Industrialisation de la chaîne de production : validation, intégration, tests

De l'atelier de développement à l'usine logicielle

Thomas Lallart - INRA-DSI
ENVOL 2012 - Biarritz
21-25 janvier 2013

Industrialisation de la chaîne de production

1) Contexte
    1) Le coût des bugs et de la non-qualité
    2) Méthodes agiles
2) Usine logicielle
    1) Gestion de versions
    2) Gestion des dépendances
    3) Build
    4) Tests
    5) Intégration continue
    6) Inspection continue
    7) Livraison continue
    8) « Documentation continue »
3) Démo
4) Synthèse & retour d'expérience

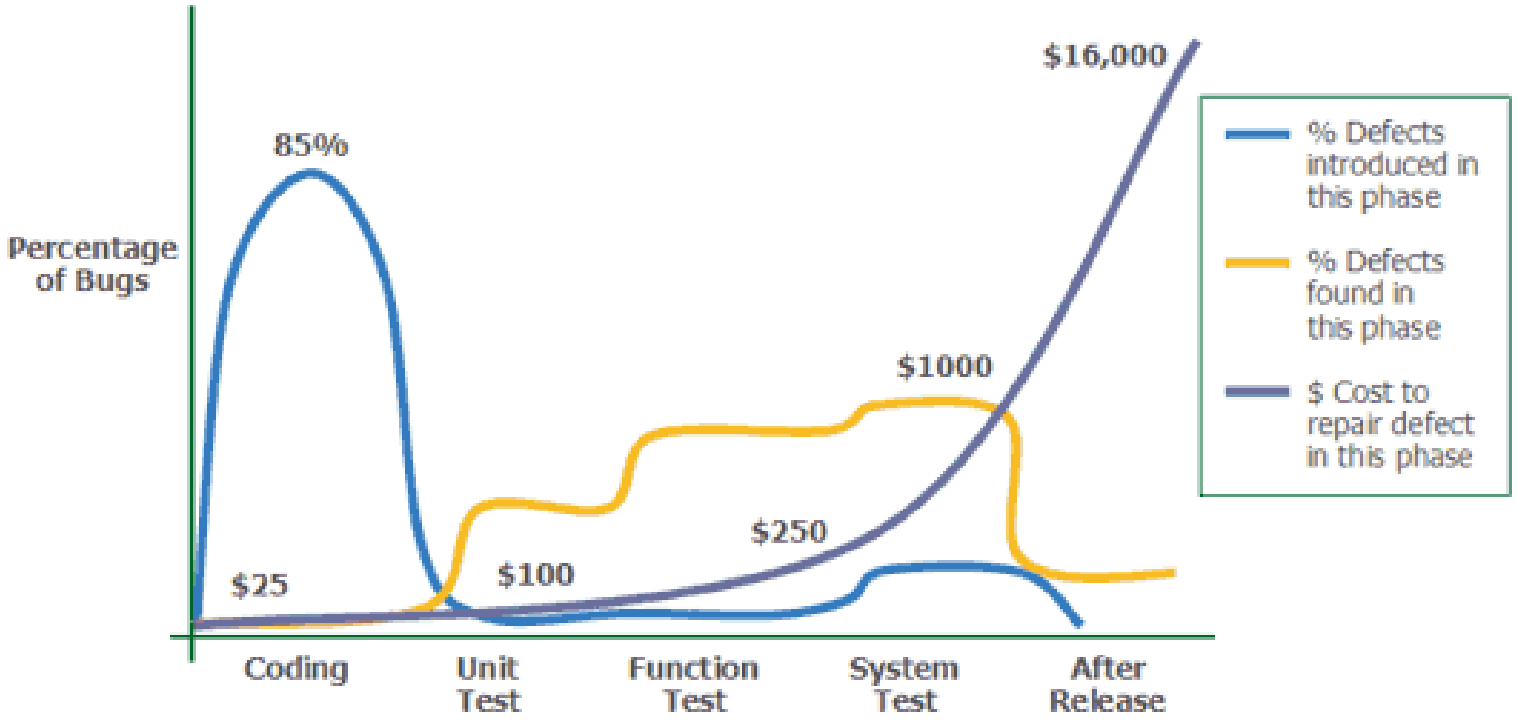
Industrialisation de la chaîne de production

- « Livrer, c'est coûteux, donc je préfère livrer rarement mais beaucoup d'un coup »
- « Tester souvent, c'est chronophage, donc pour gagner du temps on testera plus tard quand tout sera fini »
- « Pour vérifier la qualité, on a planifié une revue de code à la fin du projet »

Industrialisation de la chaîne de production

- « Livrer, c'est coûteux, donc je préfère livrer rarement mais beaucoup d'un coup »
- « Tester souvent, c'est chronophage, donc pour gagner du temps on testera plus tard quand tout sera fini »
- « Pour vérifier la qualité, on a planifié une revue de code à la fin du projet »

Coût de résolution d'un bug



Ref : http://yorktown.cbe.wwu.edu/sandvig/mis314/lectures/016_debugging.php

Fiabilité : le coût visible de la non-qualité

- Bug de l'assurance vieillesse : entre 1984 et 2009, 8m de salariés récupèrent un trimestre de trop sur leur pension.
- **Coût** (en perte brute) : 2,5 milliards €.
- **Cause** : la gestion des arrondis. « Selon le Code de la Sécu, un chômeur ayant été indemnisé pendant 50 jours par l'Unedic a droit à un trimestre de cotisations retraite auprès de la Cnav. Pour prétendre à un deuxième trimestre de cotisation, il lui faut au moins 100 jours d'indemnisation. S'il a été indemnisé 99 jours, il n'a droit qu'à un trimestre. C'est clair : on arrondit la durée d'indemnisation à la cinquantaine inférieure. Mais les informaticiens de l'Unedic implémentent, eux, qu'il faut arrondir à la cinquantaine supérieure. »
- Réf : http://www.impots-utiles.com/securite-sociale-un-bug-informatique-a-25-milliards-deuros/

Fiabilité : le coût visible de la non-qualité

- Crash de Mars Climate Orbiter : le sonde spatiale se crashe sur Mars au lieu d'entrer en orbite. Elle n'aura pris qu'une seule photo - vraisemblablement l'une des plus chères du monde - avant de s'écraser.
- **Coût** (matériel) : 900m $
- **Cause** : la différence de système métrique utilisé entre 2 modules dans l'expression d'une force de poussée - km et Newton (système métrique) pour l'un, miles et livres (système impérial) pour l'autre - a impliqué des erreurs de calcul dans la navigateur.
- Ref : http://www.nirgal.net/mco_end.html

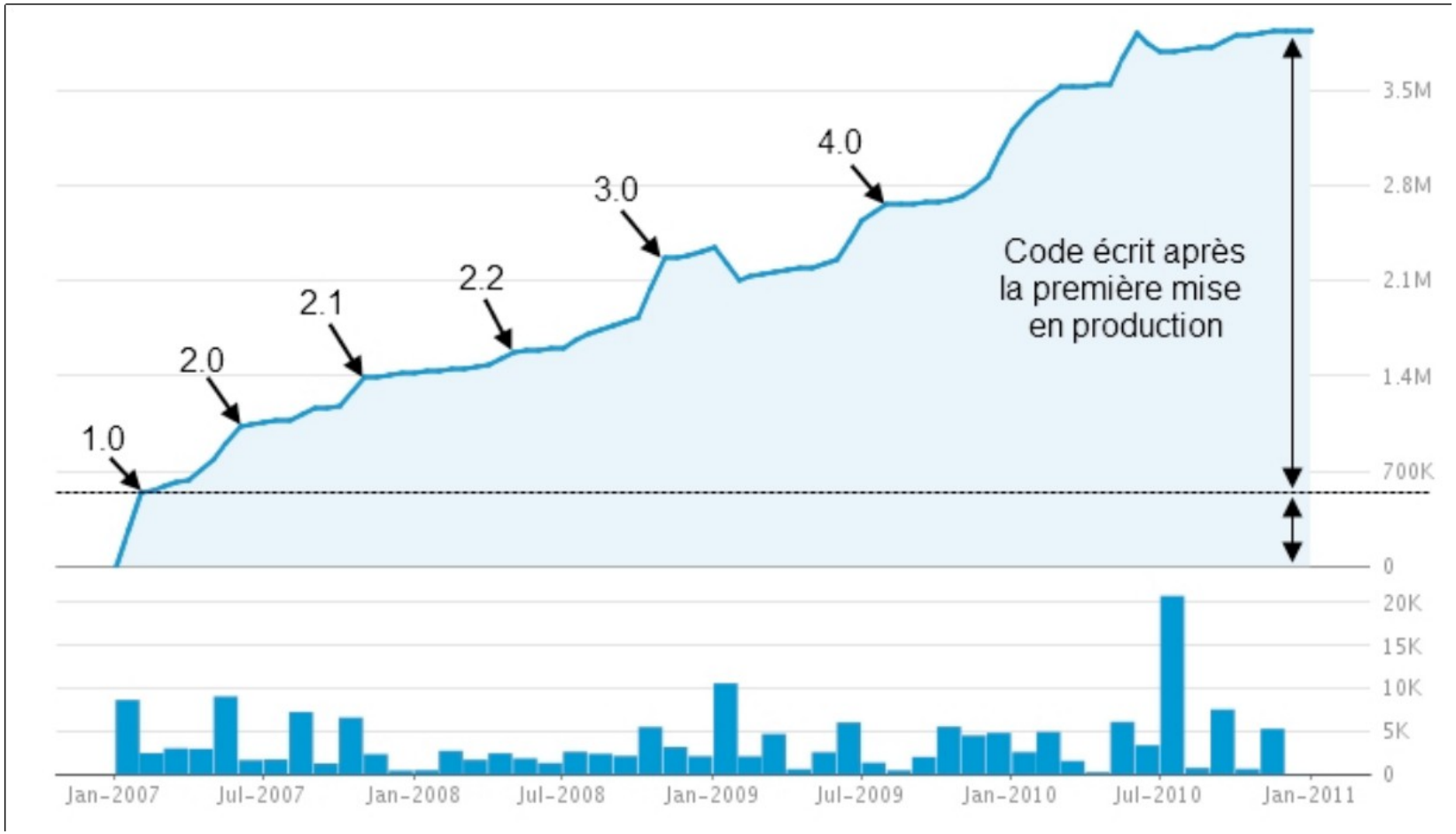
Fiabilité : le coût visible de la non-qualité

- 1er lancement d'Ariane 5, 1996 : la fusée dévie de sa trajectoire lors du décollage, puis explose en vol après 37s.
- **Coût** (matériel) : 370m $
- **Cause** : « l'exception logiciel [...] s'est produite pendant une conversion de données de représentation flottante à 64 bits en valeurs entières à 16 bits. Le nombre en représentation flottante [...] avait une valeur qui était supérieure à ce que pouvait exprimer un nombre entier à 16 bits. Il en est résulté une erreur d'opérande. Les instructions de conversion de données (en code Ada) n'étaient pas protégées contre le déclenchement d'une erreur d'opérande bien que d'autres conversions de variables comparables présentes à la même place dans le code aient été protégées. »
- Ref : http://www.astrosurf.com/luxorion/astronautique-accident-ariane-v501.htm

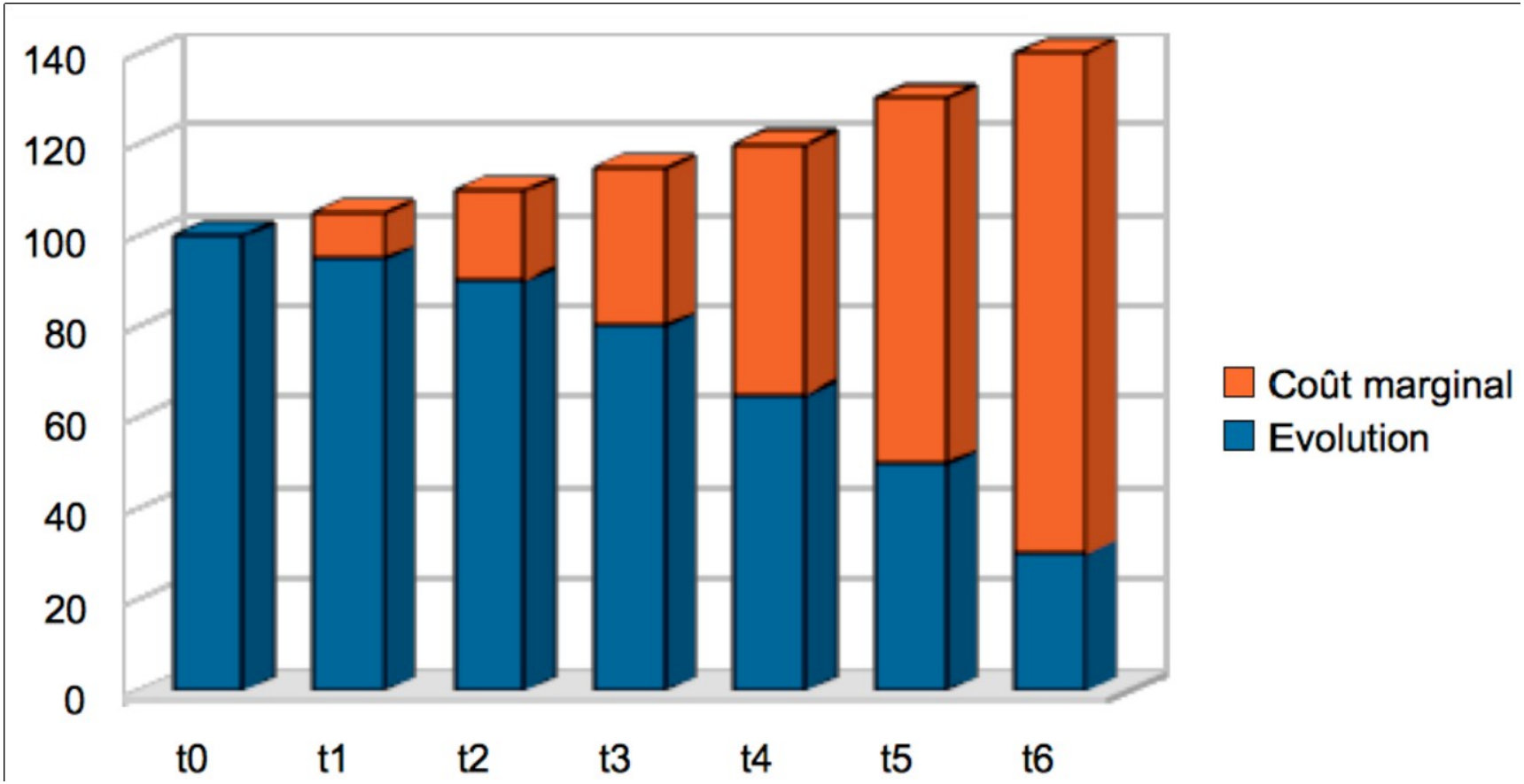
Maintenabilité : le coût caché de la non-qualité

