Trois profils de qualité pour le couplage de modules

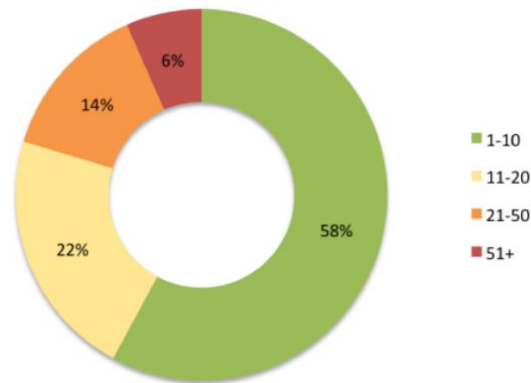
Module coupling quality profile
for Jenkins



Module coupling quality profile of
an anonymous ★★★★★ system



Module coupling quality profile
cut-offs for a ★★★★★ system

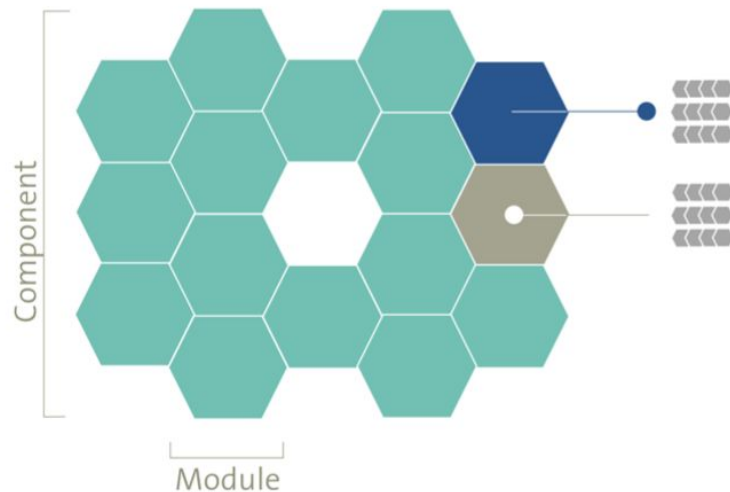


Couple Architecture Components Loosely

- Atteindre un faible couplage entre les composants de niveau supérieur.
- Pour ce faire, réduisez au minimum la quantité relative de code dans les modules exposés (c'est-à-dire pouvant recevoir des appels de) modules dans d'autres composants.
- Cela améliore la maintenabilité car les composants indépendants facilitent la maintenance isolée.
- Les composants doivent être faiblement couplés; C'est-à-dire qu'ils devraient être clairement séparés en ayant peu de points d'entrée pour les autres composantes et une quantité limitée d'informations partagées entre les composantes.
- Basse dépendance des composants permet une maintenance isolée
- Basse dépendance des composants sépare les responsabilités d'entretien

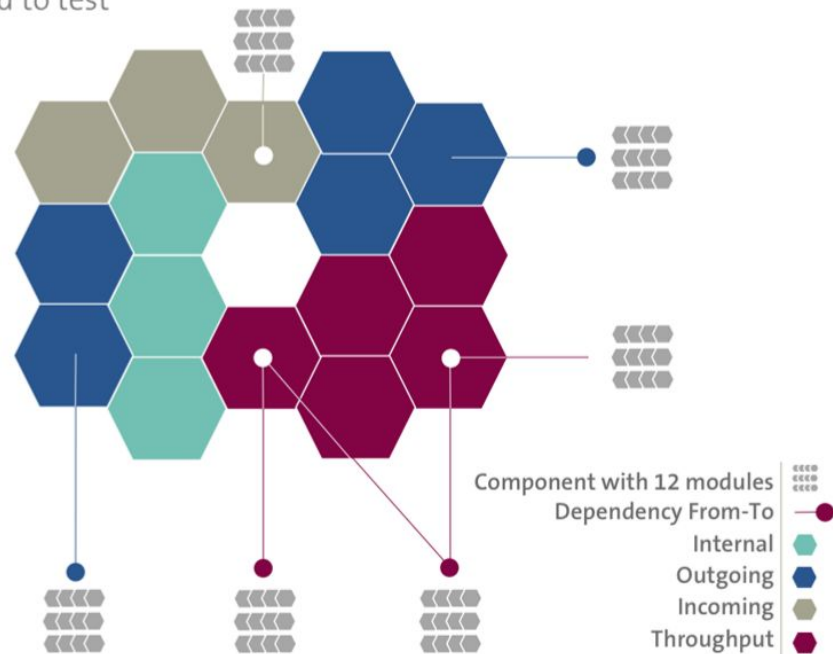
Loosely coupled components: easy to change in isolation

