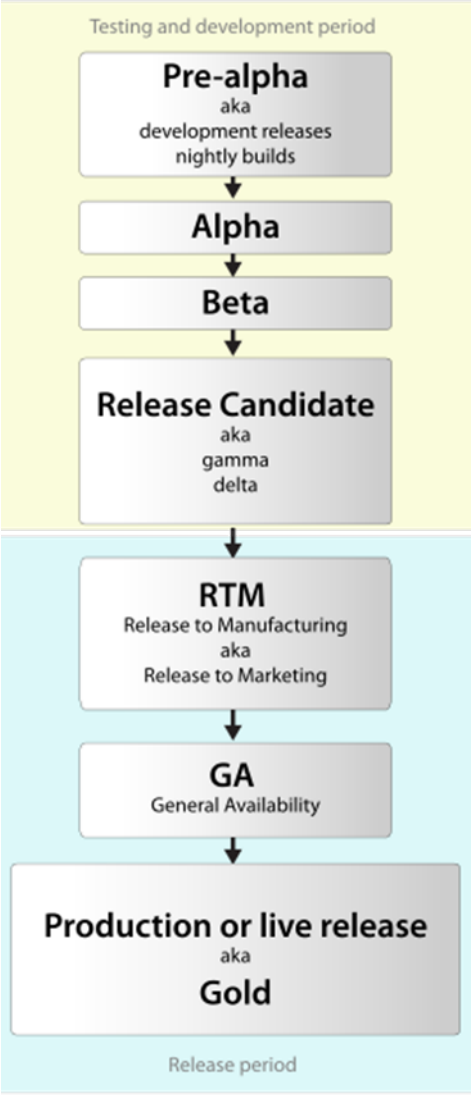
IFT 2255 - Genie Logiciel

Déploiement

Versions d'un logiciel

- **Pré-alpha:** encore en développement, avant les tests formels complétés
- **Alpha:** en mode tests fonctionnels
- **Beta:** logiciel fonctionnel, testé, dans les mains d'utilisateurs contrôlés
- **Candidat à la livraison:** potentiellement la version finale à moins de découverte d'autres défauts ; code source est figé
- **En production:** disponible au public ; version commercialisée



Séquence de version typique

- `[0-9]+ · [0-9]+ · ([0-9]+ ·)? [0-3] · [0-9]+`
  - Deux premiers nombres marquent la version (p.ex. 1.3)
  - Deux derniers nombres marquent le statut (p.ex. α, β, RC)
- La version 1.2 contient des améliorations mineures par rapport à la version 1.1
- La version 2.0 contient des améliorations majeures par rapport à la version 1.9
- 1.2.0.1: alpha 1 de la version 1.2
- 1.2.1.2: beta 2 de la version 1.2
- 1.2.2.3: release candidate 3 de la version 1.2
- 1.2.3.0: version 1.2 en production

Environnements

- **Environnement de développement**
  - Ordinateur personnel du développeur contenant:
    - Logiciels de programmation (Eclipse, NetBeans, Visual Studio)
    - Base de donnée et serveur web locaux
    - Client du système de contrôle de révision
- **Environnement de test (pré-production)**
  - Simule l'environnement de production
    - Serveur/PC (ou réseau) vanille : souvent roule sur une **machine virtuelle**
    - Seuls le logiciel et ses dépendances y sont installés
    - Peut avoir un système d'exploitation différent de l'environnement de dev
  - Sur lequel les **tests de performance et d'acceptation** sont faits
- **Environnement de production**
  - Vrai environnement où le logiciel est installé
  - Utilisateurs finaux opèrent dessus ou communiquent avec

Déploiement

Transition de l'application d'un environnement à un autre

- Consiste en différentes activités
  - Assemblage et configuration
  - Livraison
  - Transfert
  - Installation
  - Activation
  - Mise à jour

Assemblage et configuration

- Chaque **composant** définit les interfaces qu'elle fournit et qu'elle requiert
- Les composants interdépendants sont assemblés dans un **artéfact *(assembly)*** qui peut aussi requérir des interfaces
  - L'assembly communique donc avec d'autres assemblies ou avec l'environnement pour combler ses dépendances
- Déployer une application revient à **instancier** les assemblies dans un environnement
- Chaque composant est libellé par une **version** qui identifie l'ordre de sa révision et de sa variante spécifique à l'environnement ciblé

