Qualité logicielle et dette technique

Comment gérer son code source ?

Pr. Xavier Blanc – Xavier.Blanc@u-bordeaux.fr

Qualité logicielle : un enjeu majeur

Le logiciel s’exécute-t-il exactement comme son utilisateur le souhaite ?

Deux objectifs principaux pour l’utilisateur

- Service
- Usabilité

Impact pour le propriétaire

- Le logiciel fait partie de sa stratégie business
- Il est utilisé par ses clients

=> Image de marque

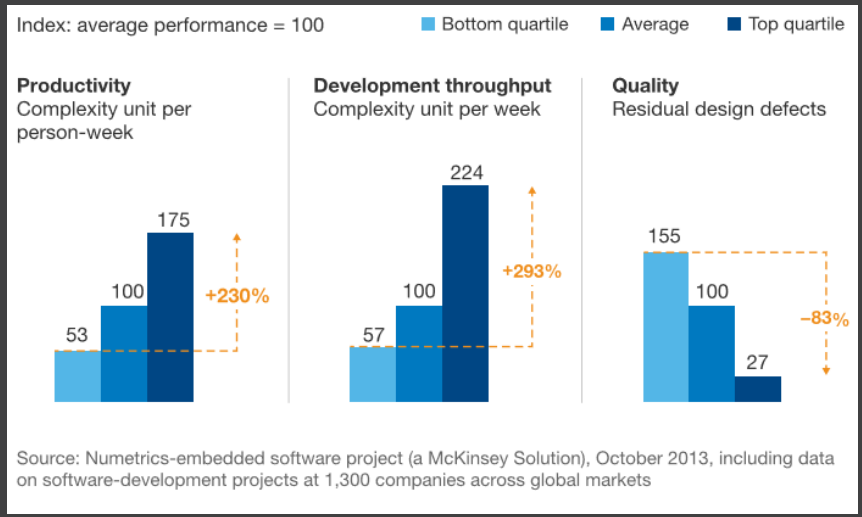
- Le logiciel est utilisé par ses collaborateurs
- Il est nécessaire au développement du business

=> Organisation

Qualité Logicielle et Business

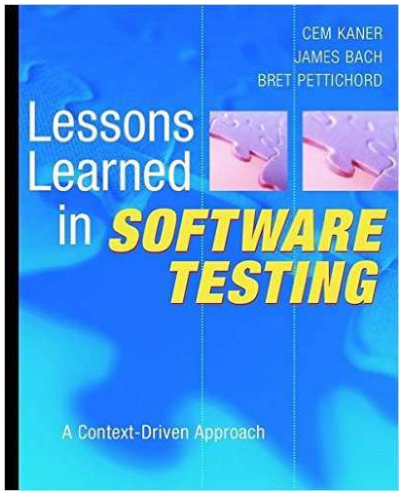
- Les entreprises deviennent soucieuses de leur qualité logicielle
- Les investissements croissent de 15% par an
- 35% du budget IT en moyenne (**efficace à partir de 30%**)

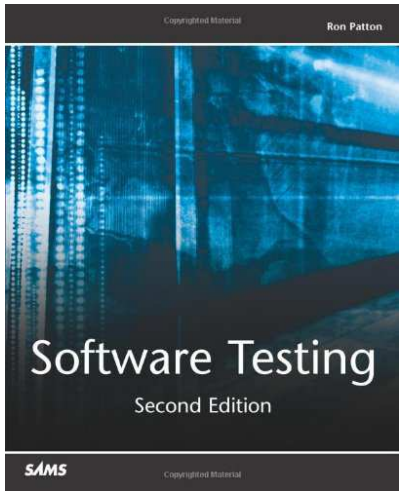
=> Un écart important entre ceux qui investissent et les autres !
(productivité, nombre de défauts…)



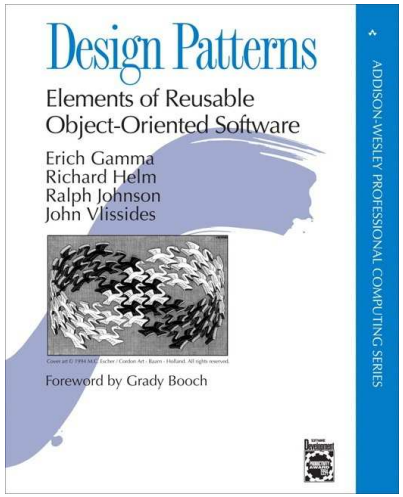
Approches : Tests et/ou Code Clean

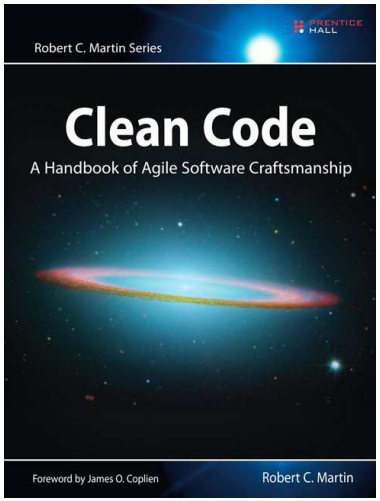
- Trouver les erreurs avant l'utilisateur, et corriger



