

Comment automatiser les tests sur tout le cycle projet pour assurer la qualité des logiciels

En route vers le Continuous Deployment



1. Contexte
2. Le Continuous Deployment et l'automatisation sont liés
3. Automatisation des tests unitaires et d'intégration : La qualité au bout des développements
4. L'automatisation des tests système : prendre la place de l'utilisateur
5. Intégrer les tests et le Continuous Deployment



Forte pression pour faire baisser le Time To Market

- Les géants du Web ont popularisé l'importance du logiciel comme avantage concurrentiel
 - Etre présent très tôt sur un marché pour occuper le terrain
 - Mettre souvent en production pour diffuser de nouvelles fonctionnalités
 - Avoir des retour rapides de la part des utilisateurs
- Développement de l'Agilité pour être plus réactif
- Extension des pratiques DevOps pour fluidifier le processus de mise en production
- Dans ce contexte, Continuous Delivery et Continuous Deployment sont des pratique fortement recommandées



**LE CONTINUOUS DEPLOYMENT
ET L'AUTOMATISATION SONT
LIÉS**



Le Continuous Deployment et l'automatisation sont liés

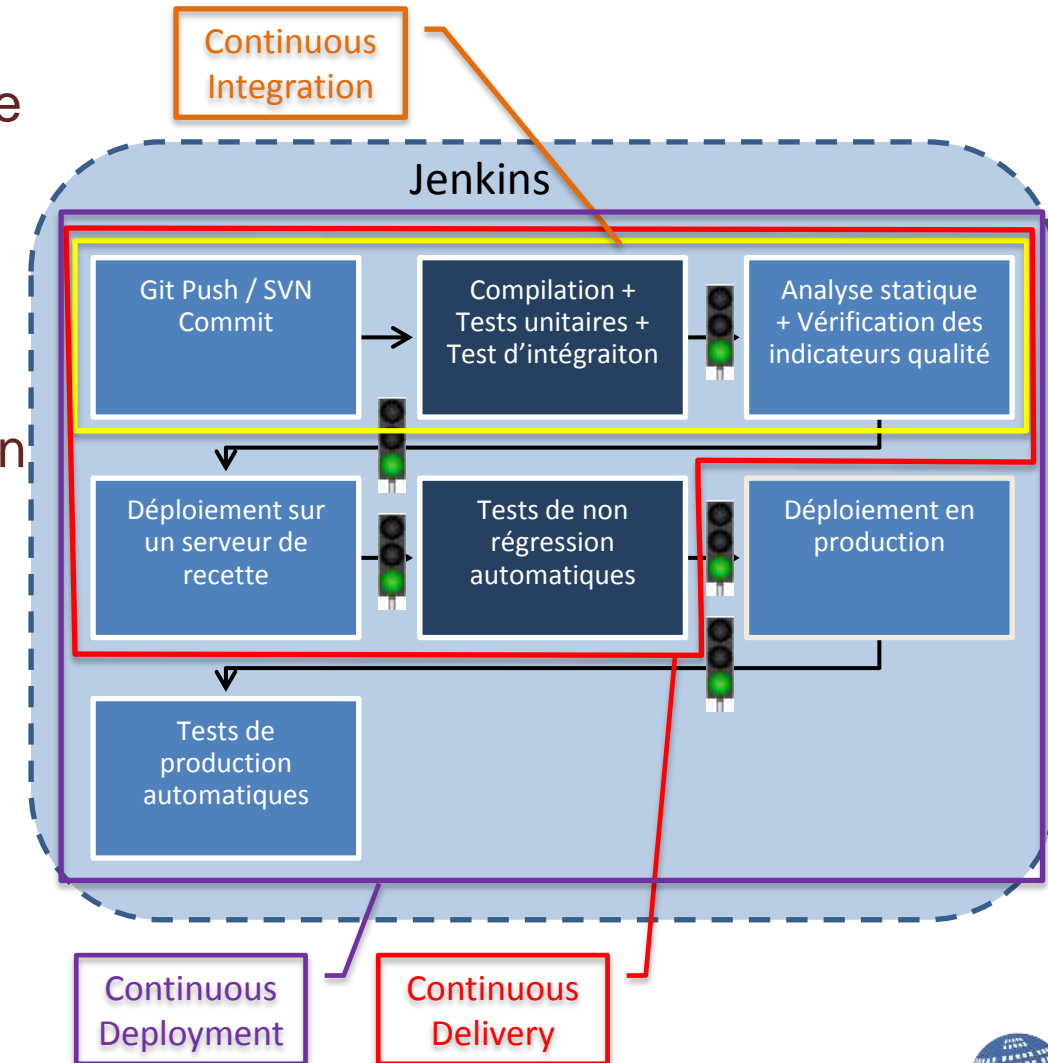
Que sont le Continuous Delivery et le Continuous Deployment ?

- Le Continuous Deployment est un processus **automatique** de contrôle, de livraison et de déploiement en production
 - A l'origine était l'Intégration Continue (CI) destinée aux développeurs
 - Continuous Delivery = Continuous Integration + tests fonctionnels automatiques
 - Continuous Deployment = Continuous Delivery + déploiement automatique en production

Il permet de:

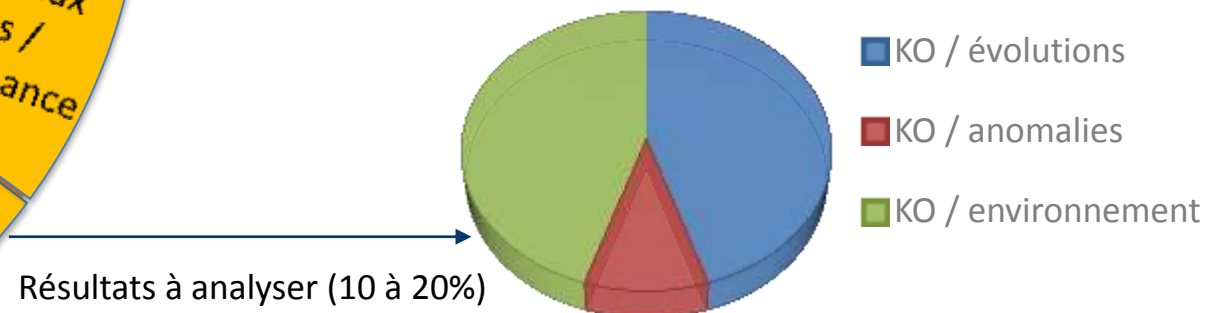
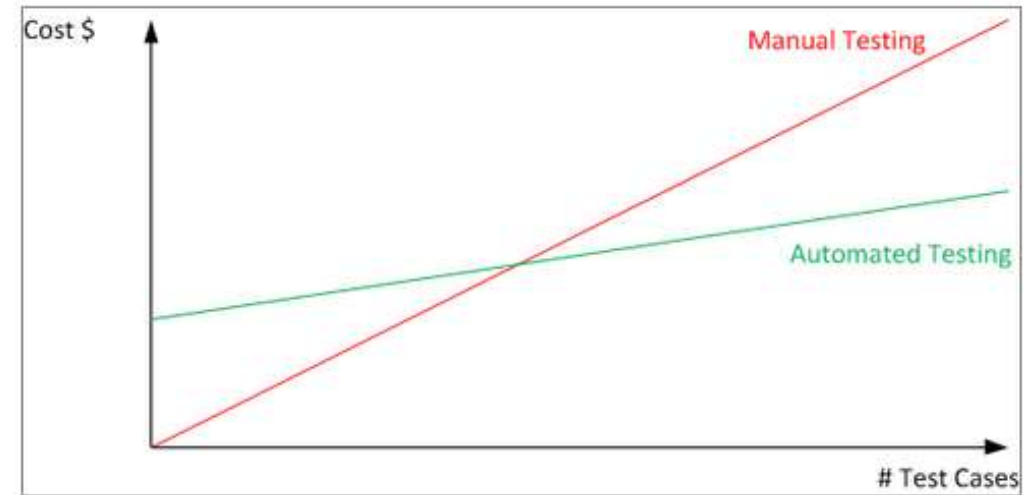
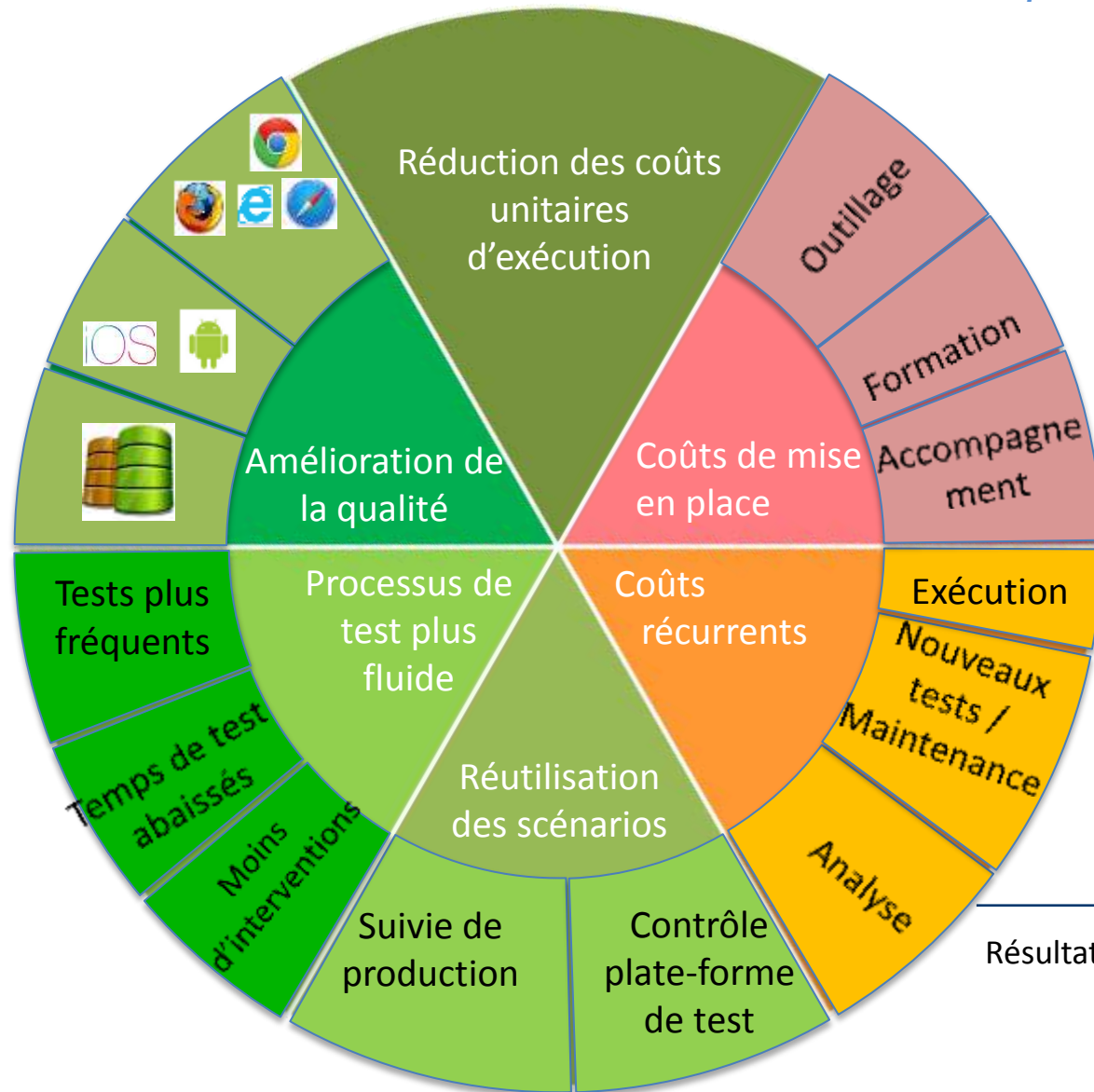
- Accélérer le processus de MEP
- Sécuriser la MEP
- Rendre reproductible un cycle complet
- Avoir des retours rapides sur la qualité