Livraison

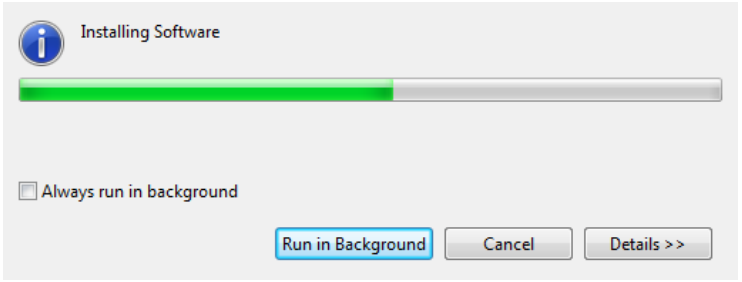
- On forme un paquet qui contient les assemblies et la méta-information de son contenu
  - Description du contenu et des ressources dont il dépend
    - Version, auteur, nom du produit, producteur, etc.
  - Chaque paquet peut contenir plusieurs implémentations des composants pour satisfaire les besoins de différents environnements matériel et logiciel
- Exemples
  - Unix: RPM, tar.gz
  - Windows: MSI, DLL
  - Android: APK


Transfert

- L'application est transférée pour être utilisée par les **utilisateurs finaux**
- On ne peut livrer le logiciel qu'une fois les tests d'acceptation complétés sans échec
- TOUS les artéfacts du projet sont livrés
  - Documentation, code source, tests, manuel utilisateur, guide d'installation
- Assignation de licence

Installation

1. **Déballe** le paquet *(unpacking)*
2. Vérifie que les exigences du système hôte sont satisfaites
3. Crée (ou copie) les fichiers nécessaires pour exécuter l'application sur le disque
4. Ajoute des données de **configurations**: registres, variables d'environnement, fichier de configuration
5. Déclenche **l'activation** du logiciel
   - Démarre l'exécution du logiciel ou met en place les déclencheurs qui le démarreront au bon moment
     - Via une interface graphique, des scripts ou des daemons



Mise à jour

- Change une partie du logiciel installé à cause de la livraison d'une nouvelle version
- Cas spécial d'installation
- Peut nécessiter la désactivation ou la désinstallation du logiciel pour effectuer les changements nécessaires

- Logiciel peut se mettre à jour sans intervention humaine
  - Vérifie automatiquement si une nouvelle version est disponible
  - Bonne pratique pour un logiciel sur le serveur, mais pas pour un logiciel chez le client !

Problèmes rencontrés

- Plus de 50% des logiciels commissionnés ne sont pas utilisés, surtout à cause d'échec lors du déploiement

- 80% du coût du logiciel se situe au moment ou après le déploiement

Quelles sont les problèmes
qui rendent le déploiement difficile ?

Est-ce que le problème est le déploiement ?

