Cycle de vie des applications
La qualité du code

Christian DARGEIN – Axel ISOARD – Kostia MANNEVY – Simon PASQUEREAU

La qualité du code

- 1. Les enjeux d'un code propre et lisible
- 2. Standards et bonnes pratiques en programmation
- 3. Les outils pour vérifier la qualité d'un code
- 4. Les méthodes
- 5. Présentation de SonarQube

1. Les enjeux d'un code propre et lisible

- Enjeux
  - Concurentiel
  - Économique
  - Relationnel

1. Les enjeux d'un code propre et lisible

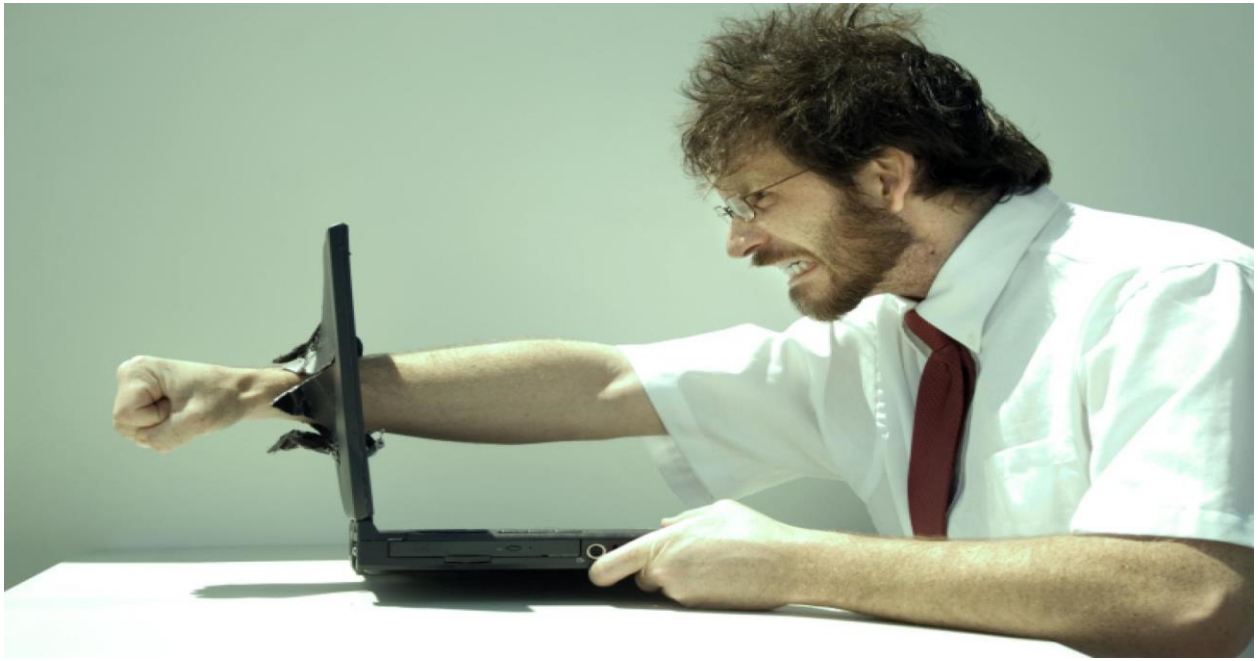
- Enjeu concurentiel
  - Difficile à tester
    - Perte de temps par rapport au concurrent
      - Enjeu Économique/Social

1. Les enjeux d'un code propre et lisible

- Enjeu économique
  - Relecture = Perte de temps
  - Ré-écriture coûteuse
    - Dette technique

1. Les enjeux d'un code propre et lisible

- Enjeu relationnel

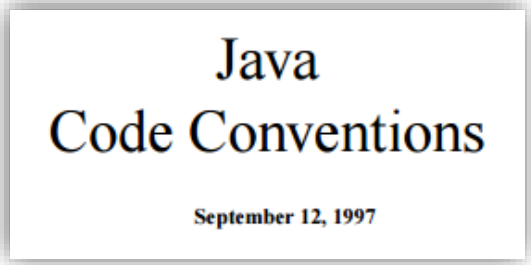


2. Standards et bonnes pratiques en programmation

- Beaucoup de règles de codage
- Buts recherchés:
  - Lisibilité
  - Compréhension
  - Problématique logicielle
- Pas de règle absolue
- Certaines règles impératives

2. Standards et bonnes pratiques en programmation

- Conventions dans certains langages



The GNU Coding Standards were written by Richard Stallman and other GNU Project volunteers. Their purpose is to make the GNU system clean, consistent, and easy to install. This document can also be read as a guide to writing portable, robust and reliable programs. It focuses on programs written in C, but many of the rules and principles are useful even if you write in another programming language. The rules often state reasons for writing in a certain way.