- > Few external dependencies
- > Code responsibilities separated
- > Easy to test



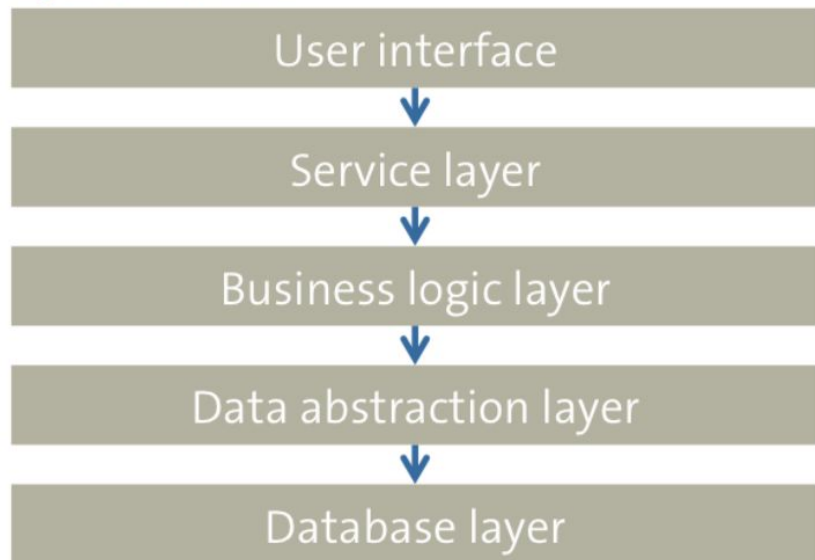
High coupled components: hard to change in isolation

- > Many external dependencies
- > Multiple code responsibilities
- > Hard to test

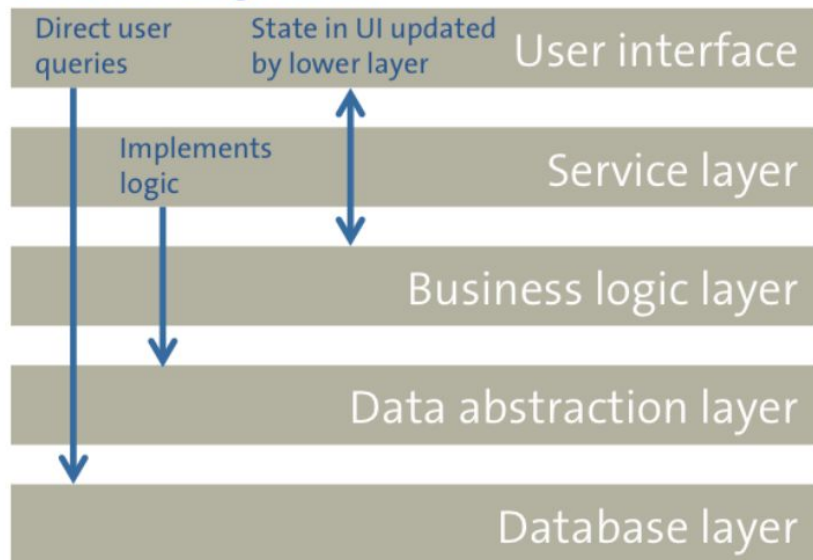


Concevoir contre l'architecture implémentée

Designed one-way dependencies from one layer to the next:

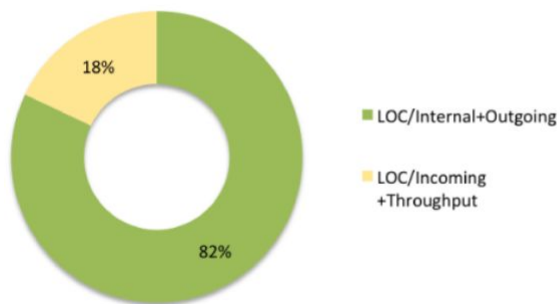


Over time, dependency violations (direct calls) lead to entanglement

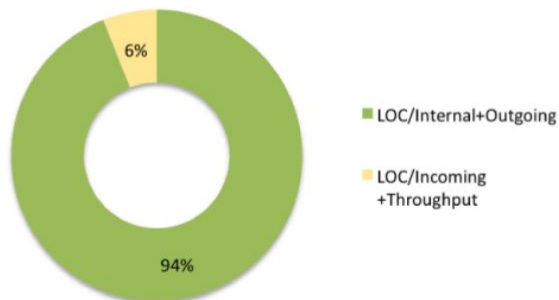


Trois profils de qualité pour l'indépendance des composants

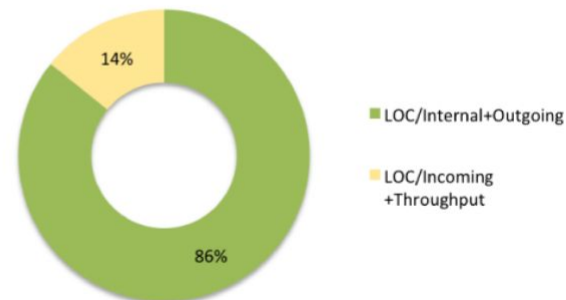
Component independence quality profile for Jenkins



Component independence quality profile of an anonymous ★★★★★ system

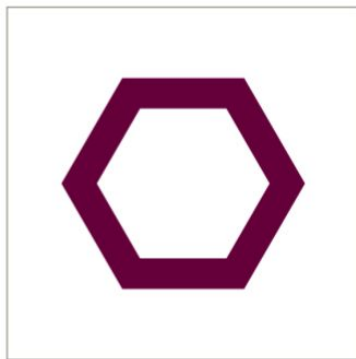


Component independence quality profile cut-offs for a ★★★★★ system

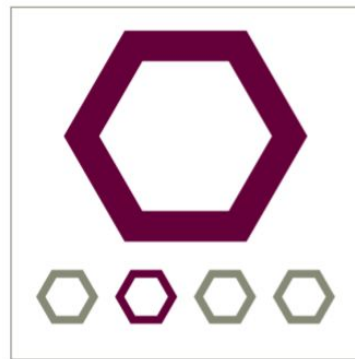


Conserver les composants d'architecture équilibrés

- Équilibrez le nombre et la taille relative des composants de niveau supérieur dans votre code.
- Faites cela par organisez le code source d'une manière telle que le nombre de composants soit proche de 9 (c'est-à-dire entre 6 et 12) et que les composants aient approximativement une taille égale
- Cela améliore la maintenabilité car les composants équilibrés facilitent la localisation du code et permettent une maintenance isolée.



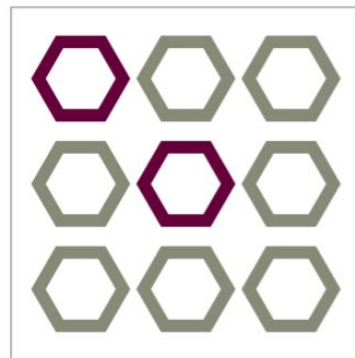
All changes will be in a single large component