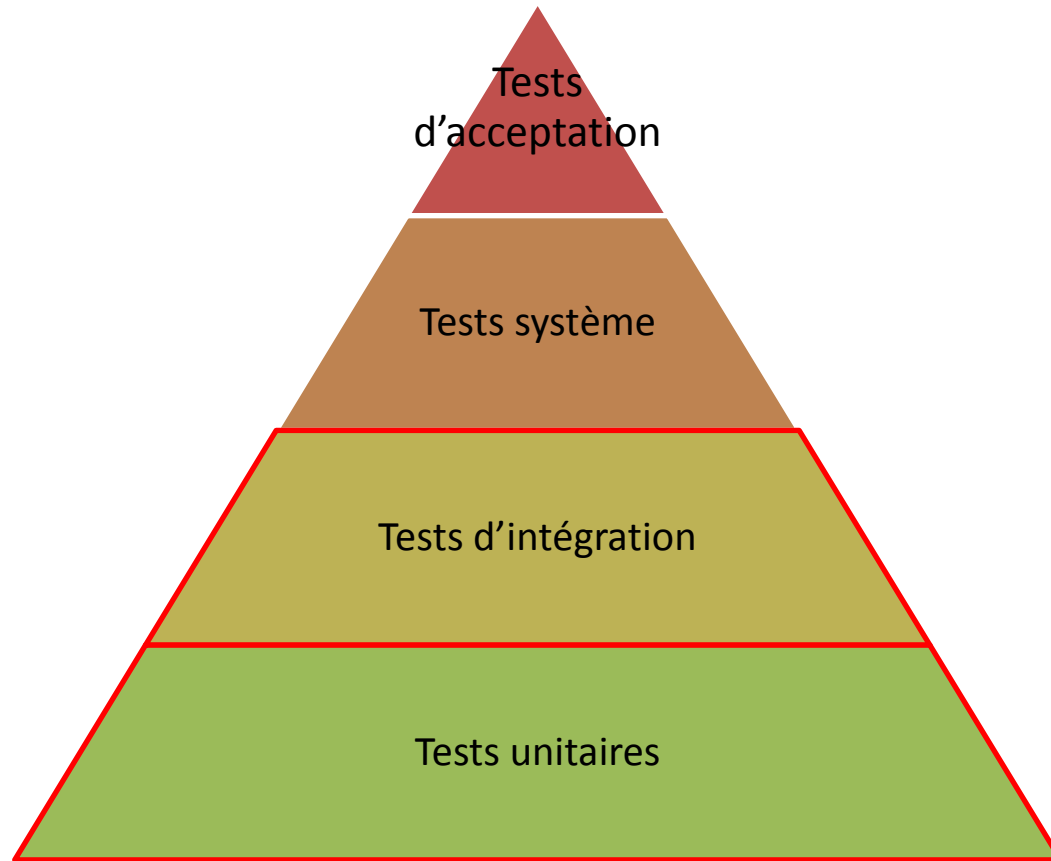
- Il faut avoir confiance → Tester!



Le Continuous Deployment et l'automatisation sont liés

Qu'est ce qui incite à automatiser les tests en Continuous Deployment ?





AUTOMATISATION DES TESTS UNITAIRES ET D'INTÉGRATION : LA QUALITÉ AU BOUT DES DÉVELOPPEMENTS

- Objectif: permettre aux développeurs d'avoir un retour rapide sur la qualité du code

- Comment:

- Utiliser un framework de tests unitaires

JUnit

nunit



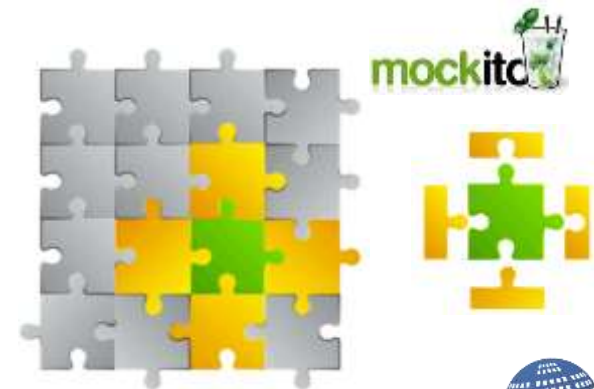
- Ne pas se contenter des nominaux

- Cas passants
- Cas aux limites
- Cas d'erreur

- Maîtriser les jeux de données

- Isoler un composant / une liaison pour le tester (bouchons)

Avantages	Inconvénients
Permet de tester des cas de figures spécifiques	Peut être long à réaliser / à maintenir
Améliore la stabilité des tests et de l'application	Demande des compétences
Jeux de données / communications externes maîtrisés	



- Pour contribuer efficacement à la qualité de l'application, la couverture des tests doit être importante

- Fixer un seuil minimum
- Suivre le taux de couverture à chaque livraison
- On peut utiliser des taux de couverture différenciés

Coverage				
64.4%	45.3%	46.1%	Overall Line Coverage	67.8%
Overall Coverage	Coverage	IT Coverage	Overall Condition Coverage	55.2%
	364		Overall Uncovered Lines	1,864
	Unit Tests		Overall Uncovered Conditions	960
			Line Coverage	47.4%
			Condition Coverage	39.4%
			Uncovered Lines	3,040
			Uncovered Conditions	1,299
			IT Line Coverage	50.7%
			IT Condition Coverage	33.6%
			IT Uncovered Lines	2,852
			IT Uncovered Conditions	1,423
			Lines to Cover	5,782
			Unit Test Errors	0
			Unit Test Failures	1
			Skipped Unit Tests	0

```
public String sayHello(String userId) {  
    String message = null;  
  
    /* Get the user from the DAO. */  
    User user = null;  
  
    if (StringUtils.isNotBlank(userId)) {  
        user = this.userDao.getUser(userId);  
    }  
  
    if (null != user) {  
        message = MessageFormat.format(MSG_HELLO,  
    }  
  
    if (StringUtils.isBlank(message)) {  
        message = StringUtils.EMPTY;  
    }  
  
    return message;  
}
```