- Maintenabilité : capacité à corriger, capacité à évoluer ou plus généralement « tolérance aux changements »
- 60 à 80 % des développements sont réalisés après la 1ère mise en production
- En moyenne, la durée de vie d'un logiciel est de 4-8 ans
- Le coût des évolutions est de plus en plus élevé au fil du temps
- Le coût de la non-qualité ne se mesure donc pas uniquement sur la phase de réalisation ou les anomalies à la mise en production mais bien sur l'ensemble de la vie du logiciel

Evolution du nombre de lignes de code dans le temps



Evolution du coût marginal dans le temps



Agilité

- Forte augmentation des développements en mode itératif et incrémental organisés en « mode agile »
- Chaque développement est vu comme une évolution de la version précédente
- Livraisons plus rapprochées et plus fréquentes (notions de sprint et d'itération)
- Mise en avant de certaines pratiques : TDD, Refactoring, Intégration Continue, Livraison Continue, Sotfware Craftmanshift
- Attention continue portée à la conception et à la qualité
- La plus haute priorité est la satisfaction utilisateur

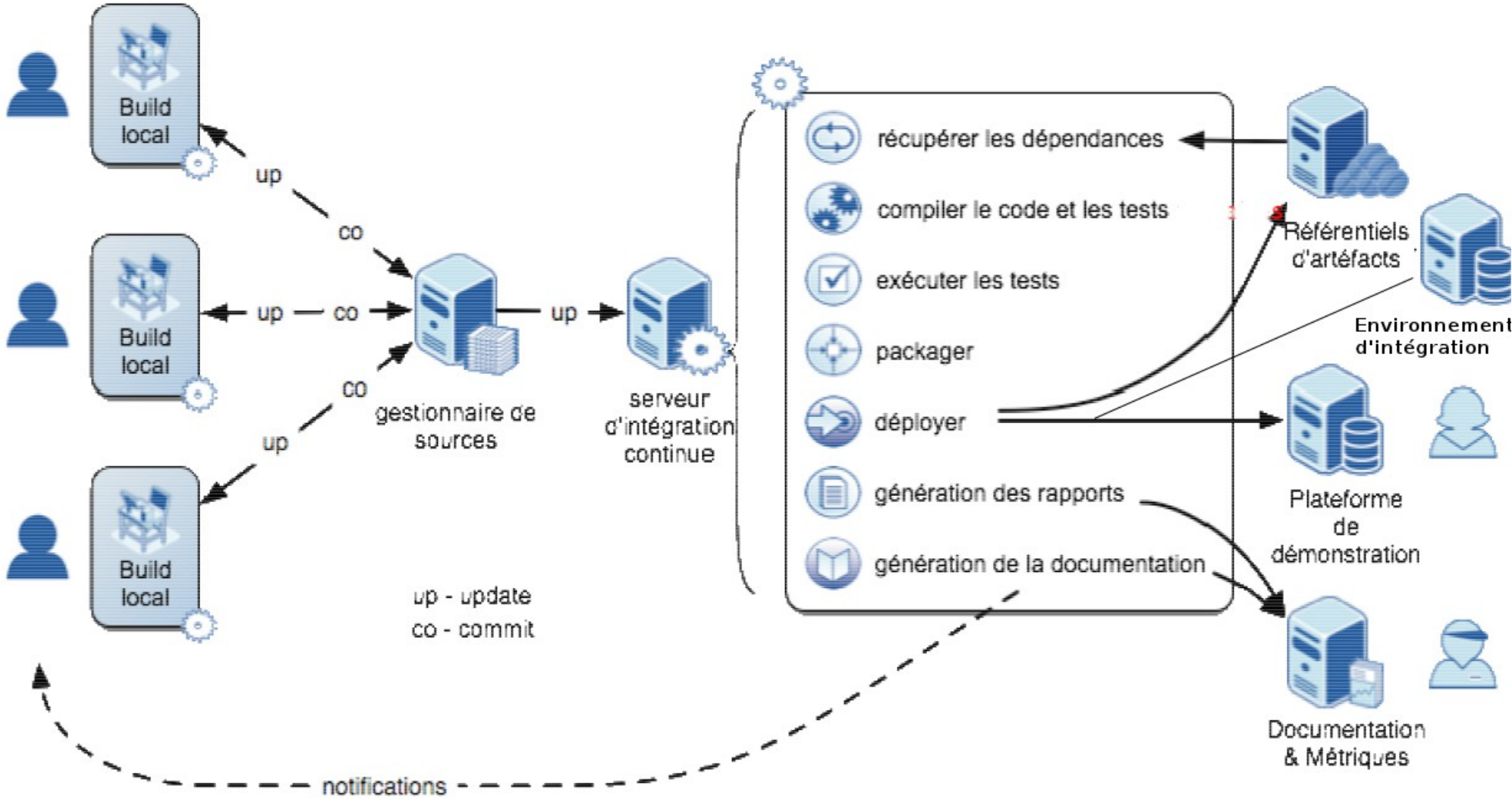
Objectifs

- Tester plus, livrer plus tôt, plus souvent et plus sereinement, réduire les anomalies, améliorer la qualité
- Cela passe par :
  - Maîtriser les développements
  - Maîtriser les tests
  - Maîtriser les versions
  - Maîtriser les livraisons
  - Maîtriser la qualité
- … par des pratiques et l'automatisation des pratiques

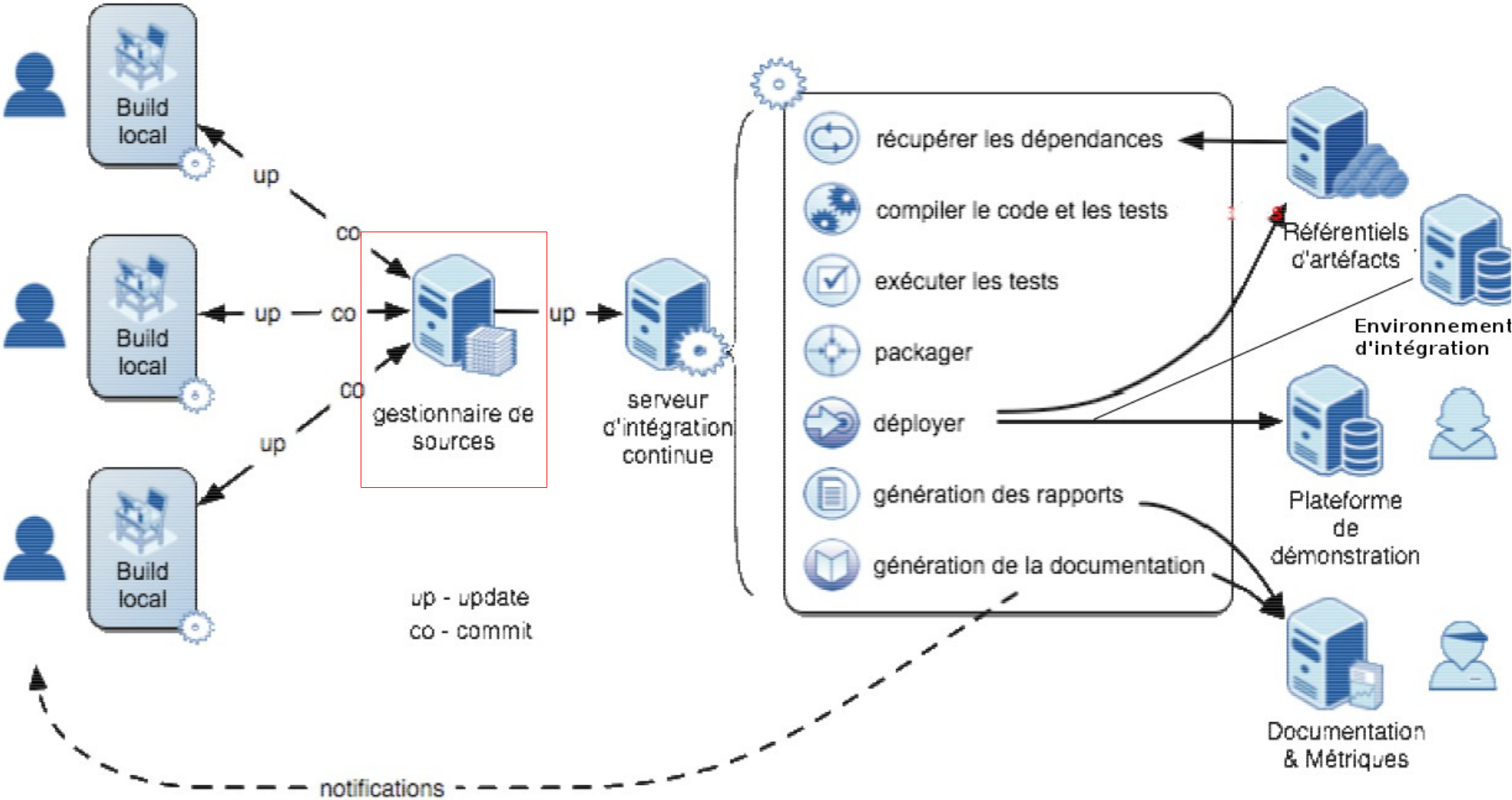
(Une) Définition

L'usine logicielle gère la fabrication (au sens large) du produit ; l'organisation y est découpée comme une chaîne de production où les tâches répétitives seront automatisées comme le lancement routinier de la compilation, l'exécution des tests unitaires (et des autres types de tests), le déploiement.

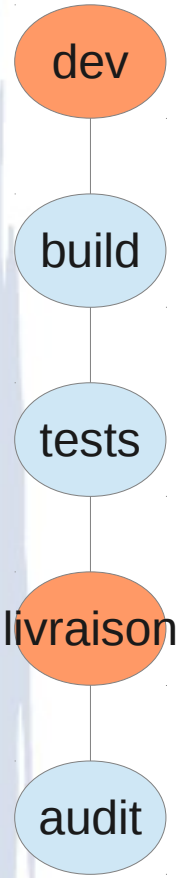
Cible

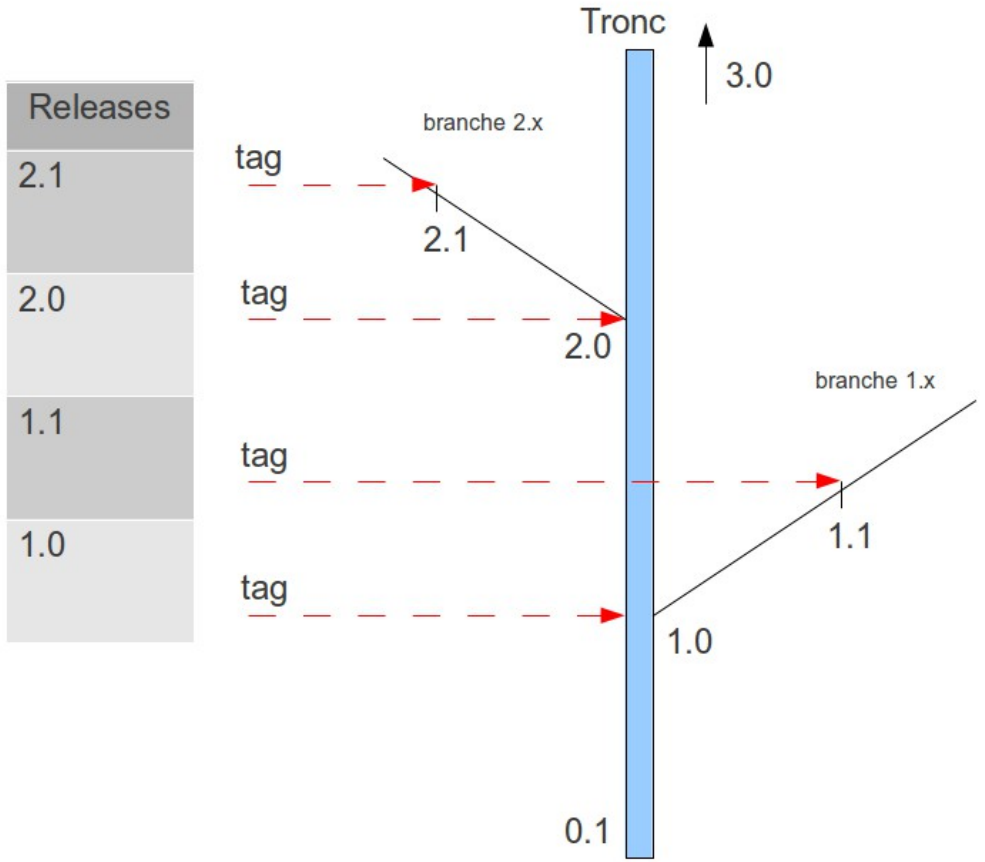

Build local
Build local
Build local
up
co
up
co
co
up
gestionnaire de sources
up
serveur d'intégration continue
récupérer les dépendances
compiler le code et les tests
exécuter les tests
packager
déployer
génération des rapports
génération de la documentation
Référentiels d'artéfacts
Environnement d'intégration
Plateforme de démonstration
Documentation & Métriques
up - update
co - commit
notifications

Gestion des versions

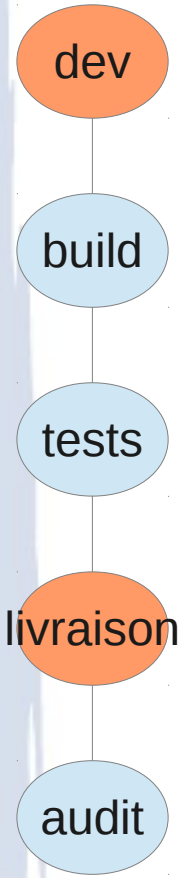


Gestion des releases (par branche corrective)