Most changes will be in a single large component



Many changes scattered across multiple components

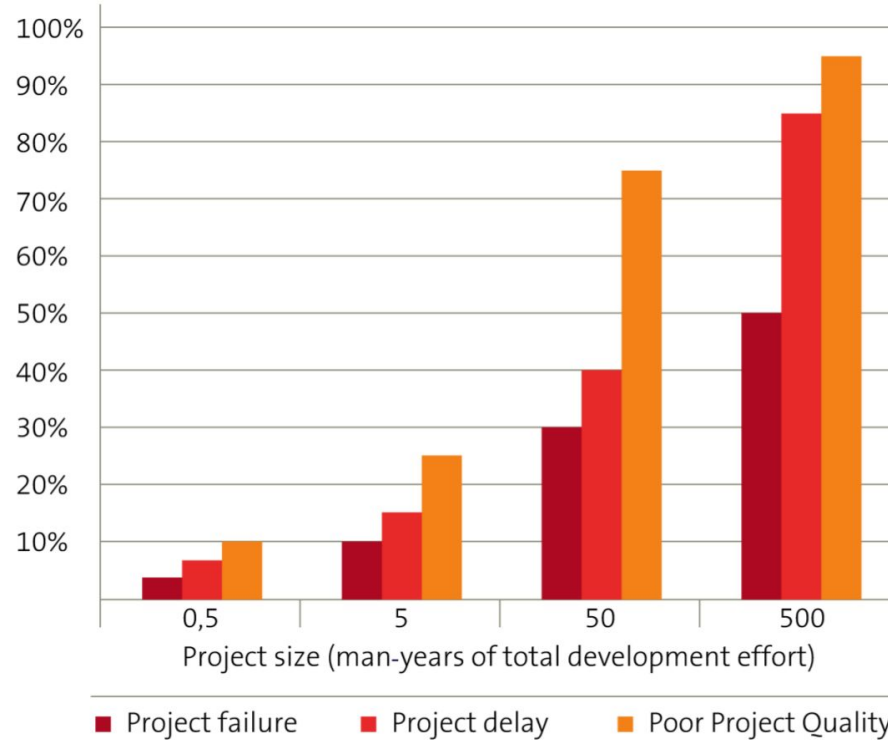


Changes can be isolated in one or two components of limited scope

Gardez votre Codebase petit

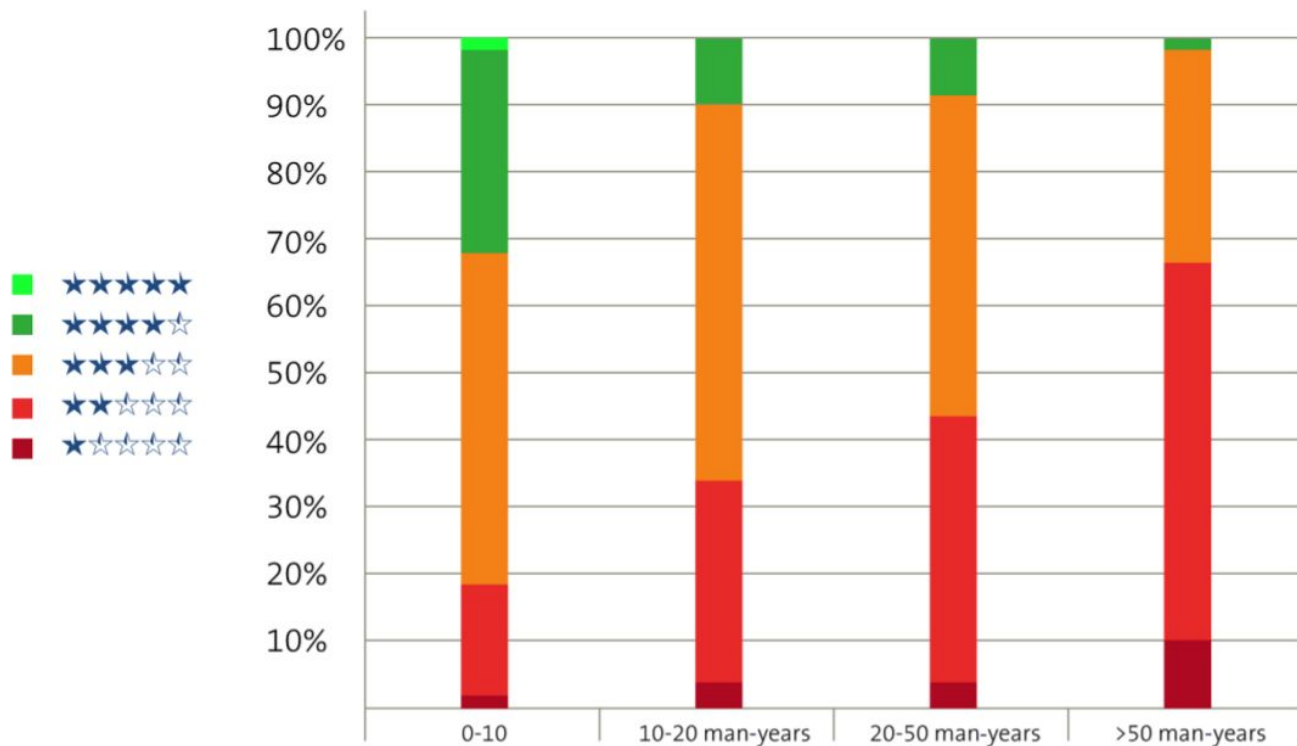
- Un Codebase est une collection de code source qui est stockée dans un dépôt de données.
- **Gardez votre Codebase aussi petite que possible.**
- Faire ceci par évitez la croissance du Codebase et réduisez activement la taille du système.
- Cela améliore la maintenabilité car avoir un petit produit, un projet et une équipe est un facteur de succès.