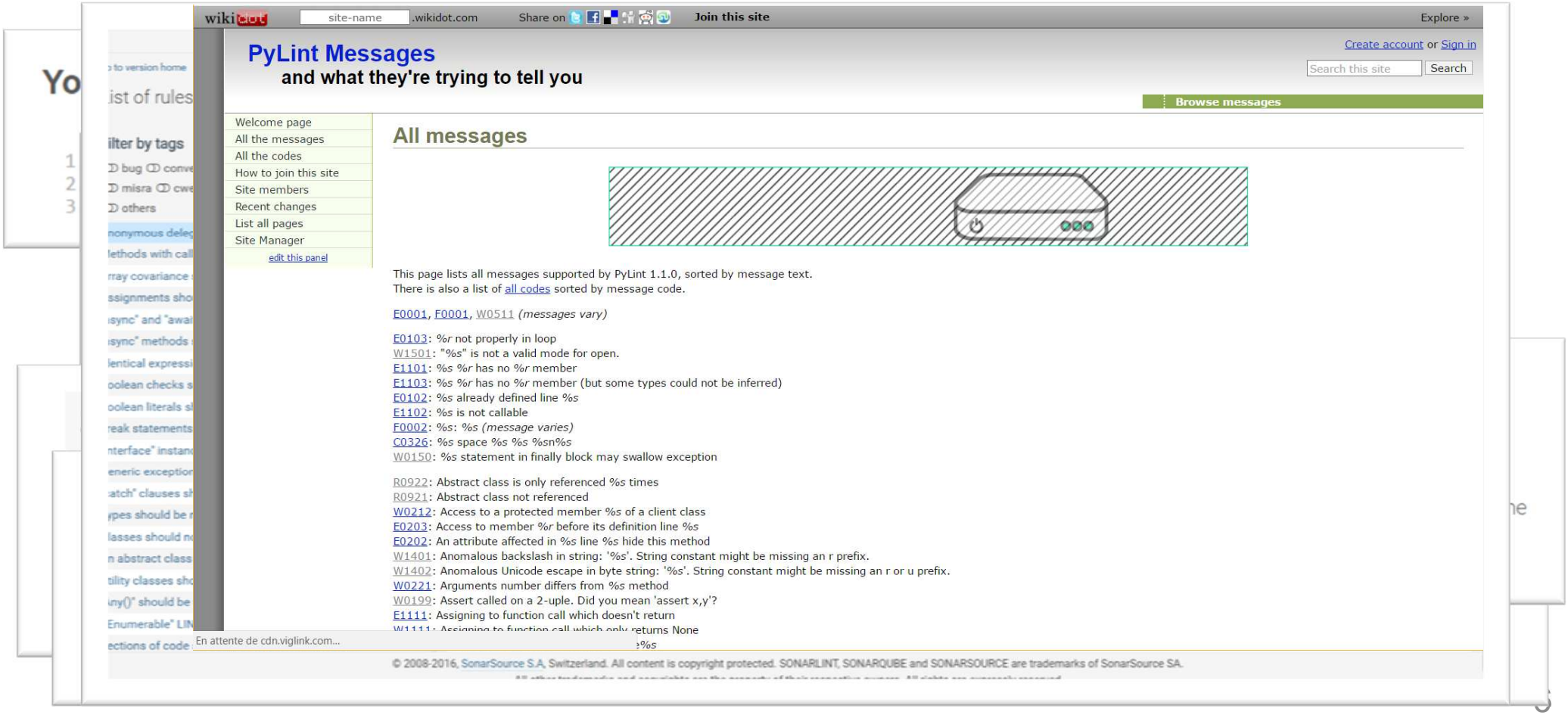


- Coder « propre » pour éviter les erreurs, et améliorer l'évolution





Règles de codage pour coder « propre »



Coder « propre » ou pas

« Code de mauvaise qualité que l'on corrigera plus tard »

Dette Technique !