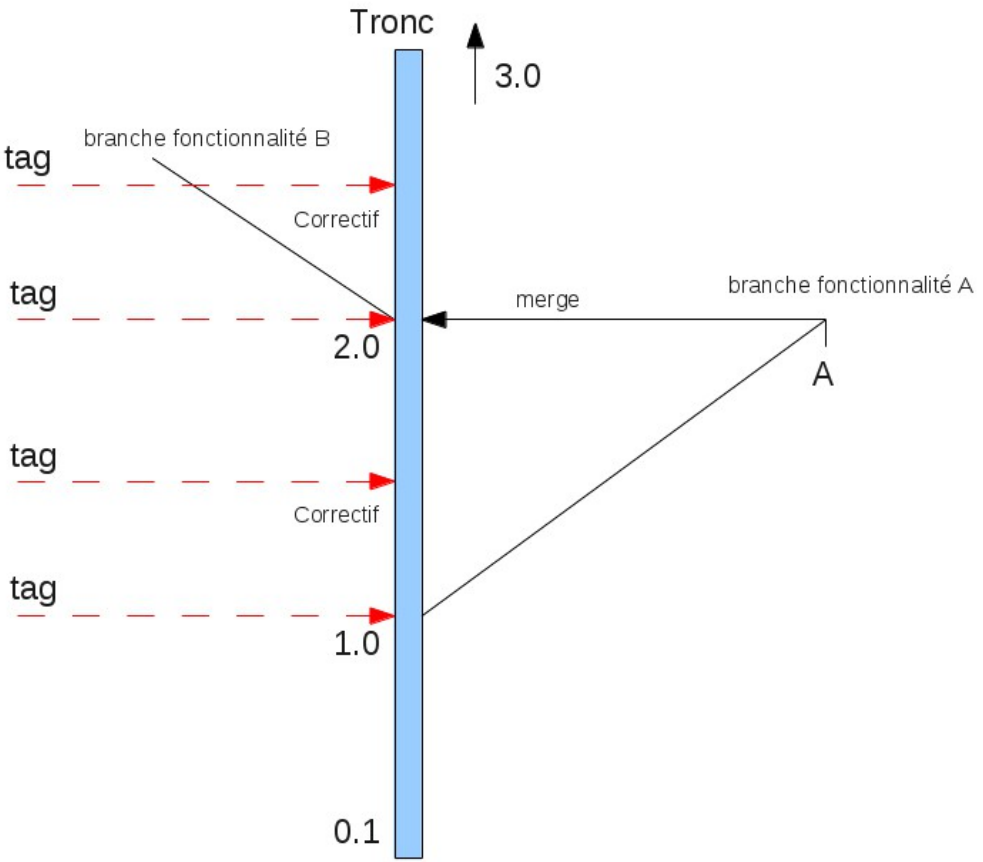




Gestion des releases (par branche de fonctions)



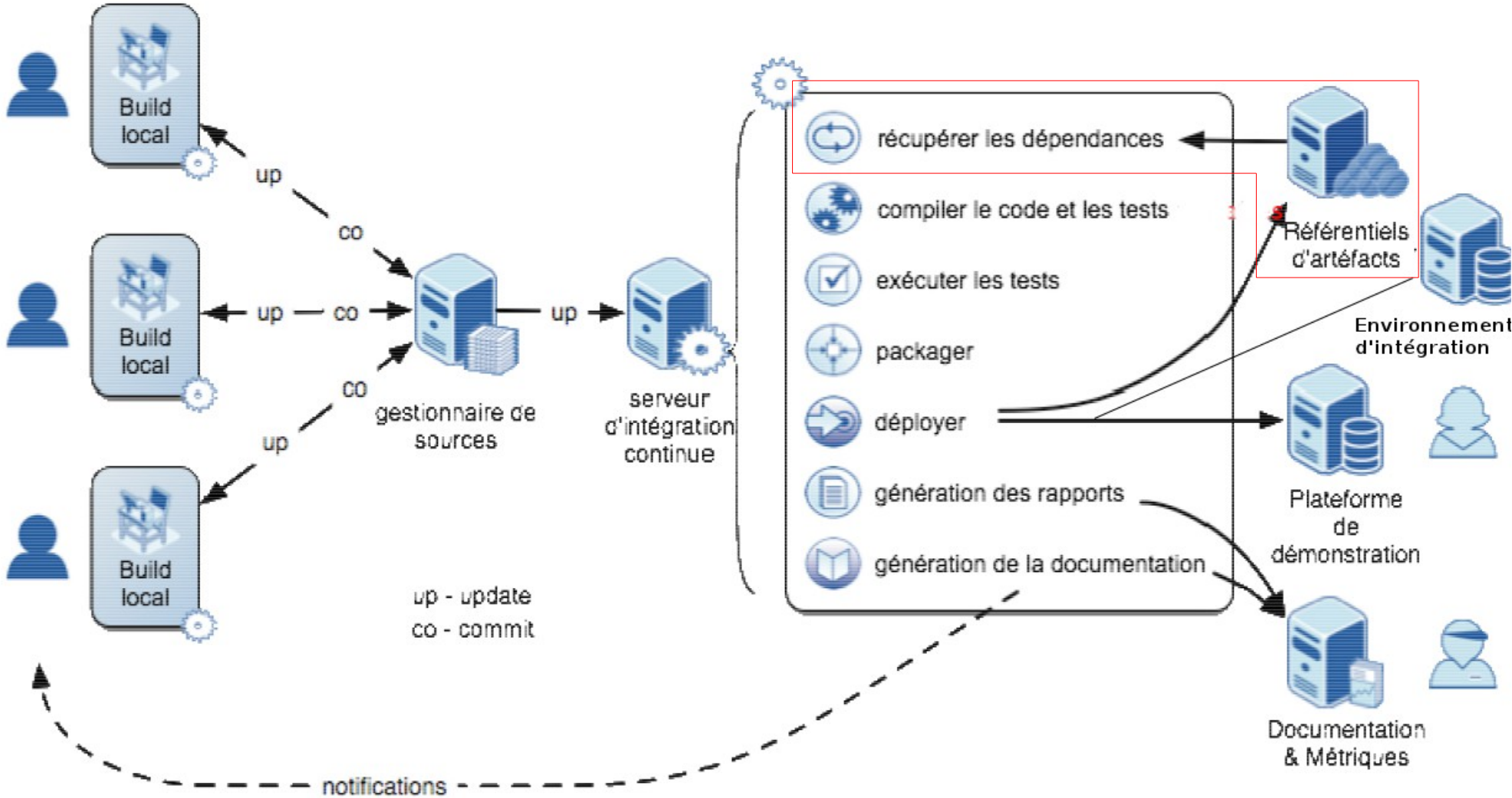
Releases
2.1
2.0
1.1
1.0



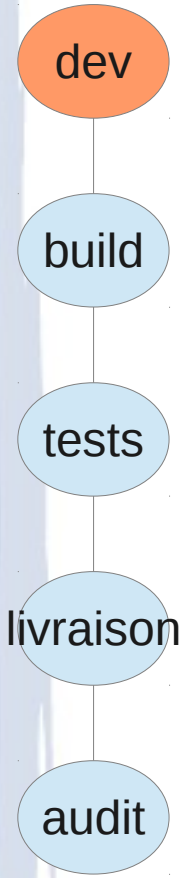
Gestions de version (pratiques)

- Toujours ajouter un message de commit
- Toujours exécuter Update avant Commit
- S'assurer que le projet compile (« builde ») avant commit
- Exécuter des petits commit fréquemment (au moins une fois par jour)
- Intégrer les concepts de release (version figée / taguée) et de snapshots (version courante en cours de développement)

Gestion des dépendances



Industrialisation des développements : référentiel des dépendances et des artefacts



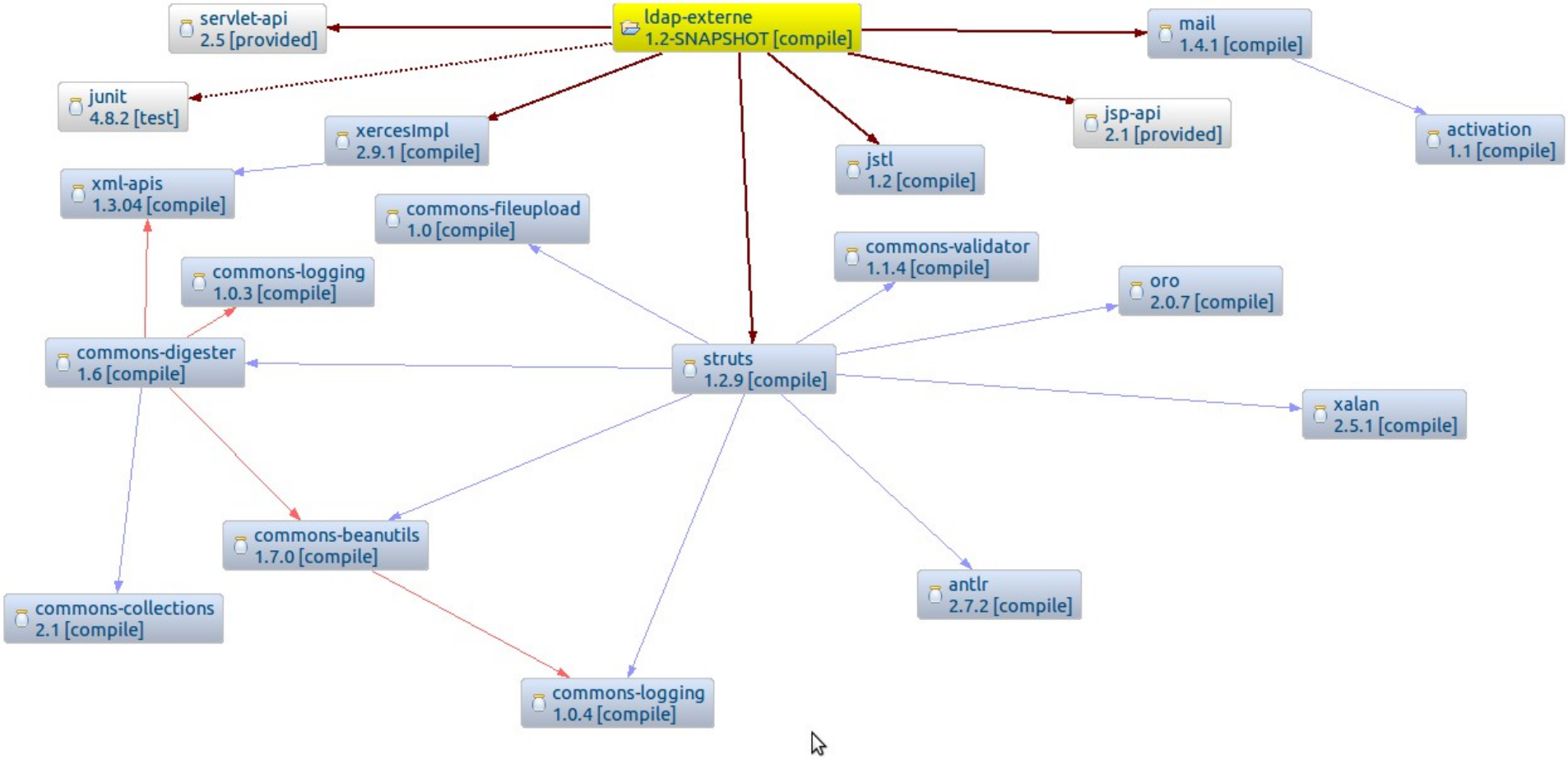
Problématique :

- Comment gérer « l'application A v1.1 dépend de la librairie B v3.2 et la librairie C v2.6 » ?
- Comment gérer « l'application A v1.2 dépendra de la librairie B v3.4, la librairie C v2.7 et la librairie D v1.0 qui elle-même dépend de la librairie E v2.1» ?
- Comment archiver l'ensemble des librairies utilisées/utilisables dans les applications ?

Solution : référentiel de librairies (artefacts)

- Centralisation sur plusieurs référentiels distants (repository) des archives des librairies
- Description des dépendances dans le projet