Probabilité d'échec du projet selon la taille du projet



Source: The Economics of Software Quality by Capers Jones and Olivier Bonsignour (Addison-Wesley Professional 2012). The original data is simplified into man-years (200 function points/year for Java).

Grandes Codebase sont plus difficiles à maintenir



Distribution of system maintainability in SIG benchmark among different volume groups

Automatiser les tests

- **Automatisez les tests** pour votre codebase.
- Faire ceci par la rédaction des tests automatisés à l'aide d'un **framework de test**.
- Cela améliore la maintenabilité car les tests automatisés rendent le développement **prévisible** et moins risqué.
- Mesurer la **couverture** pour déterminer s'il y a suffisamment de tests.
- Nous devrions viser une couverture de ligne d'au moins 80% avec un nombre suffisant de tests—c'est-à-dire, autant de lignes de code de test que le code de production.



Écrire un code propre

- Ne laissez aucune odeur de code derrière au niveau d'une l'unité.
- Ne laissez aucun mauvais commentaire derrière.
- Ne laissez aucun code dans les commentaires derrière.
- Ne laissez aucun code mort derrière.
- Ne laissez aucun nom d'identifiant long derrière.
- Ne laissez aucune constante magique derrière.
- Ne laissez aucune exception mal traitée derrière.


“Écrire du code propre est ce que vous devez faire pour vous appeler un professionnel.”

—Robert C. Martin

BetterCodeHub - la directive d'implementation



Better Code Hub  petrillo  SIGN OUT








Your repositories > Your results

Compliance  8 of 10

petrillo/
jpacman-framework-v5

Last analysis: a day ago Branch: master (default)

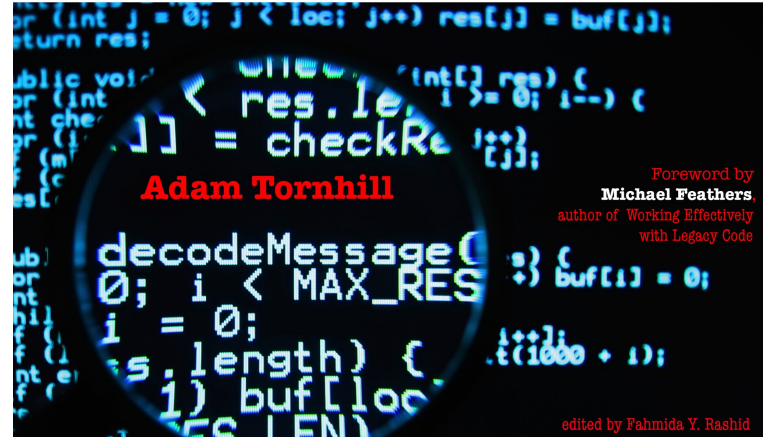
 

	Write Short Units of Code	✓	⋮
	Write Simple Units of Code	✓	⋮
	Write Code Once	✓	⋮
	Keep Unit Interfaces Small	✓	⋮
	Separate Concerns in Modules	✓	⋮
	Couple Architecture Components Loosely	✓	⋮
	Keep Architecture Components Balanced	✗	⋮

<https://bettercodehub.com/>

Your Code as a Crime Scene

Use Forensic Techniques
to Arrest Defects, Bottlenecks, and
Bad Design in Your Programs



Free available on <https://www.banq.qc.ca>

CODESCENE

BY EMPEAR®

The history of your code will decide its future.