- L'analyse statique du code permet également de limiter le risque d'anomalies

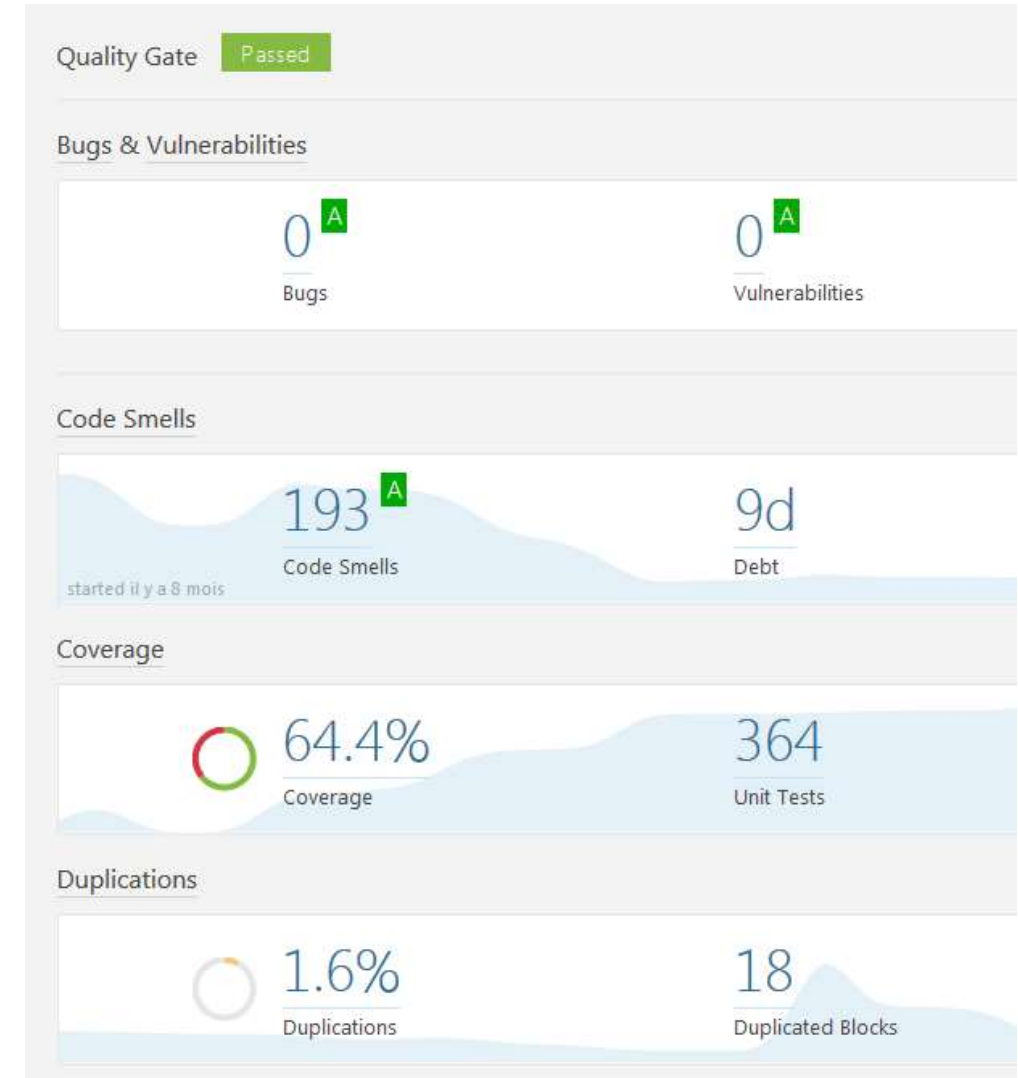
```
public void compare(String uneChaine) {  
    "machaine".equals(uneChaine);  
}  
  
public void compare(String uneChaine) {  
    uneChaine.equals("machaine");  
}
```