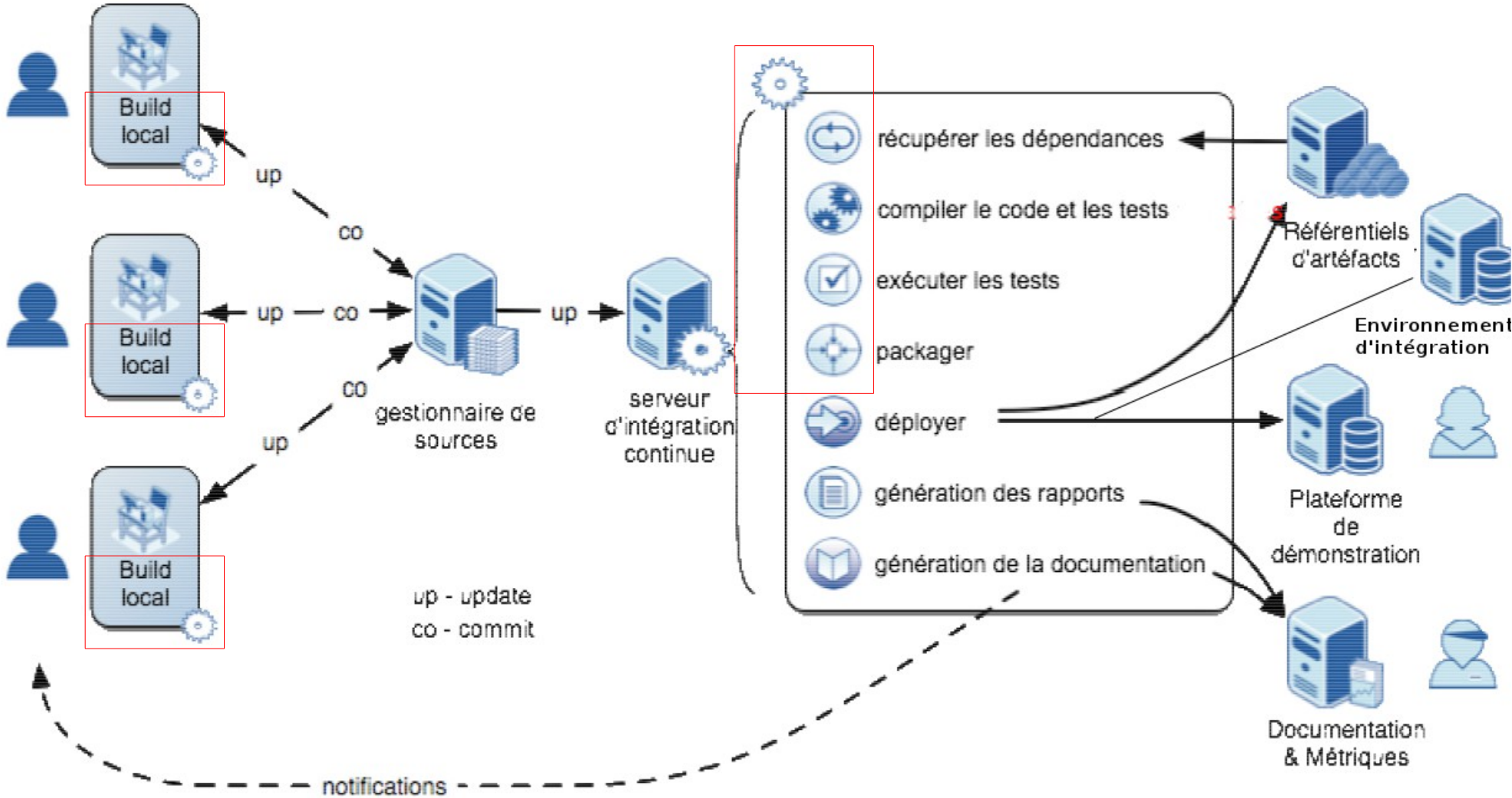
Exemple de graphe de dépendances



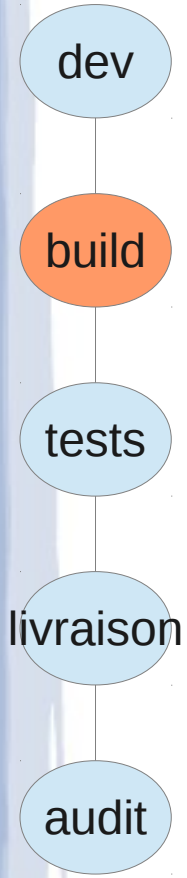
Pratiques

- Archiver les releases et les snapshots
- Utiliser au maximum les repositories publics
- Eviter d'intégrer manuellement des artefacts dans le repository

Automatisation du build



standardisation de la fabrication (build)



- Problématique :
  - La fabrication d'un livrable peut être un processus complexe (compilation, assemblage, configuration, environnement local,...)
  - La gestion manuelle de la fabrication pour chaque application et pour chaque environnement représente un coût en temps et un risque d'erreur très important...
  - … d'autant plus si chaque projet a une façon différente de fabriquer le produit logiciel.
- Solution : automatisation de la fabrication
  - Gestion de tout le cycle de construction (dépendances, ressources, compilation, tests, packaging)
  - Procédure identique et automatisable pour chaque projet indépendante de l'environnement local

Exemple : cycle de build Maven

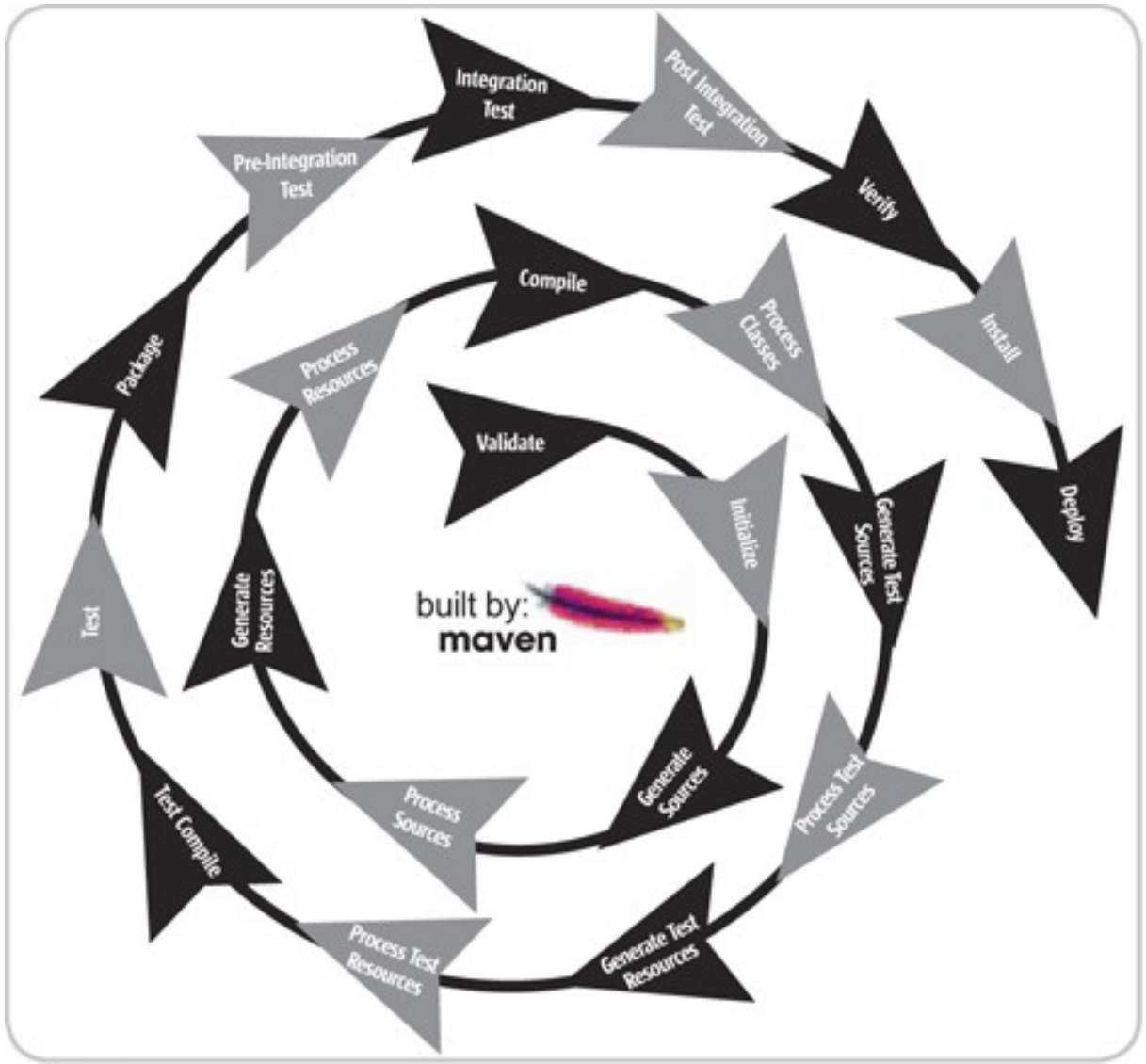
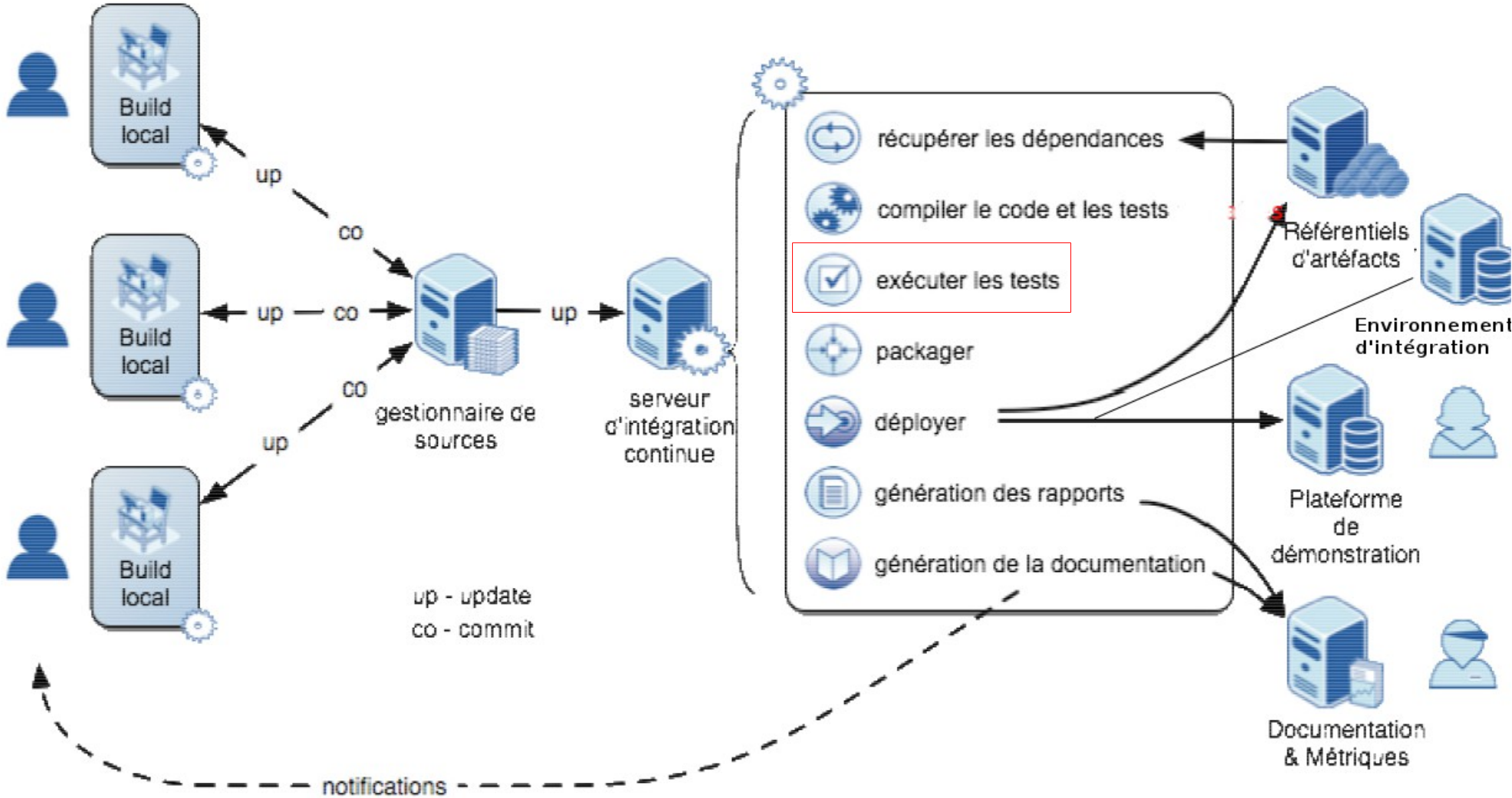
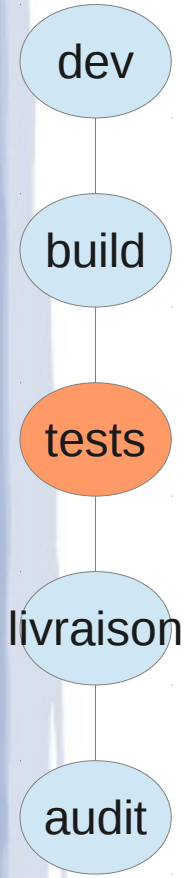


Figure 1

Tests

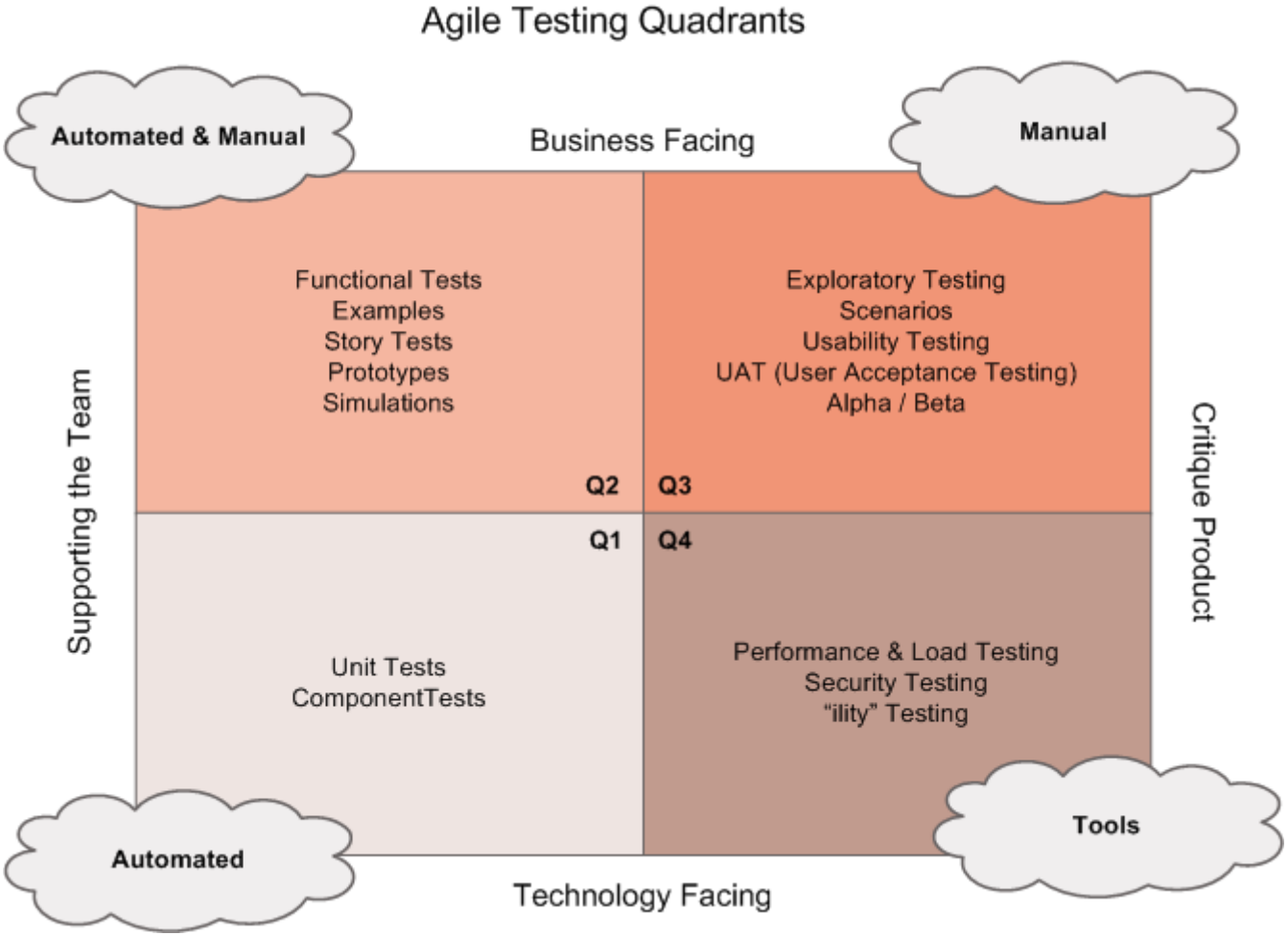

Build local
Build local
Build local
up
co
up
co
co
up
gestionnaire de sources
up
serveur d'intégration continue
récupérer les dépendances
compiler le code et les tests
exécuter les tests
packager
déployer
génération des rapports
génération de la documentation
Référentiels d'artéfacts
Environnement d'intégration
Plateforme de démonstration
Documentation & Métriques
up - update
co - commit
notifications