Log in with GitHub



<https://codescene.io>



Fork me on GitHub

PMD is a source code analyzer. It finds common programming flaws like unused variables, empty catch blocks, unnecessary object creation, and so forth. It supports Java, JavaScript, Salesforce.com Apex, PLSQL, Apache Velocity, XML, XSL. Additionally it includes CPD, the copy-paste-detector. CPD finds duplicated code in Java, C, C++, C#, Groovy, PHP, Ruby, Fortran, JavaScript, PLSQL, Apache Velocity, Scala, Objective C, Matlab, Python, Go, Swift and Salesforce.com Apex.

Latest version(s)	<div>Latest version(s)</div> <h3>5.5.4 (25th February 2017)</h3> <ul style="list-style-type: none">• Release Notes• Download (Sourcecode, Documentation)• Online Documentation• Requires at least java 7, Salesforce.com Apex requires at least java 8 <h3>5.4.5 (25th February 2017)</h3> <ul style="list-style-type: none">• Release Notes• Download (Sourcecode, Documentation)• Online Documentation• Requires at least java 7
Get Involved	
Plugins	
Recent Announcements	
Next development version	
Previous versions	

<https://pmd.github.io/>

About

Checkstyle

[Release Notes](#)

[Consulting](#)

[Sponsoring](#)

Documentation

▼ Configuration

[Property Types](#)

[Filters](#)

[File Filters](#)

▼ Running

[Ant Task](#)

[Command Line](#)

▼ Checks

[Annotations](#)

[Block Checks](#)

[Class Design](#)

[Coding](#)

[Headers](#)

[Imports](#)

[Javadoc Comments](#)

[Metrics](#)

[Miscellaneous](#)

[Modifiers](#)

[Naming Conventions](#)

[Regexp](#)

[Size Violations](#)

[Whitespaces](#)

▼ Style Configurations

[Google's Style](#)



[Sun's Style](#)

Developers

Overview



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 is highly configurable and can be made to support almost any coding standard. An example configuration files are supplied supporting the [Sun Code Conventions](#) , [Google Java Style](#) .


A good example of a report that can be produced using Checkstyle and [Maven](#)  can be [seen here](#) .

Important Development Changes

As of September 2013, the Checkstyle project is using GitHub for hosting the following:

- [GitHub Source code repository](#)  - replacing the Mercurial repository on SourceForge.
- [GitHub Issue management](#)  - replacing the Bugs/Feature/Patches on SourceForge. All new issues should be raised at GitHub, and pull requests are now the preferred way to submit patches.

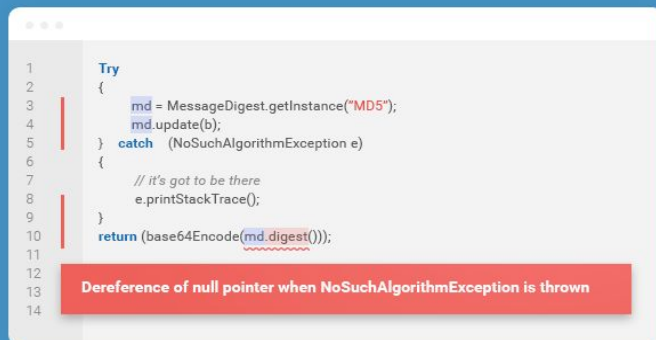
SourceForge will still be used for website hosting and binary hosting for downloads.

Releases will happen at the end of each month if functional changes exists in [master branch of our repo](#) .



The leading platform for

Continuous Code Quality

[DOWNLOAD 6.2](#)[USE ONLINE](#)

On 20+ languages >

Java

JavaScript

C#

C/C++

Cobol

[AND MORE](#)

Features

Write Clean Code

DevOps Integration

Centralize Quality

Effective SOFTWARE DEVELOPMENT SERIES 
Scott Meyers, Consulting Editor

CODE *Quality*

The Open Source Perspective



Diomidis Spinellis

TP2 - Qualité du code

En utilisant les approches que j'ai présentées aujourd'hui, analyser le projet open source que vous avez étudié précédemment.

Suivre les 10 guides?

Utilisez BetterCodeHub, CodeScene, Sonar, Et/ou tout autre outil.

Passez une bonne
“*semaine de relâche*”!