=> Payer la dette : Temps nécessaire (en jour-homme) pour rendre le code propre

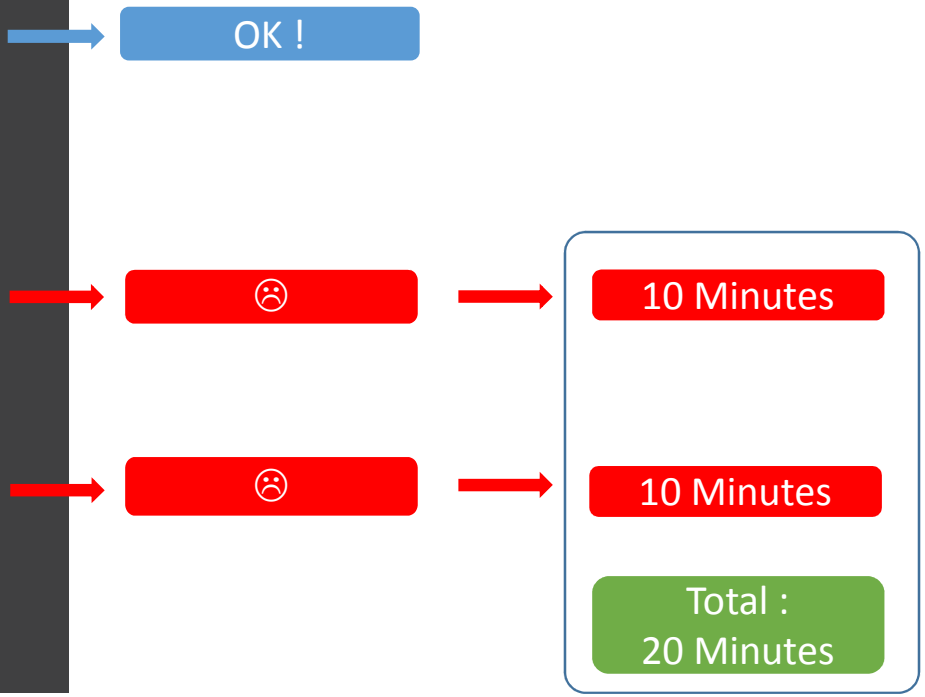
Exemple de dette : La documentation

Exemple de bonne pratique : « chaque fonction doit être commentée »

```
/** Cette fonction effectue une addition */
function addition(int a, int b) {
    return a + b;
}

function soustraction(int a, int b) {
    return a - b;
}

function multiplication(int a, int b) {
    return a * b;
}
```



Outils de mesure de la dette

Les « Linter » analysent automatiques les coding standards:

- Java
  - PMD : 300 règles
  - Checkstyle : 153 règles
  - Sonar : 376 règles
- JavaScript
  - EsHint : 200 règles
  - JsHint : 196 règles
- Ruby
  - Rubocop : 120 règles
- Python
- …

```
except socket.error, (errno, strerror):
        print "ncfiles: Socket error (%s) for host %s (%s)" % (errno,
for h3 in page.findAll("h3"):
    value = (h3.contents[0])
    if value != "Afdeling":
        print >> txt, value
        import codecs
        f = codecs.open("alle.txt", "r", encoding="utf-8")
        text = f.read()
        f.close()
        # open the file again for writing
        f = codecs.open("alle.txt", "w", encoding="utf-8")
        f.write(value+"\n")
        # write the original contents
        f.write(text)
        f.close()
```

Impact positif vérifié !

Etudes sur plusieurs logiciels (GitHub, Google, Microsoft)

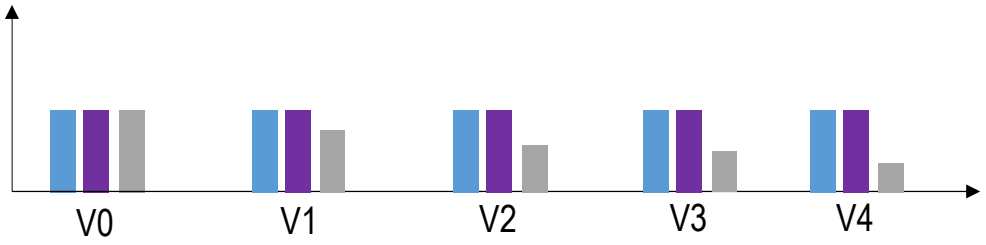
- Impact des métriques statiques sur les bugs
  - Analyse quantitative
- Poids du passé
  - Ne rien faire coûte