Industrialisation des développements : tests



- Problématique :
  - Tester le logiciel le plus tôt possible
  - « Valider » que le code écrit répond bien à la fonctionnalité prévue
  - Limiter les régressions
- Solution : écrire des tests automatisables
  - Tests unitaires : teste une fonction de manière isolée
  - Tests d'intégration : teste une fonction de manière intégrée avec tout ou parti de ses dépendances
  - Tests de performance
  - Tests IHM
  - Test d'acceptation/acceptance/vérification

Typologie des tests



Ref : http://agile2009.blogspot.fr/2009/08/agile-testing-quadrants.html

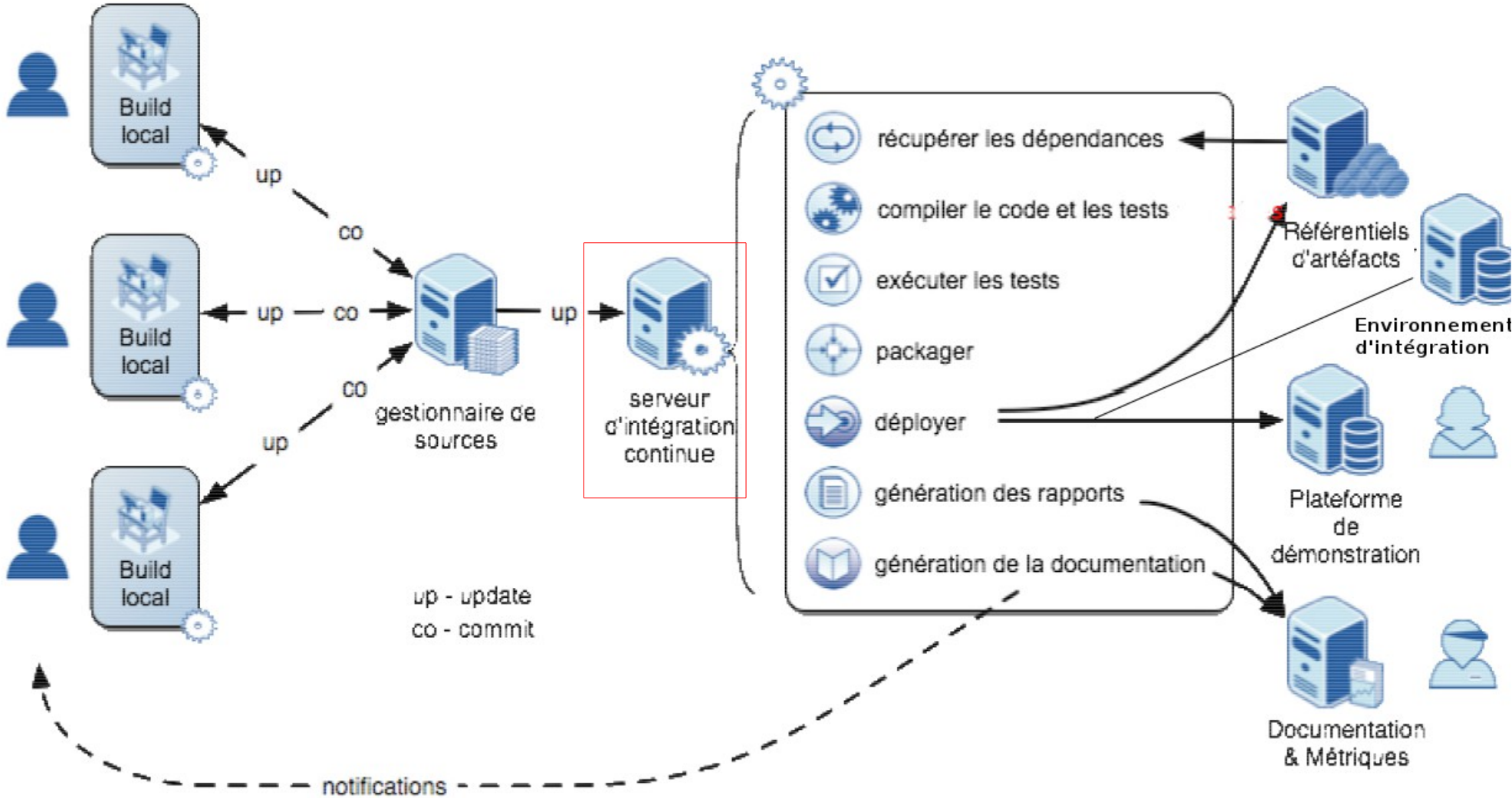
Typologie des tests

- Boite blanche
  - Unitaires
  - Intégration
- Boite noire
  - Intégration / Système
  - Performances (unitaires et en charge)
  - Fonctionnels / acceptance
  - IHM
  - Robustesse (soak)
  - Sécurité

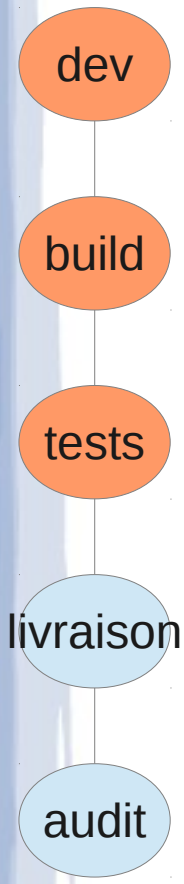
Tests : pratiques

- Adapter les types de tests au contexte du projet
- Une correction d'anomalie doit être l'occasion de créer (au moins) un test ; il sert à prouver l'anomalie et permet d'améliorer la couverture des tests
- Test Driven Development
- Les tests dépendant de données doivent être écrits sur la base d'un jeu de données réduit mais stable et représentatif de la production
- Si on ne pilote pas par les tests, écrire essentiellement les tests qui sur les portions de code qui portent le plus de sens (ex. sur les API, sur la partie métier/gestion)
- Les tests sont soumis au même respect des règles de codage, d'écriture, de conception et de refactoring que le code principal.
- Veiller à ce que le coût de maintenance des tests n'excède pas le coût de maintenance du code principal

Intégration continue



Intégration continue

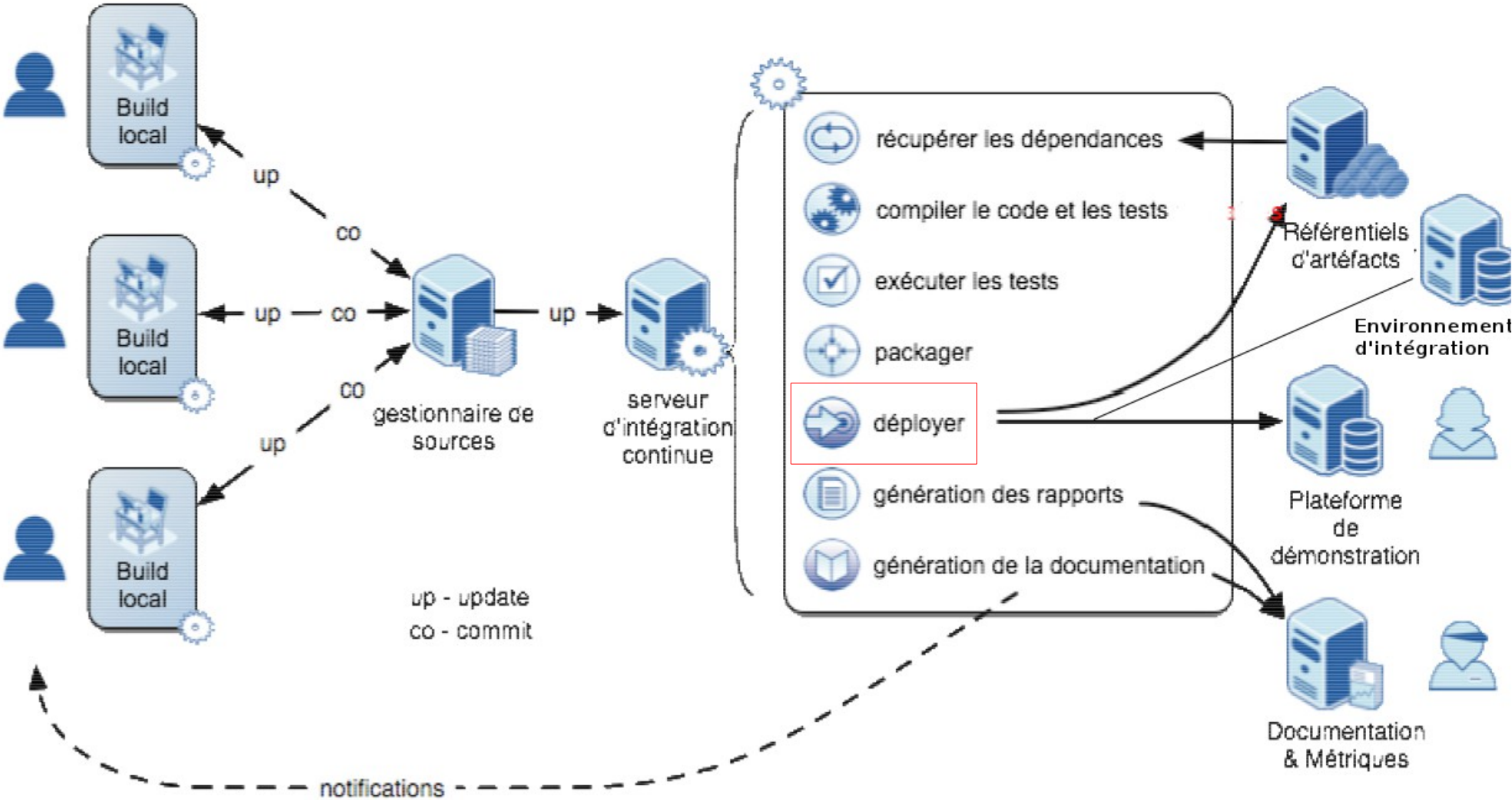


- Problématique :
  - Comment garantir que le projet « builde » toujours ?
  - Comment garantir que les tests « passent » toujours ?
  - Comment garantir que le projet s'installe toujours ?
- Solution : l'intégration continue
  - Exécution de l'ensemble du processus de fabrication à chaque modification dans la base de code
  - Retour immédiat au développeur sur les modifications qu'il a apportées
  - Ouvre la possibilité de déclencher ou d'ordonnancer toute opération déjà automatisée

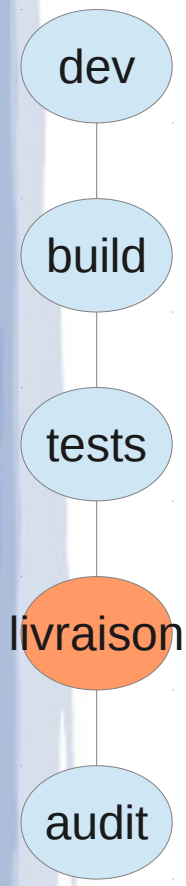
Intégration continue (pratiques)

- Maintain a Single Source Repository
- Automate the Build
- Make Your Build Self-Testing
- Everyone Commits To the Mainline Every Day
- Every Commit Should Build the Mainline on an Integration Machine
- Keep the Build Fast
- Test in a Clone of the Production Environment
- Make it Easy for Anyone to Get the Latest Executable
- Everyone can see what's happening
- Automate Deployment