2. Standards et bonnes pratiques en programmation

- Des conventions spécifiques à des projets
- Avantages:
  - Test productifs
  - Beaucoup d'information fournie
  - Fonctions de recherches/modifications efficaces
  - Clarté améliorée
  - Esthétisme amélioré

2. Standards et bonnes pratiques en programmation

- Exemples de styles d'indentation

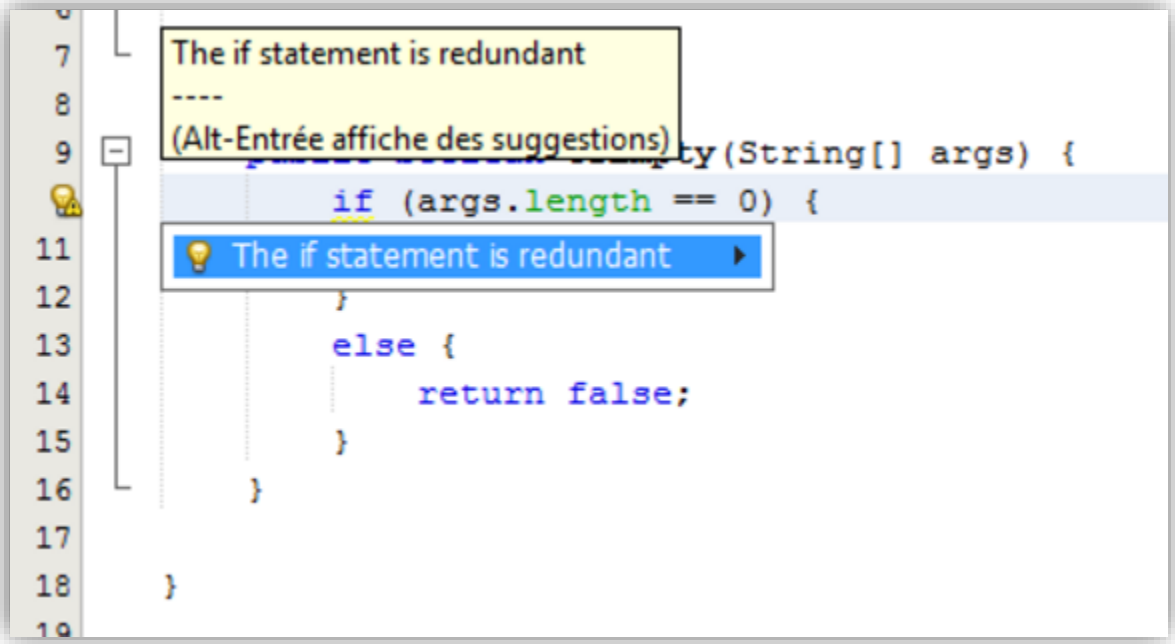
```
void a_function(void)
{
    if (x == y) {
        something1();
        something2();
    } else {
        somethingelse1();
        somethingelse2();
    }
    finalthing();
}
```

Style Kernighan et Richie

```
void a_function(void)
{
    if (x == y)
    {
        something1();
        something2();
    }
    else
    {
        somethingelse1();
        somethingelse2();
    }
    finalthing();
}
```

Style Allman (proche du style Whitesmiths)