**Risque de
NullPointerException**

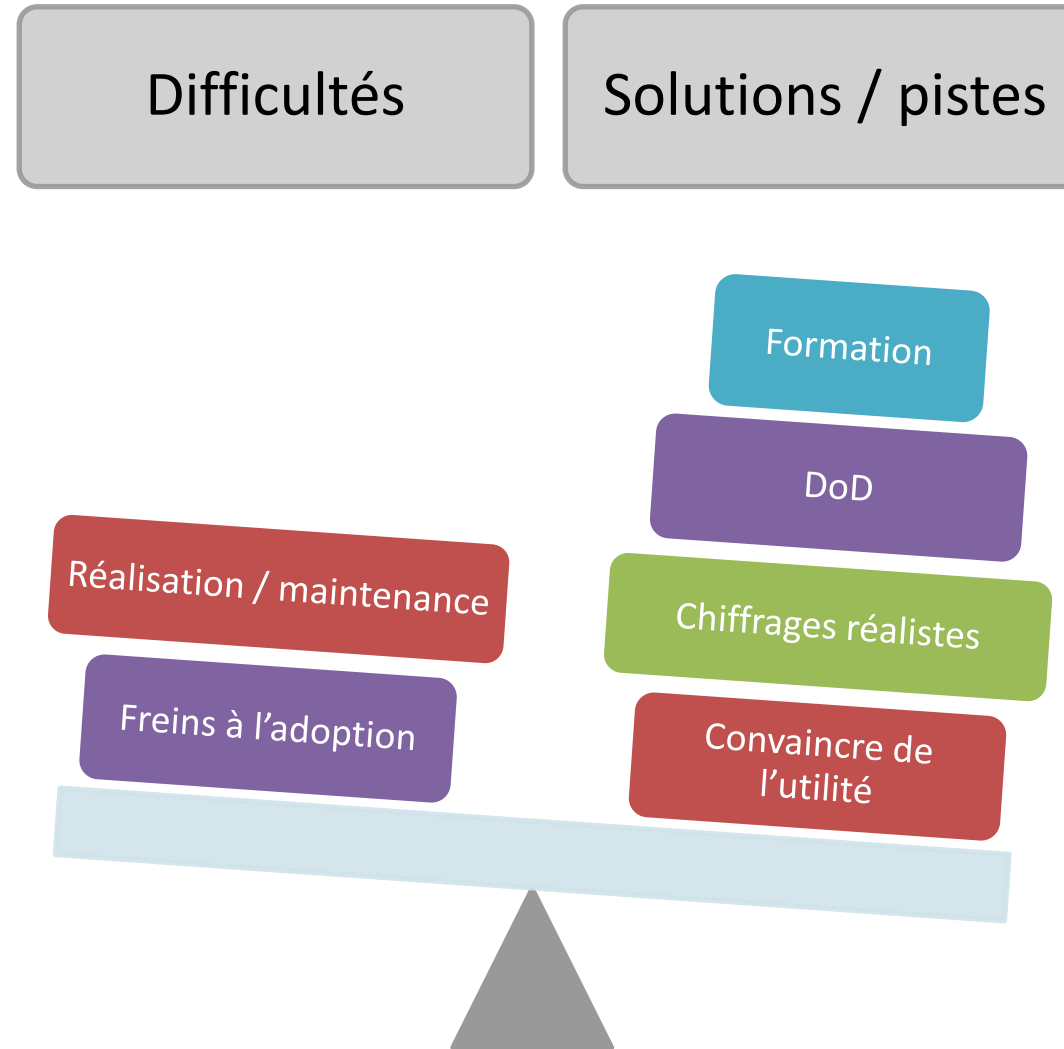
- La dette technique est un indicateur

- De maturité du code
- De risque d'anomalies futures



Automatisation des tests unitaires et d'intégration

Mettre en œuvre des tests unitaires et d'intégration ne va pas de soi



Une fois mature, le Test Driven Development peut permettre d'aller plus loin



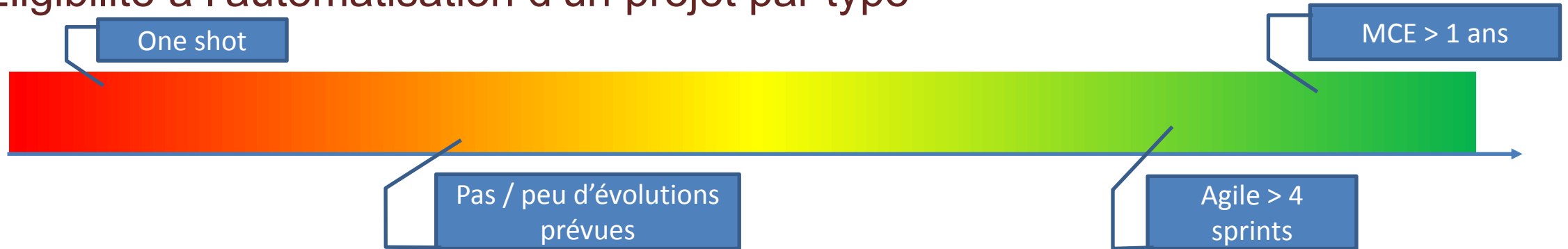
Regression:
"when you fix one bug, you
introduce several newer bugs."



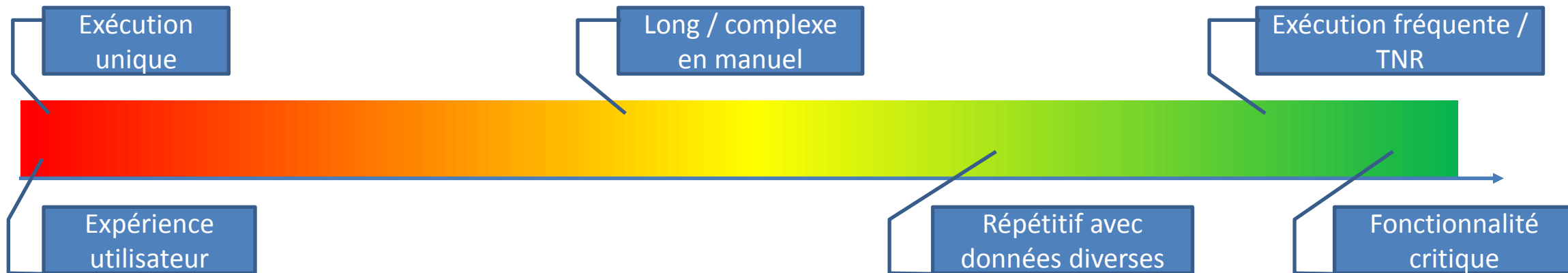
**L'AUTOMATISATION DES TESTS
SYSTÈME : PRENDRE LA PLACE
DE L'UTILISATEUR**



■ Eligibilité à l'automatisation d'un projet par type



■ Eligibilité à l'automatisation d'un cas de test



- Il faut automatiser un test éligible au plus tôt après la première exécution manuelle pour maximiser le ROI



L'automatisation des tests système

Automatiser nécessite des outils spécifiques

Tests IHM Web



Tests mobiles



Tests services



Framework de test



SeleniumRobot

Tests batches



- Lire le Syllabus CFTL niveau fondation § 6.3!
- Impliquer les équipes dans l'utilisation des nouveaux outils
 - Faire participer toute l'équipe à la réflexion
 - Adapter les outils aux besoins spécifiques de l'équipe
 - Le testeur va passer plus de temps à concevoir et analyser qu'à exécuter les tests
- Si possible dédier une personne à l'automatisation
 - Maintenance des outils et scripts
 - En communication étroite avec les développeurs et les testeurs
- Ne pas chercher à généraliser tout de suite