Déploiement automatisé



Livraison continue

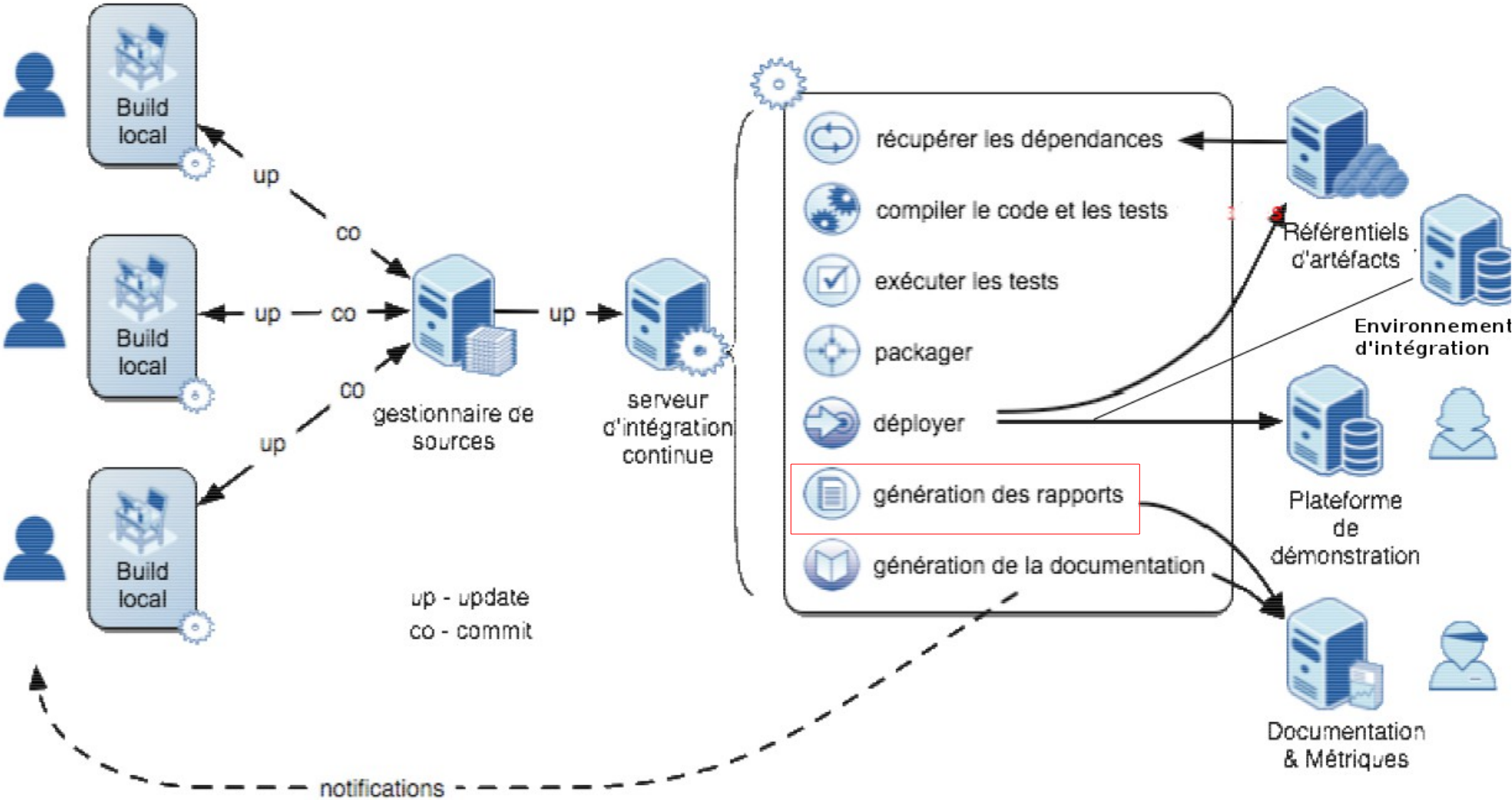


- Problématique :
  - La livraison d'un produit logiciel sur un environnement peut être complexe.
  - Peut nécessiter une édition de fichiers de paramétrage
  - Peut être une source d'erreur (erreur de versions, d'environnements, de configuration)
- Solution : déploiement automatisé
  - Déploiement du tronc de développement ou de n'importe quelle release
  - Sécurise et simplifie les livraisons
  - Facile à mettre en place sur des projets simples

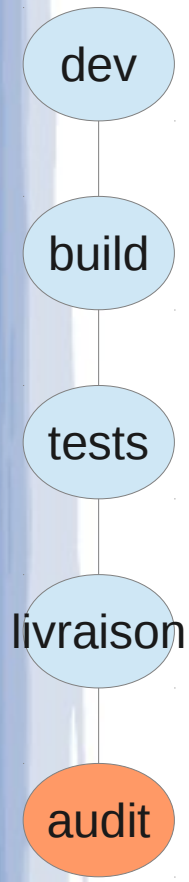
Livraison continue (pratiques)

- Le script (ou l'outil) effectuant la livraison doit être capable d'installer « par dessus » la version précédente (parfois délicat s'il y a des mises à jour du schéma BD, de l'arborescence, de données initiales, de la configuration)
- Le script (ou l'outil) devrait prévoir le « rollback » de la version précédente en cas de problème

Inspection continue

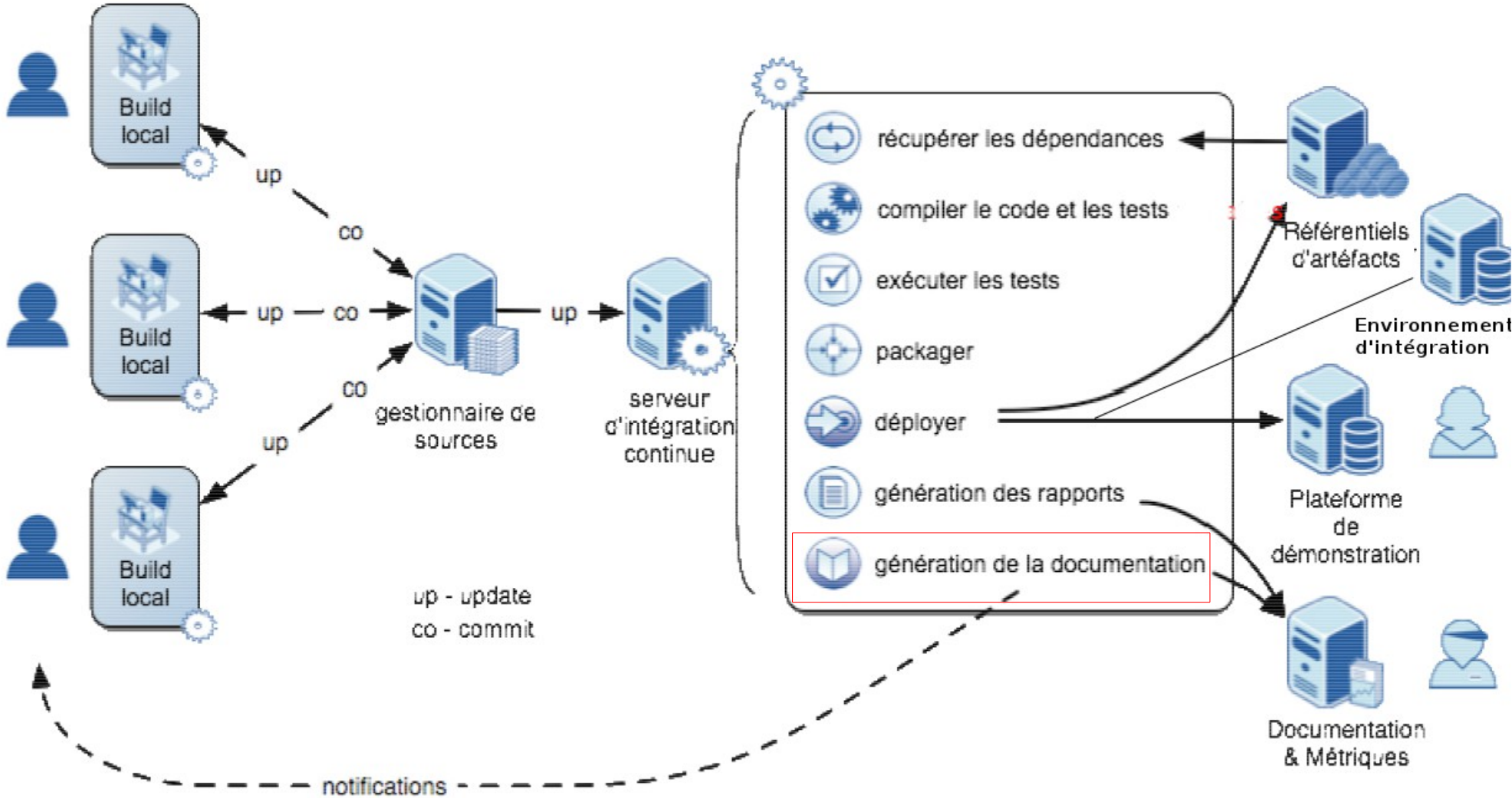


Inspection continue

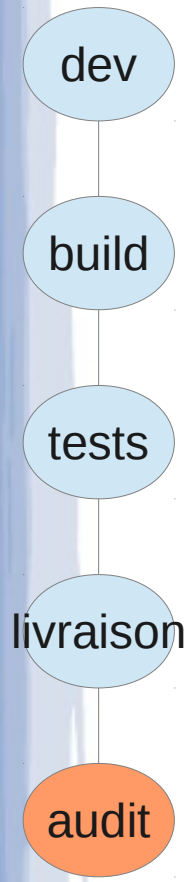


- Problématique :
  - Mesurer la qualité du code développé.
  - Analyser les risques associés à la manière dont le code est écrit
  - Obtenir les métriques sur l'évolutivité, la cohérence et le respect des normes du code produit
- Solution : audit automatisé de code
  - Fournit un rapport complet d'analyse statique du code
  - Fournit des indicateurs sur la santé d'un développement (dette technique)
  - Aide au reporting de la production informatique
  - Aide au développeur pour la qualité de son code et la détection de bugs
  - Aide au CP pour piloter la qualité

Documentation



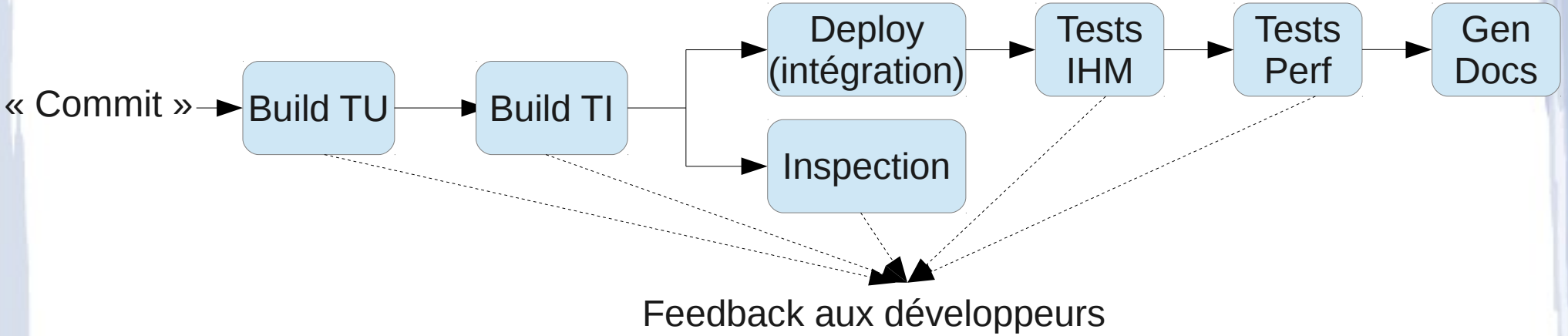
« Documentation continue »

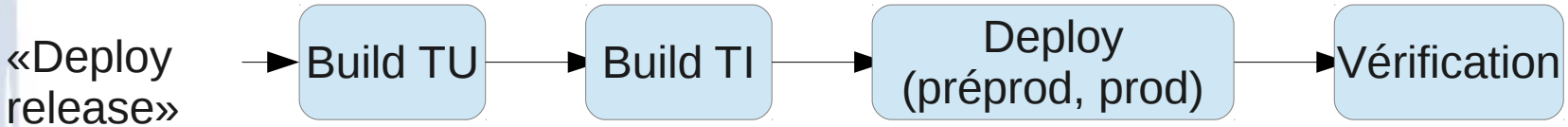


- Problématique :
  - Comment avoir une documentation toujours à jour ?
- Solution : génération automatisé de la documentation
  - Génération puis publication automatisée de la Javadoc, PHPDoc, …
  - Compléments de documentation par exemple par Doxygen