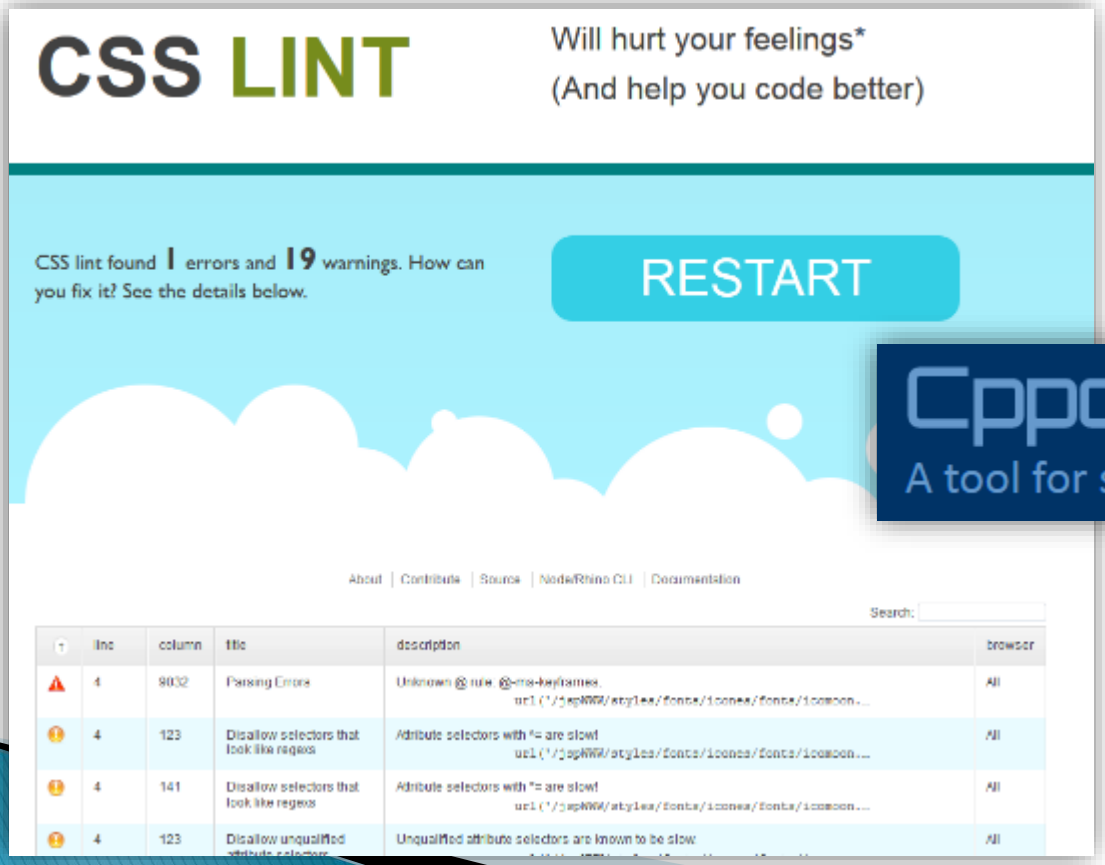
3. Les outils pour vérifier la qualité d'un code




```
public boolean isEmpty(String[] args) {
    return args.length == 0;
}
```

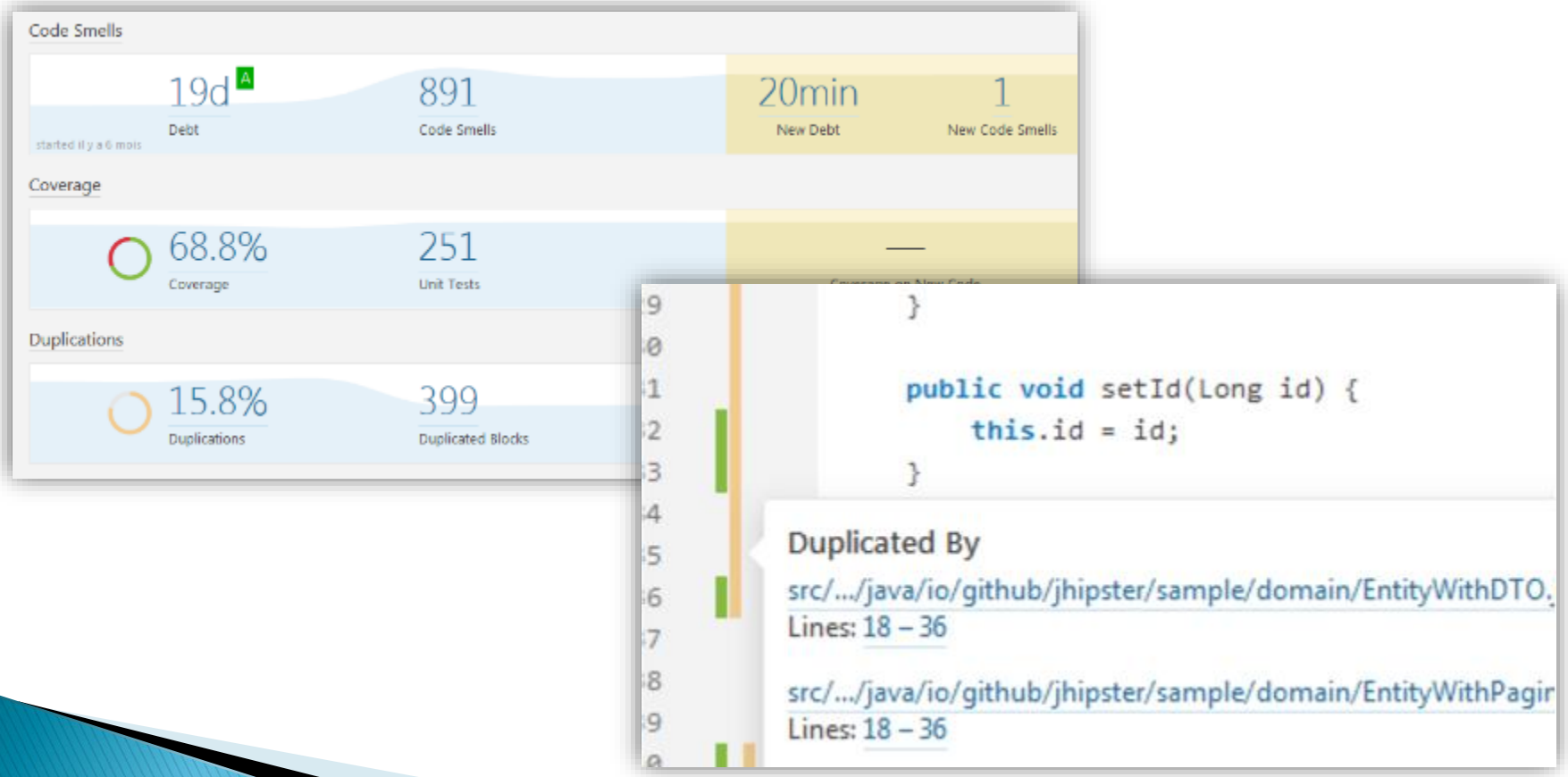
3. Les outils pour vérifier la qualité d'un code