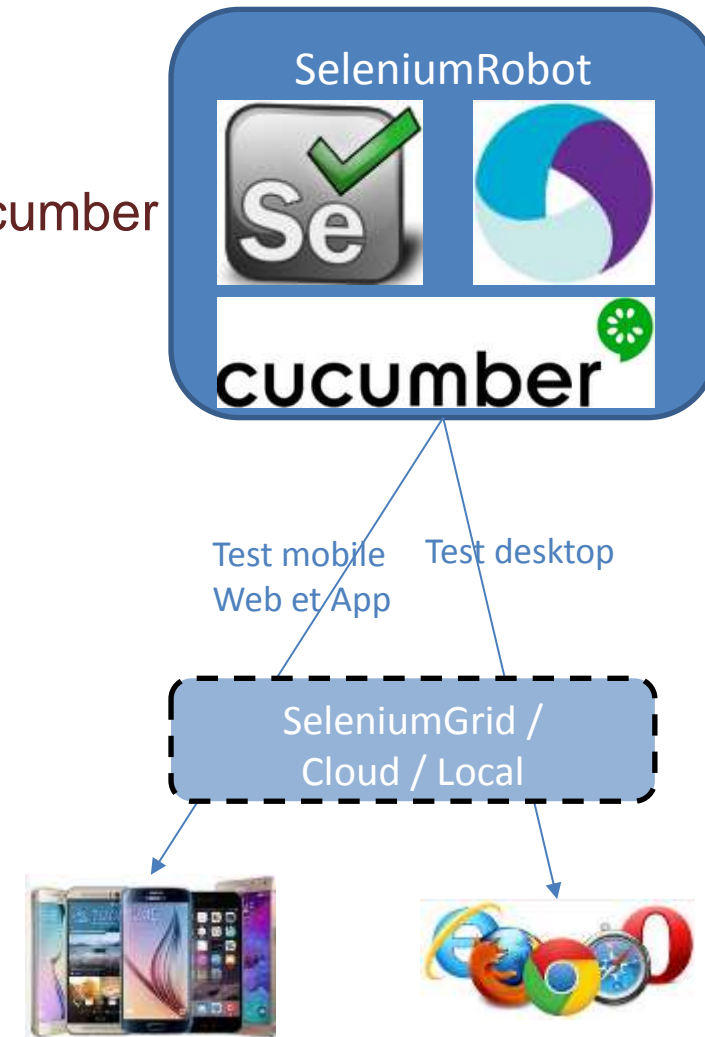
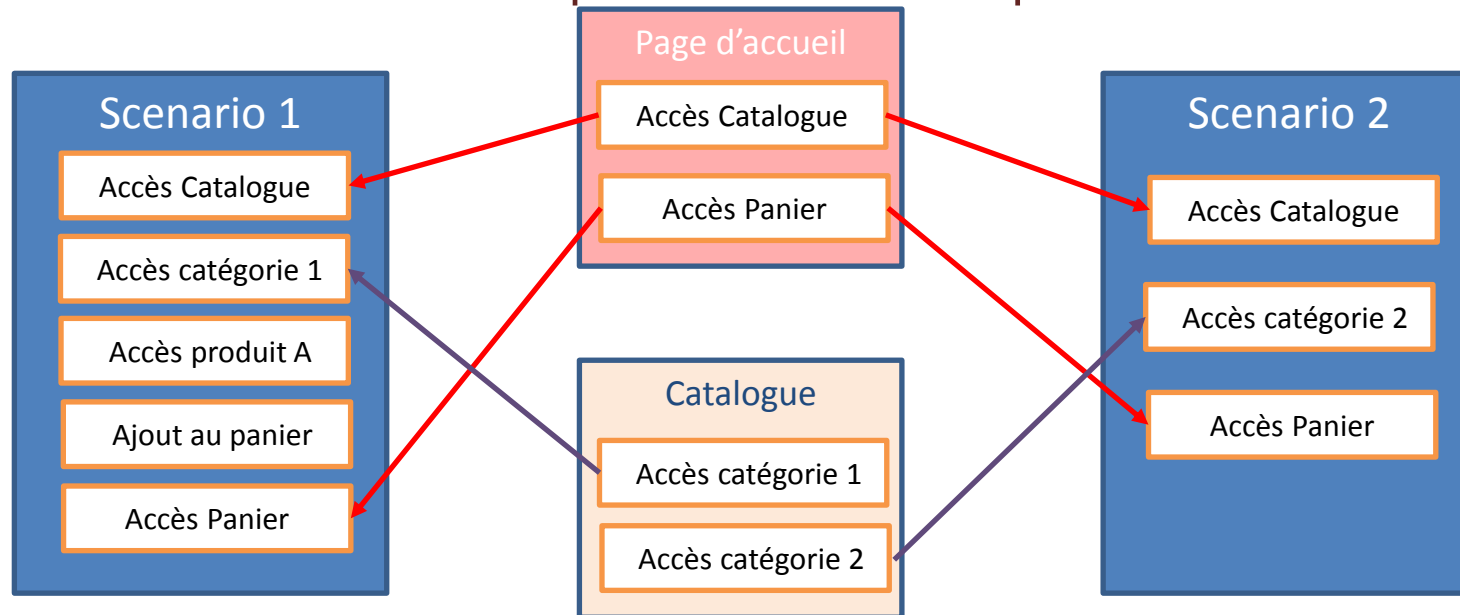
Automatisation sur application
pilote

Retour d'expérience et adaptation
des pratiques

Généralisation sur le périmètre
choisi



- Facilite l'écriture et l'exécution des tests
 - Limitation des configurations
 - Fiabilisation de l'API Selenium
 - Met en œuvre la méthode du langage naturel avec Gherkin / Cucumber
- Limite les coûts de maintenance
 - Réutilisation des briques de tests dans plusieurs scénarios



Réduire les coûts

ADAPTER LA STRATEGIE AU PUBLIC / CONTEXTE

- Ne pas viser l'exhaustivité
- Sélectionner les plates-formes les plus pertinentes

OPTIMISER LA CREATION ET LA MAINTENANCE

- Ne pas multiplier les outils
- Capitaliser sur les compétences
- Utiliser un framework de test
- Etablir une collaboration entre testeurs et développeurs
- Gérer l'automatisation comme tout développement logiciel

ROI

Maximiser les gains

PARALLELISER LES TESTS

- Multiplier les environnement de test et les automates
 - ⚠ Coût des environnements
- Avoir un retour plus rapide sur l'état du logiciel
- Faire plus de tests différents en temps limité
 - ⚠ Temps d'analyse!

MULTIPLIER LES USAGES

- Valider l'application
- Vérifier les déploiements
- Surveiller les environnements
- Valider sur des environnements différents

MULTIPLIER LES SCENARIOS DERIVES

- Cas d'erreur
- Scénarios nominaux
- Scénarios aux limites

Faire en sorte que les tests automatiques soient exploités et ne soient pas obsolètes au bout de quelques mois

1. Ne pas mettre la charrue avant les boeufs
2. Convaincre son équipe et les responsables
3. Justifier les coûts par des gains prévisionnels
4. Repenser l'organisation et les pratiques
5. Choisir l'outillage en impliquant les utilisateurs
6. Définir une stratégie d'automatisation
7. Réaliser un POC
8. Retour d'expérience et correction
9. Impliquer tous les acteurs (formation et information, REX)
10. Généraliser sur un projet
11. Gérer la maintenance
12. Mesurer les avancements et la qualité
13. Suivre les résultats et analyser les erreurs
14. Généraliser sur plusieurs projets