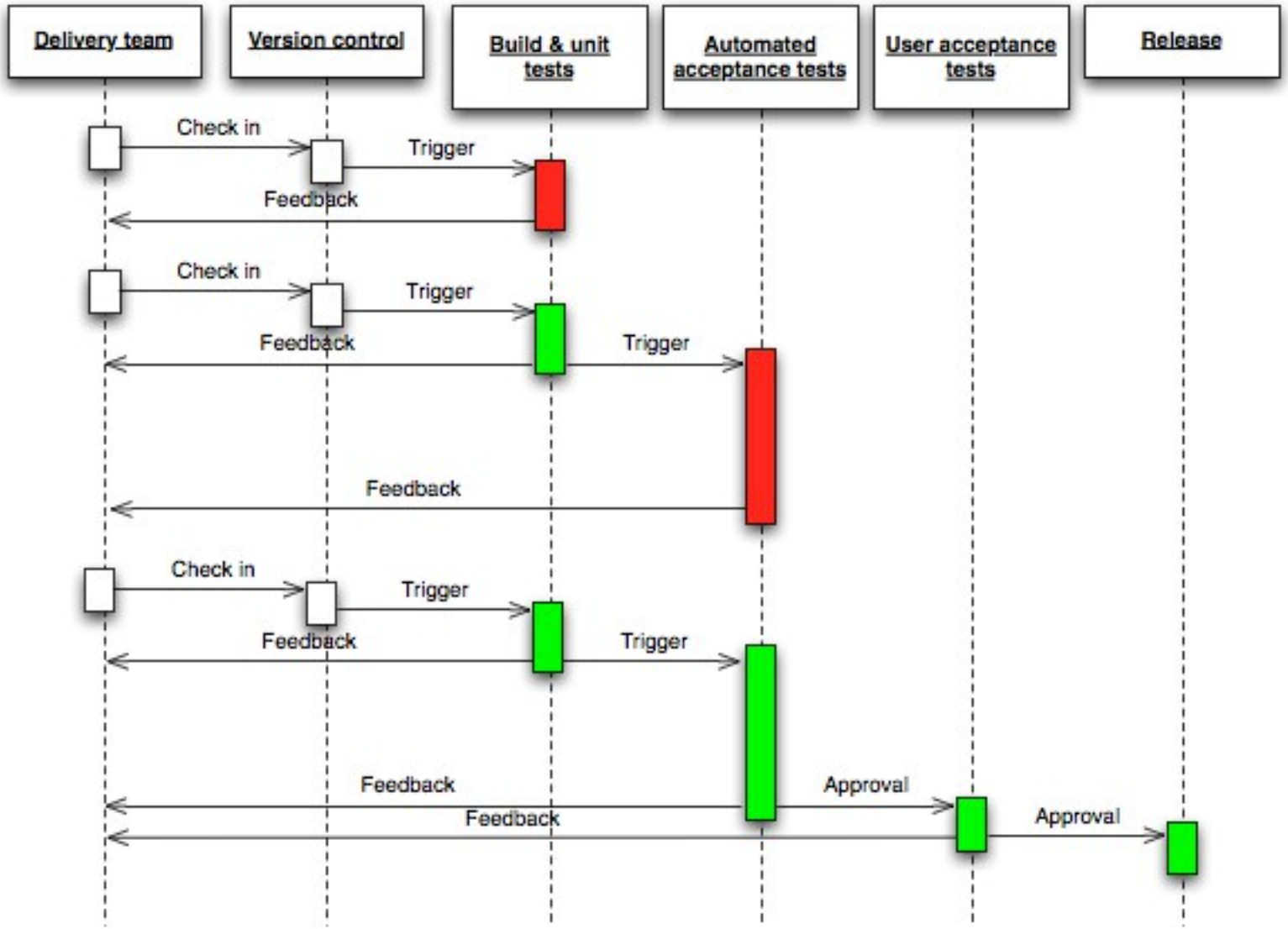
Pipeline

- Exemples de pipeline





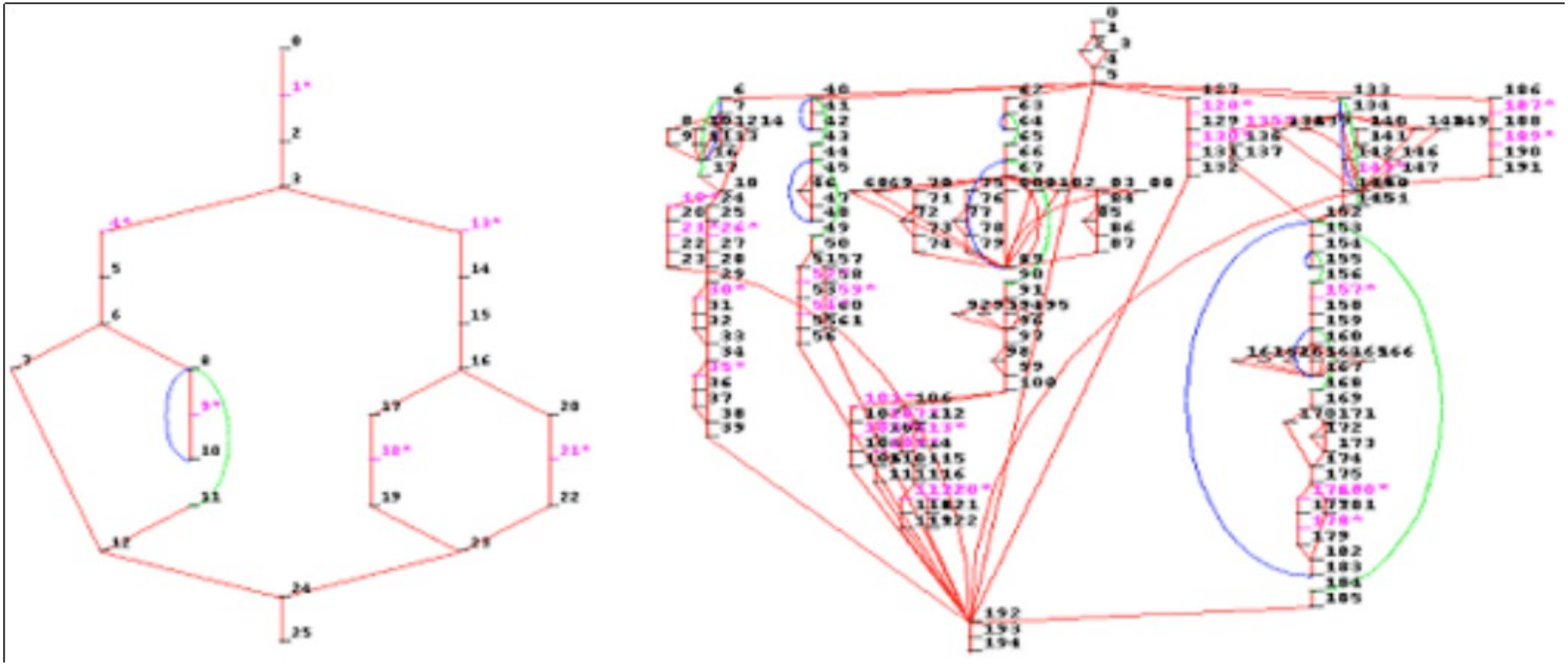
Pipeline de déploiement continu



Indicateurs qualité

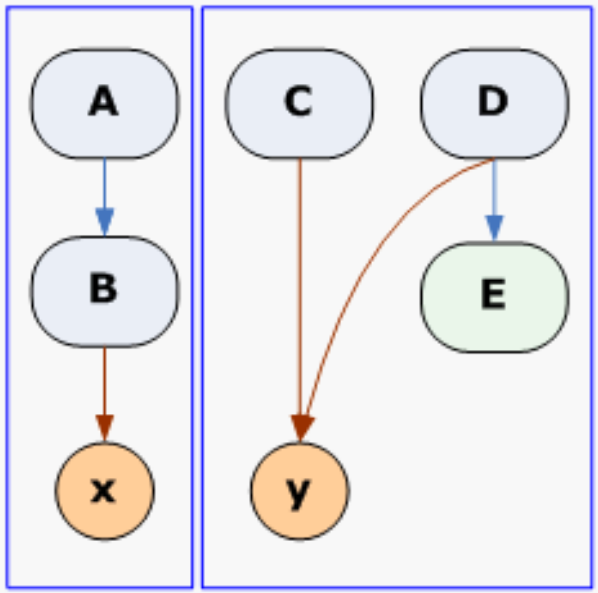
- Plusieurs indicateurs qualité « directs » :
  - Duplications
  - Couvertures des tests
  - Commentaires
  - Respect des règles de codage
- D'autres plus orientés conception
  - Complexité cyclomatique
  - LCOM4
  - Cycles
- Agrégation des métriques :
  - Dette technique, SQI

Compléxité cyclomatique



Ref : http://www.mccabe.com/

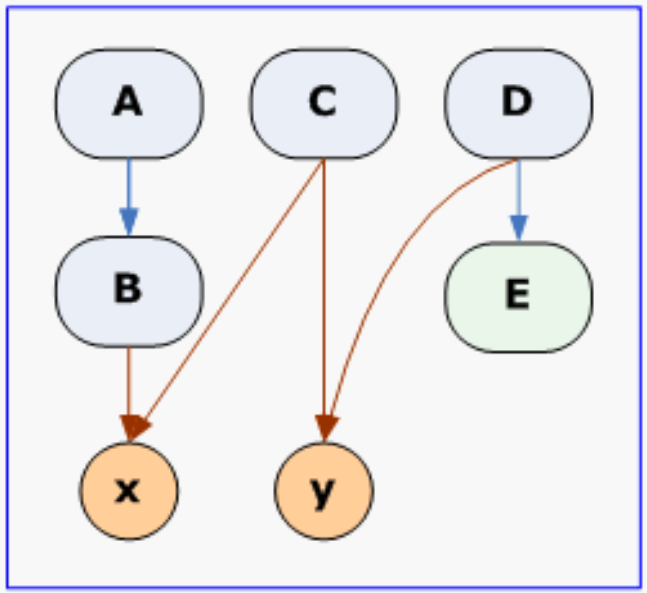
LCOM4



LCOM4 = 2

The example on the left shows a class consisting of methods A through E and variables x and y. A calls B and B accesses x. Both C and D access y. D calls E, but E doesn't access any variables.

This class consists of 2 unrelated components (LCOM4=2). You could split it as {A, B, x} and {C, D, E, y}.



LCOM4 = 1

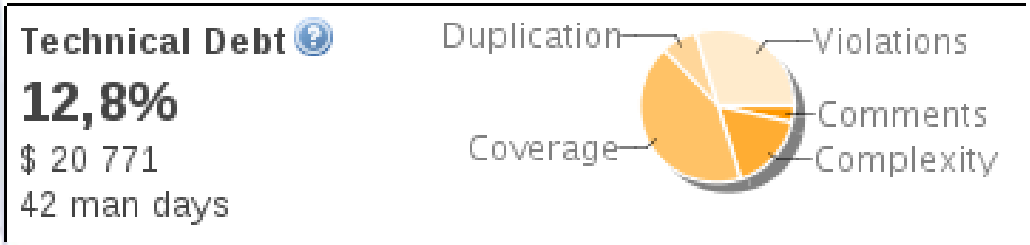
In the example on the right, we made C access x to increase cohesion.

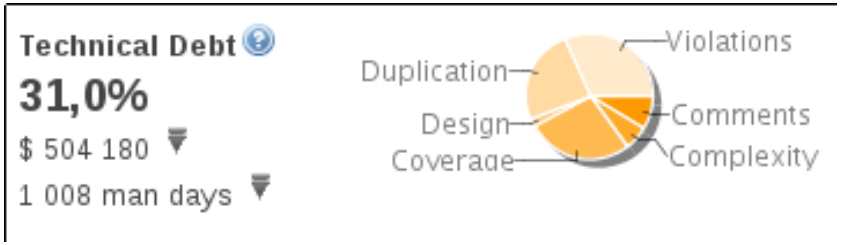
Now the class consists of a single component (LCOM4=1). It is a cohesive class.

Ref : http://www.aivosto.com/project/help/pm-oo-cohesion.html#LCOM4

Dette technique

- Notion introduite par Ward Cunningham
- Comparable à un emprunt dont on paie les intérêts tant qu'elle n'est pas remboursée
- « Qui paye ses dettes s'enrichit ! »





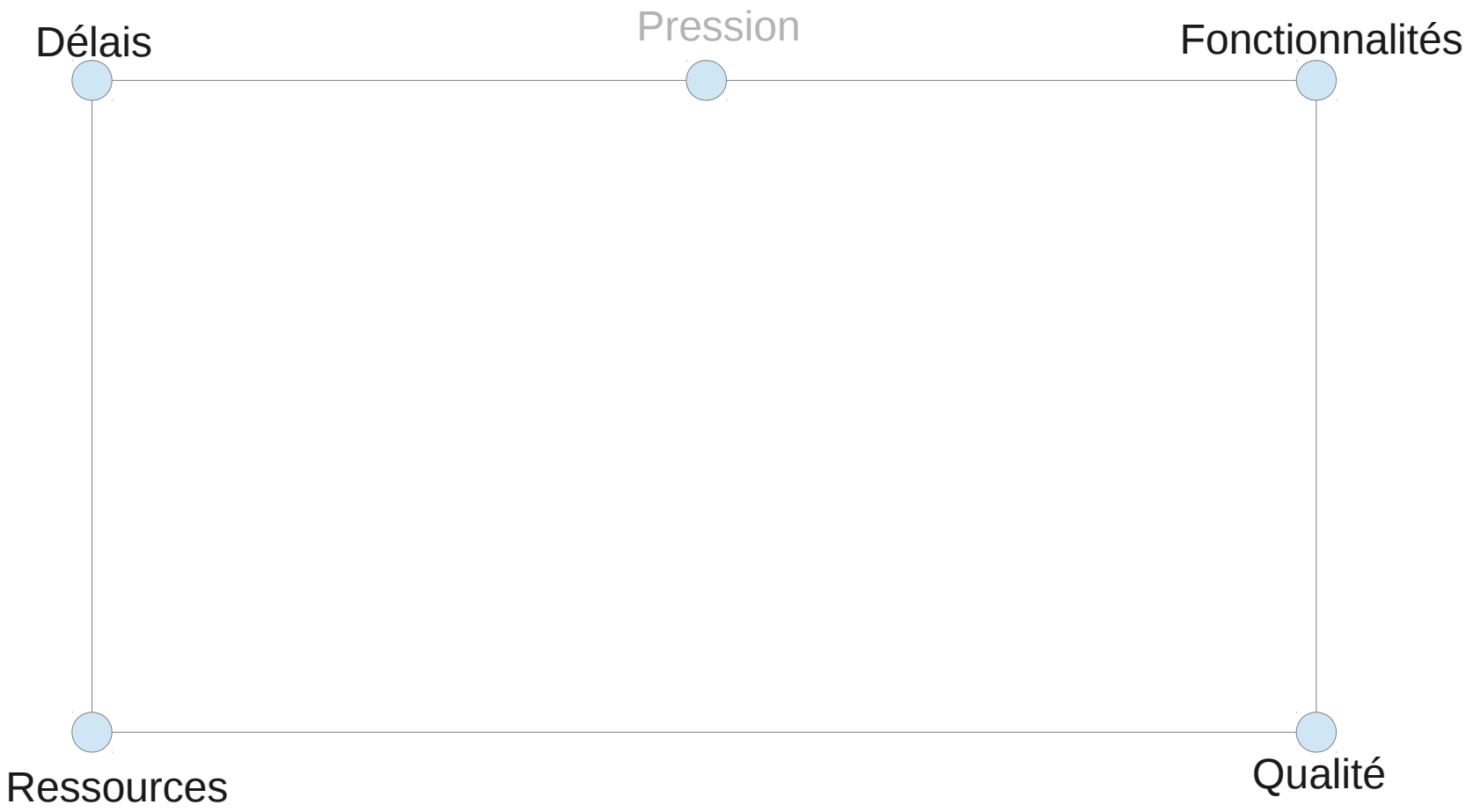
Démo

- Intégration Continue :
  - Modification de code → Tests (avec puis sans erreur)→ Commit → deploy
  - Rapport de tests TU
  - Rapport de tests TI
  - Rapport de tests TP
- Sonar
  - Voir les rapports bruts
  - Voir les tendances
  - Zoom sur certains concepts avancés de qualité