- Les outils au « coup par coup »



3. Les outils pour vérifier la qualité d'un code

- Les outils d'analyse continue



3. Les outils pour vérifier la qualité d'un code

- Le remaniement = « refactoring »

```go
// filename: message.go
package message
import "fmt"
func FormatMessage(name string) string{
if len(name) == 0 { return "Welcome" } else { return fmt.Sprintf("Hi, %s", name) }
}
```

Output:

```go
// filename: message.go
package message

import "fmt"

func FormatMessage(name string) string {
        if len(name) == 0 {
                return "Welcome"
        } else {
                return fmt.Sprintf("Hi, %s", name)
        }
}
```

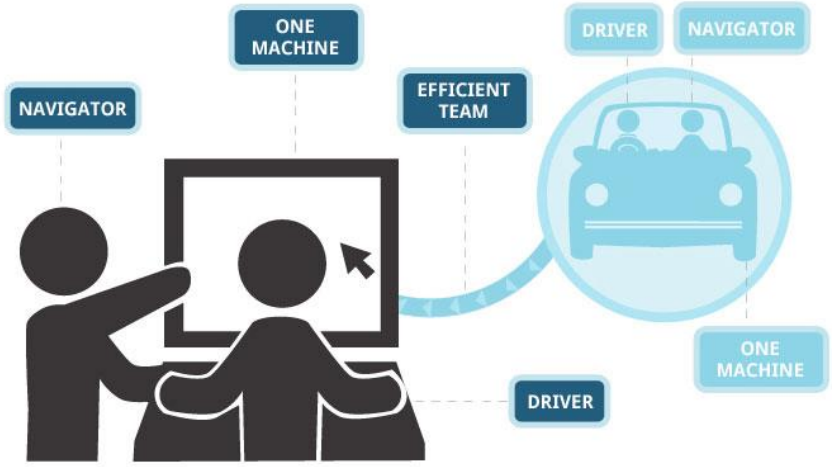
4. Les méthodes

- Les fondamentaux

- Pratiques de base dès la première rédaction
- Ne pas indenter
- Ne pas commenter
- Faire des classes/méthodes trop longues
- Respect des principes de la POO
- Etc.

4. Les méthodes

- Une méthode humaine très efficace = le « pair programming »
  - Revue du code immédiate
  - Fonction de « co-pilote »
  - Utilisée en Xtrem Programming



4. Les méthodes

- L'analyse de code automatisée: exemple de SonarQube

- L'implémentation de tests automatisés, permettant de remonter très rapidement la source du problème en cas de refactoring par exemple.
- La "*Boy Scout Rule*" qui est plus un principe qu'une méthode, peut se résumer par *"A chaque fois que je parcours du code, je dois essayer de l'améliorer"*.