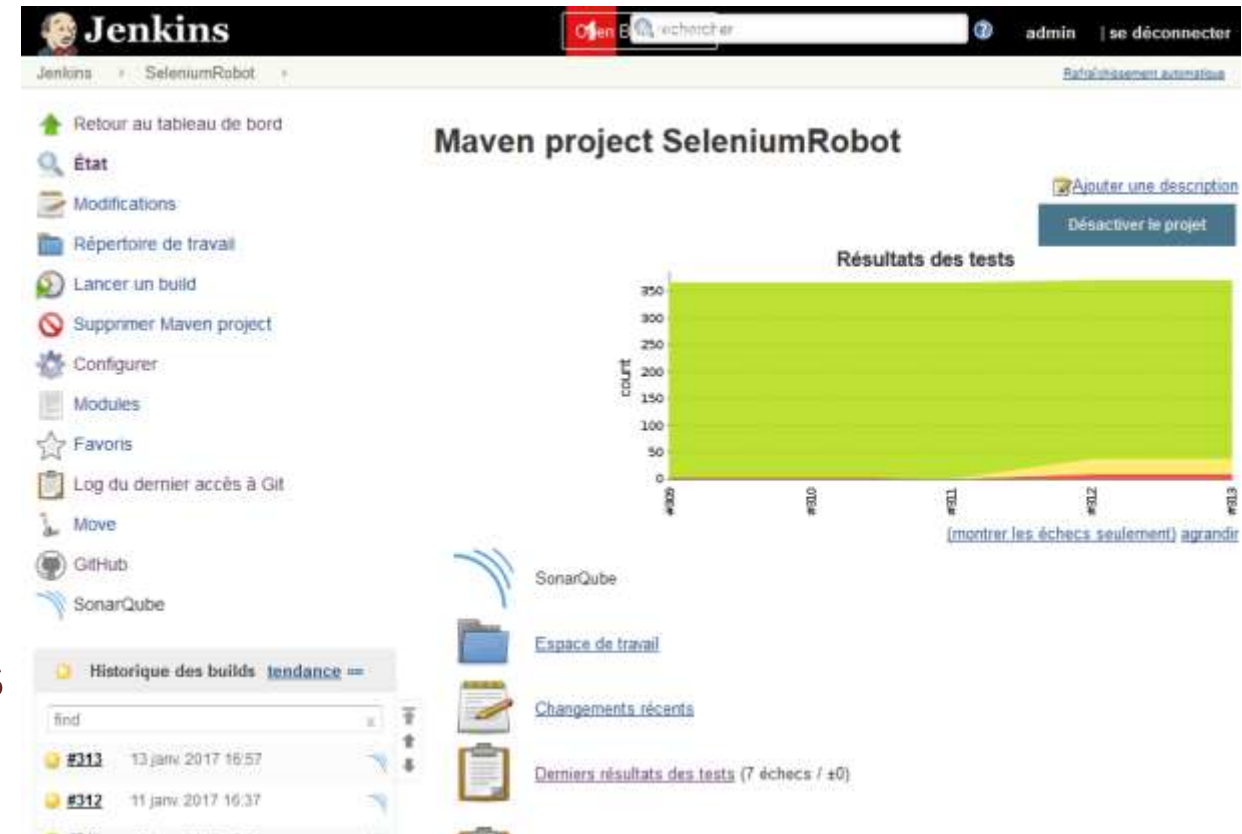




INTÉGRER LES TESTS ET LE CONTINUOUS DEPLOYMENT



- Ordonnanceur de jobs
- Un job peut être
 - Une construction
 - Un déploiement
 - Des tests
 - Tout cela à la fois
- Plusieurs jobs peuvent être enchaînés



File d'attente des constructions	
File d'attente des constructions vide	
État du lanceur de compilations	
maître	
1 Au repos	
2 Au repos	

Backup	4 mo. 2 j - #4	59 s	
envmonitor_install_int_pipe	1 mo. 1 j - #15	1 mn 36 s	
EnvMonitor_pipeline	1 mo. 1 j - #34	8 mn 28 s	
infolidays_android	7 mo. 6 j - #5	7 mo. 6 j - #2	1 mn 40 s
Jenkins_API	4 mo. 3 j - #12	5 mo. 2 j - #9	32 s
	3 mo. 27 j -		




Intégrer les tests et le Continuous Deployment

Orchestrer les tests avec le pipeline de build Jenkins

- Le pipeline de build est une bonne pratique pour le Continuous Deployment
 - C'est un enchaînement d'étapes
 - Meilleure lisibilité par rapport à des jobs classiques enchaînés
 - Possibilité de paralléliser les étapes
 - Possibilité de stocker le pipeline en gestion de configuration
- Les phases de tests sont intégrées

Pipeline JPetStore_pipeline

 [Ajouter une description](#)



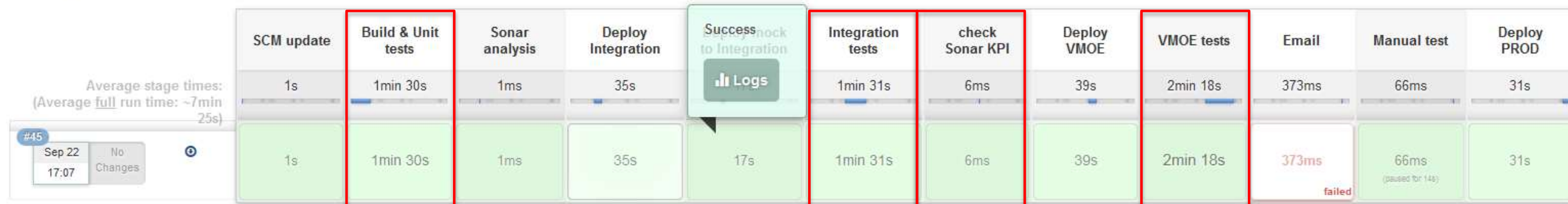
Last Successful Artifacts

 [jpetstore.war](#) 43,04 MB  [view](#)