- Organisation du développement
  - Difficulté de la mise en place

Les coûts de la qualité

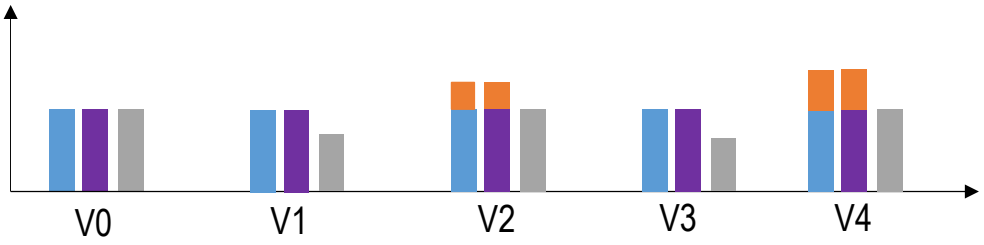
Coût
Délai
Qualité
Overhead

Aucune gestion de la qualité



Coût risqué important

Gestion de la qualité par objectif



Overhead important

Automatiser les bonnes pratiques de qualité logicielle?

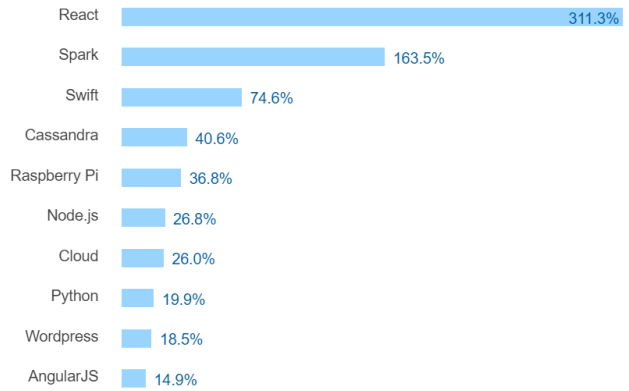
Code source : Respect de conventions et de règles (langages, frameworks)

Méthodologie : Agilité, Code review, DevOps, …

Tests : Couverture de test, Test-driven development, …

Standard ou Custom , chaque entreprise met en place ses propres pratiques

Les pratiques évoluent



Augmentation des tendances sur StackOverflow (Jan 2015 – Jan 2016
http://stackoverflow.com/research/developer-survey-2016

Les technologies et frameworks se renouvellent en permanence

De nouveaux paradigmes apparaissent (DevOps)

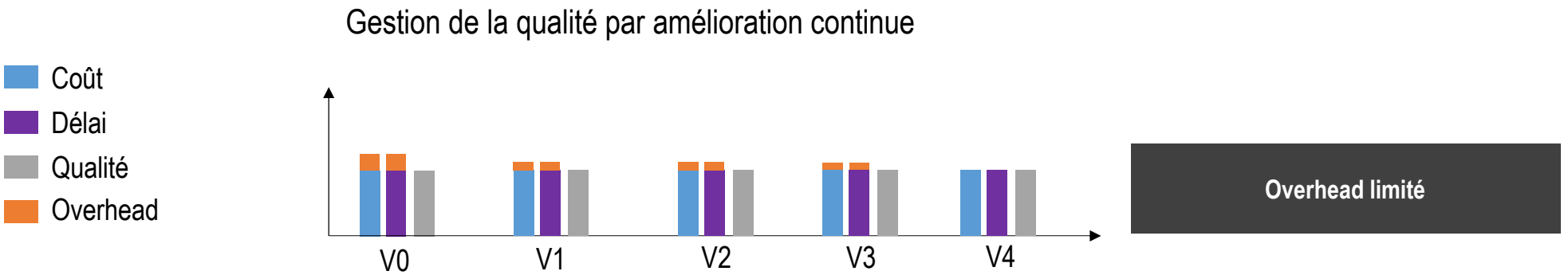
Suivre la qualité en temps réel

Contribution

Impact

=> Importance du procédé mais aussi de l’humain

La qualité, une (pré)occupation constante



Conclusion

Pour les propriétaires

- Ne pas négliger la qualité
- Faire une estimation
- Organiser votre production (liberté vs efficacité)

Pour les utilisateurs

- Estimer la qualité du logiciel
- Ne pas s'enfermer (solutions ouvertes)
- Faire pression sur les propriétaires ?

R&D en qualité logicielle

- Mesure d’impact (ROI)
- Les bonnes pratiques dédié
- Gestion des évolutions (API)
- Automatisation de l’intégration (pull request, merge, etc.)
- Gestion d’équipe et qualité