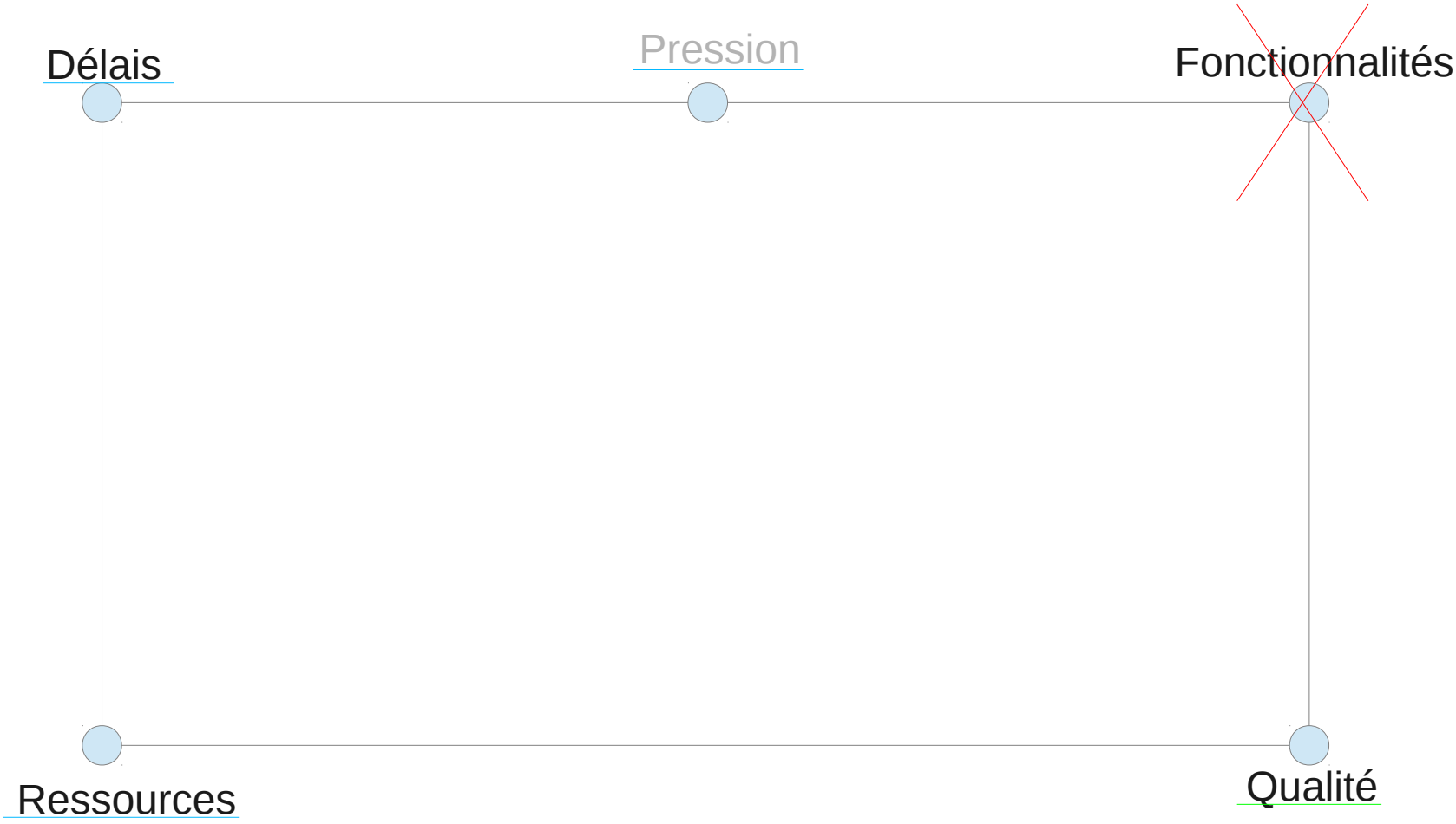
En termes de gestion de projets

- Fiabilisation des développements
- Détection des bugs, des régressions au plus tôt
  - Réduction des phases de recette
- Suivi constant de la qualité
  - Indicateurs objectifs sur la santé du produit, pouvant être utilisés en pilotage : la **dette technique** peut être un élément de pilotage
  - Mesurer la qualité permet aussi d'évaluer certains risques
- Capacité à livrer tôt, souvent (en continu) à moindre risque
  - réduction de l'effet tunnel
  - possibilité d'impliquer le client pour avoir du feedback très tôt.
  - livraison des nouvelles fonctionnalités plus rapide

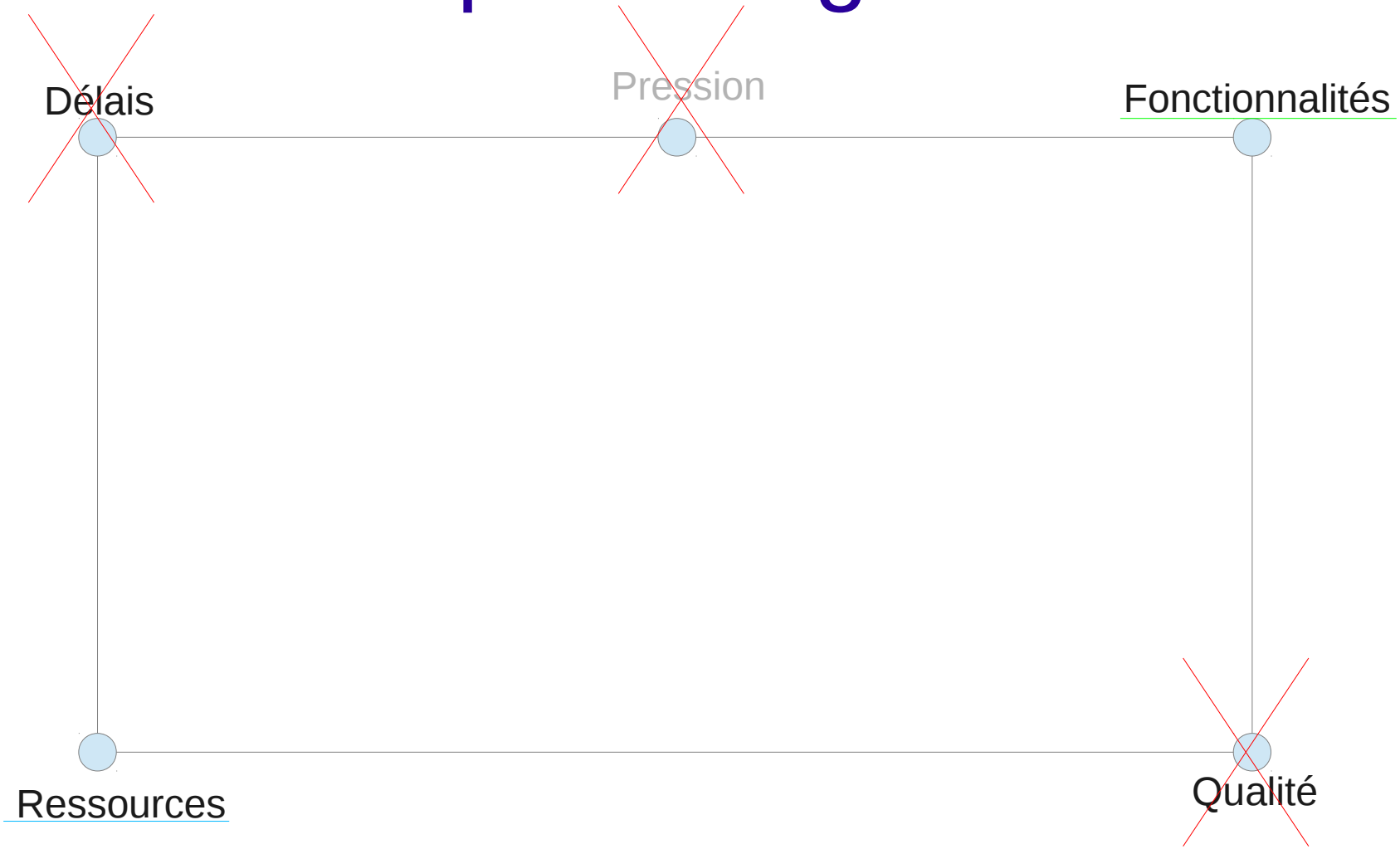
4 (5?) éléments de pilotage



4 (5?) éléments de pilotage : réponse « traditionnelle »



4 (5?) éléments de pilotage : réponse Agile



Retour d'expérience

- Contexte : équipe 10 personnes, niveaux et compétences hétérogènes, 7 développeurs (4 expérimentés, 3 débutants), beaucoup de projets en « maintenance » (corrections + quelques évolutions), 2 nouveaux projets par an.
- Démarche d'amélioration continue (pas de rupture brutale dans les pratiques)
- (Faire) accepter le fait que développer avec des tests automatisés et dans une perspective de haute qualité représente au moins 30% de temps de travail en plus que pour une même fonctionnalité développée en mode « quick and dirty »
- Adapter / contextualiser : ne pas mettre en place de pratique si elles ne sont pas utiles, ne pas être dogmatique
- Avoir des indicateurs permettant de mesurer le ROI des pratiques (satisfaction utilisateur, anomalies réduites, plaisir de l'équipe, évolutivité...)
- Ne pas faire des tests pour faire des tests, ni de la qualité pour des indicateurs : seul objectif = satisfaction utilisateur grâce à fiabilité et évolutivité
- Les membres de l'équipe doivent partager les objectifs qualité, les maîtriser et être un élément moteur de la démarche. L'équipe et le manager doivent partager cette même vision
- Utiliser des outils « standards », libres, éviter les « factory » propriétaires dans laquelle vous risquez d'être enfermés.
- Sur des projets déjà en maintenance, se concentrer sur ce qui apporte le plus à moindre coût (tests d'intégration/fonctionnels automatisés) ; sur les nouveaux projets, il est plus simple de commencer avec de nouvelles pratiques.

Perspectives

- Continuous deployment sans interruption de service et sans perte de session (Facebook, Google,...).
  - Blue-green deployment (http://martinfowler.com/bliki/BlueGreenDeployment.html)
  - Exemple de facebook : http://www.facebook.com/video/video.php?v=10100259101684977&oid=9445
- DevaaS :
  - http://www.cloudbees.com/
  - https://www.shiningpanda-ci.com/
  - https://travis-ci.org/
  - https://semaphoreapp.com/
  - http://www.atlassian.com/software/bamboo/overview
  - http://www.cloudforge.com
  - Ou autre PaaS : google App Engine, Heroku, OpenShift, CloudFoundry...

Références

- http://martinfowler.com/articles/continuousIntegration.html
- http://continuousdelivery.com/
- http://www.sqale.org/
- http://fr.slideshare.net/ehsavoie/usine-logicielle-orange-labs
- http://agilemanifesto.org/
- http://martinfowler.com/bliki/TechnicalDebt.html
- http://manifesto.softwarecraftsmanship.org/
- http://www.agiliste.fr/items/continuous-delivery/
- http://blog.xebia.fr/2010/12/21/livre-blanc-qualite-logicielle/
- http://blog.xebia.fr/2011/09/30/livre-blanc-maitrisez-votre-dette-technique/
- Les fiches PLUME sur Sonar, Jenkins, Archiva, Maven, Subversion, etc... !

Questions / réponses et discussions

Annexes

Manifeste Agile (extrait des 12 principes)

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Continuous attention to technical excellence and good design enhances agility.

(Quelques) Outils

Gestion de version

Build

Make (& co)

Entrepôts d'artefacts

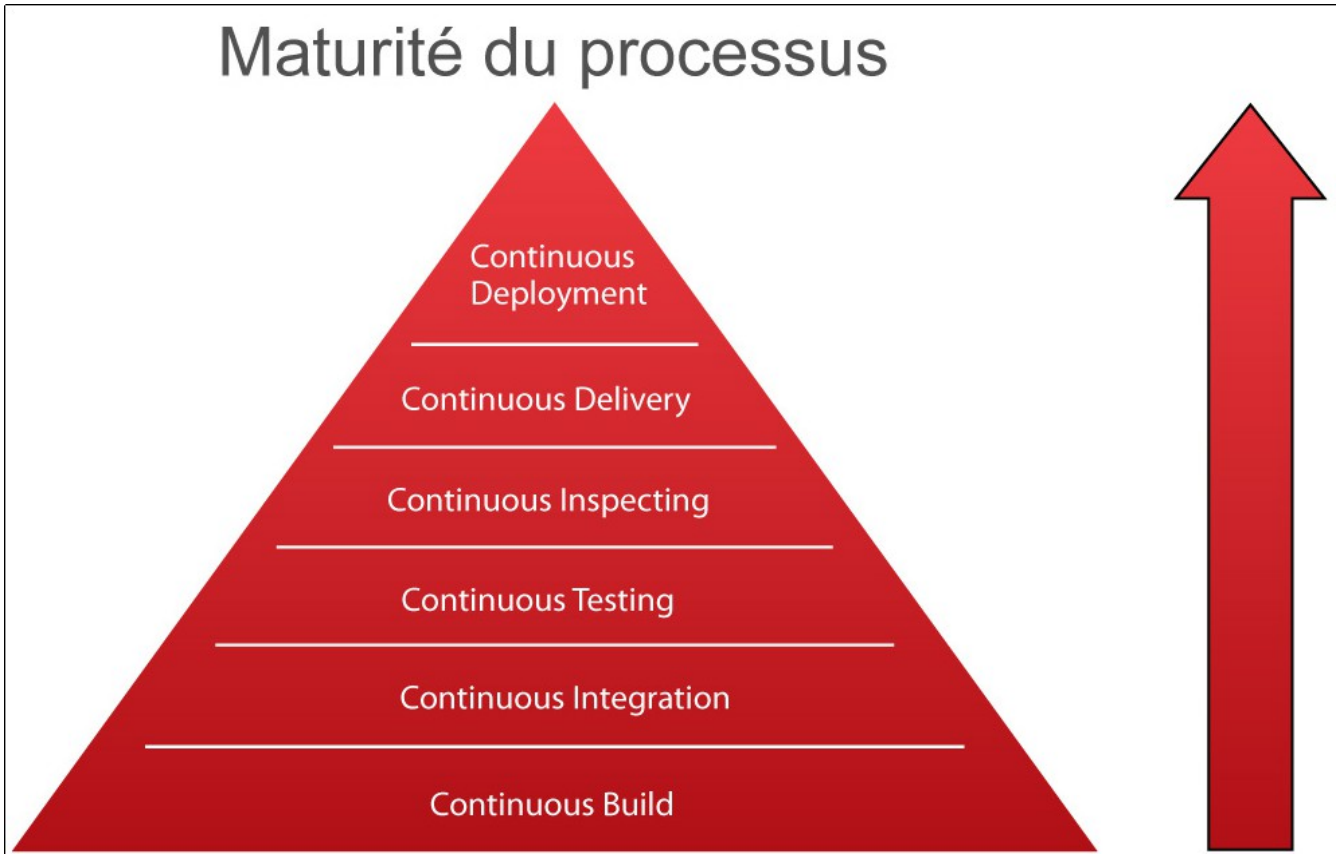
Tests

Watir

Intégration continue

Inspection

Fixer les limites par rapport à sa maturité



Source : http://www.zenika.com/Conseil/integration-continue.html

Software craftsmanshift manifesto

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

- Not only working software,

  but also well-crafted software

- Not only responding to change,

  but also steadily adding value

- Not only individuals and interactions,

  but also a community of professionals

- Not only customer collaboration,

  but also productive partnerships

That is, in pursuit of the items on the left we have found the items on the right to be indispensable.