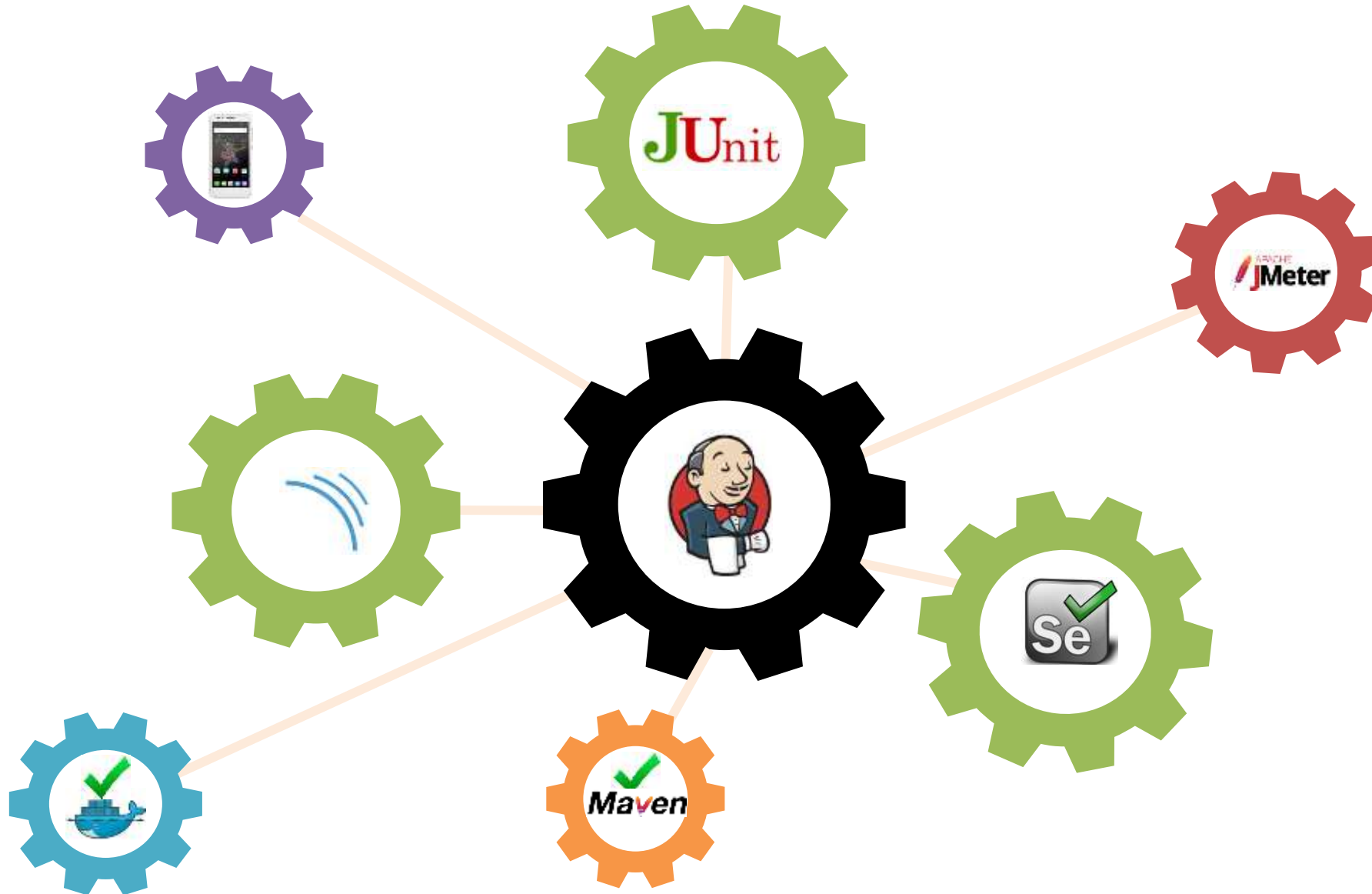
[Recent Changes](#)

Stage View



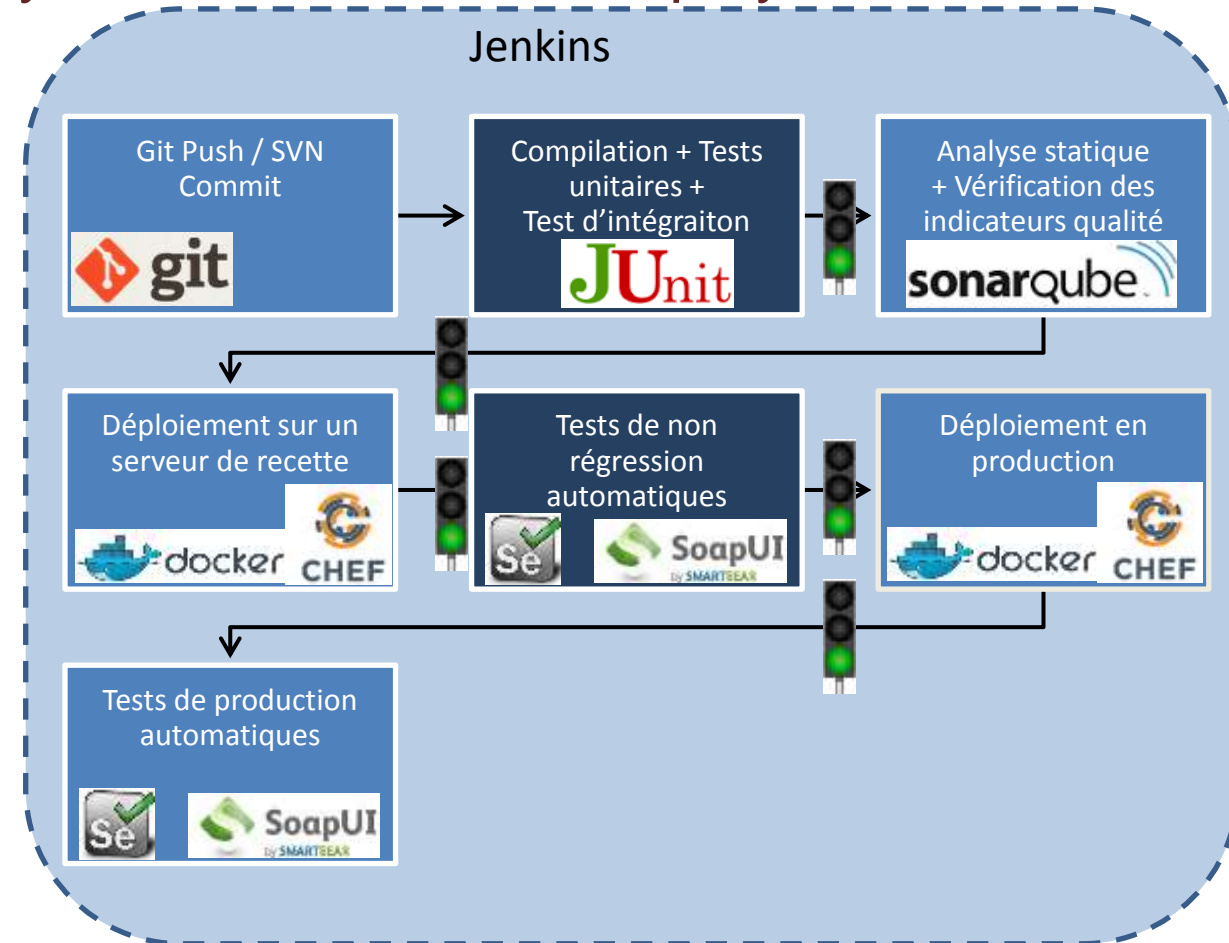
Intégrer les tests et le Continuous Deployment

Un pipeline de build peut embarquer tous les contrôles qualité





- Mettre en place du Continuous Delivery ou du Continuous Deployment réclame entre autres
 - Une grande maîtrise technique
 - Des équipes motivées et formées
- **Et surtout beaucoup de tests automatisés!**



Questions



bertrand.hecquet@infotel.com