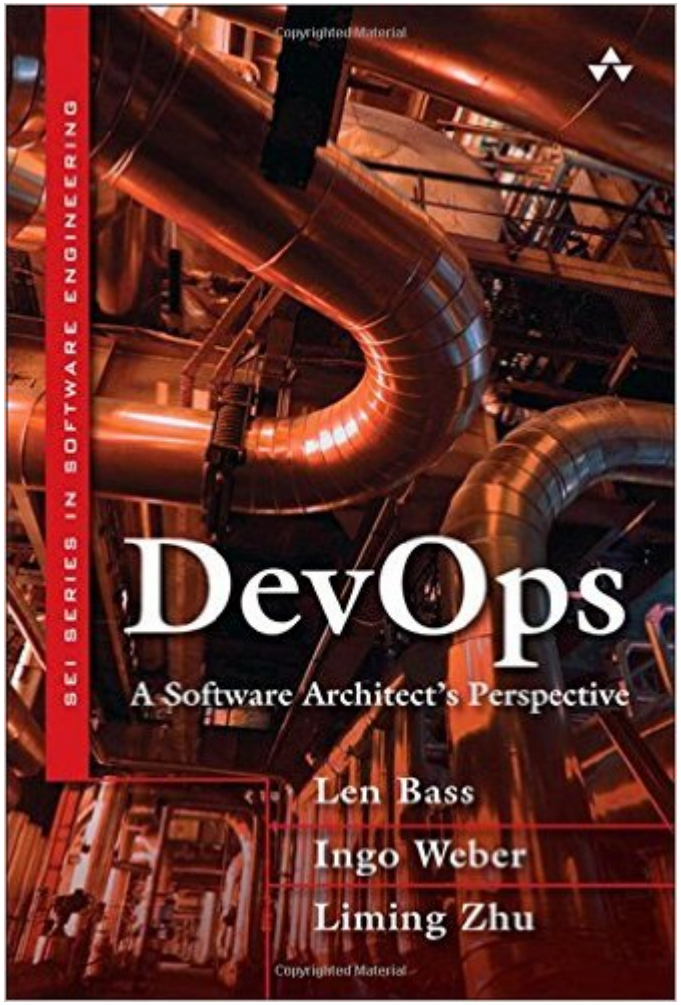
- Pas toujours
- Souvent, les problèmes qui surviennent lors du déploiement proviennent d'erreurs dans l'analyse des exigences
  - C'est un peu tard de les corriger arrivé là…
- Pour éviter cela, le code doit passer par une **longue batterie de tests** avant d'être déployé
- Il y a cependant des enjeux essentiels lors de la transition

Enjeux de la livraison

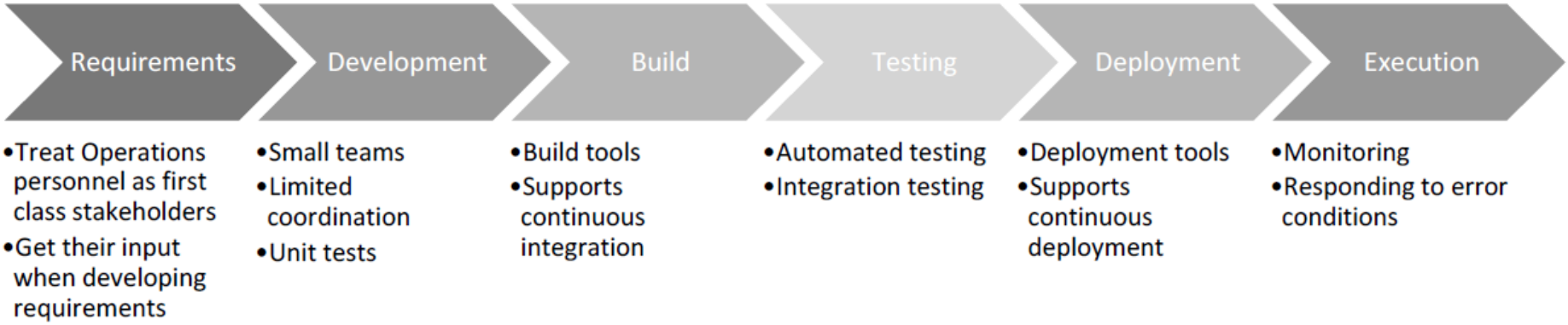
- **Processus d'affaires**
  - La plupart des grands systèmes logiciel demandent que le client change sa façon de travailler
  - Est-ce que le logiciel répond vraiment aux besoins du client ?
- **Formation**
  - Il faut s'assurer que les utilisateurs sont formés adéquatement pour utiliser le logiciel
  - Selon leur rôle d'utilisation ~ acteurs
- **Stratégie de déploiement**
  - Comment installer le logiciel ? Intervention humaine ? Requiert un administrateur système ? Comment le rendre disponible ?
- **Équipement**
  - S'assurer que le matériel de l'utilisateur satisfait les prérequis de l'application
- **Intégration**
  - Comment le logiciel s'intègre-t-il avec les autres logiciels du client ?

Transition entre les développeurs et les opérateurs

- Buts opposés
  - Dev veut introduire de nouvelles fonctionnalités
  - Ops veut conserver la disponibilité du système
    - Ralentit l'échéancier des livraisons
- Cultures différentes
  - Ops ont des capacités de dev limitées
  - Dev ont une vision étroite des opérations



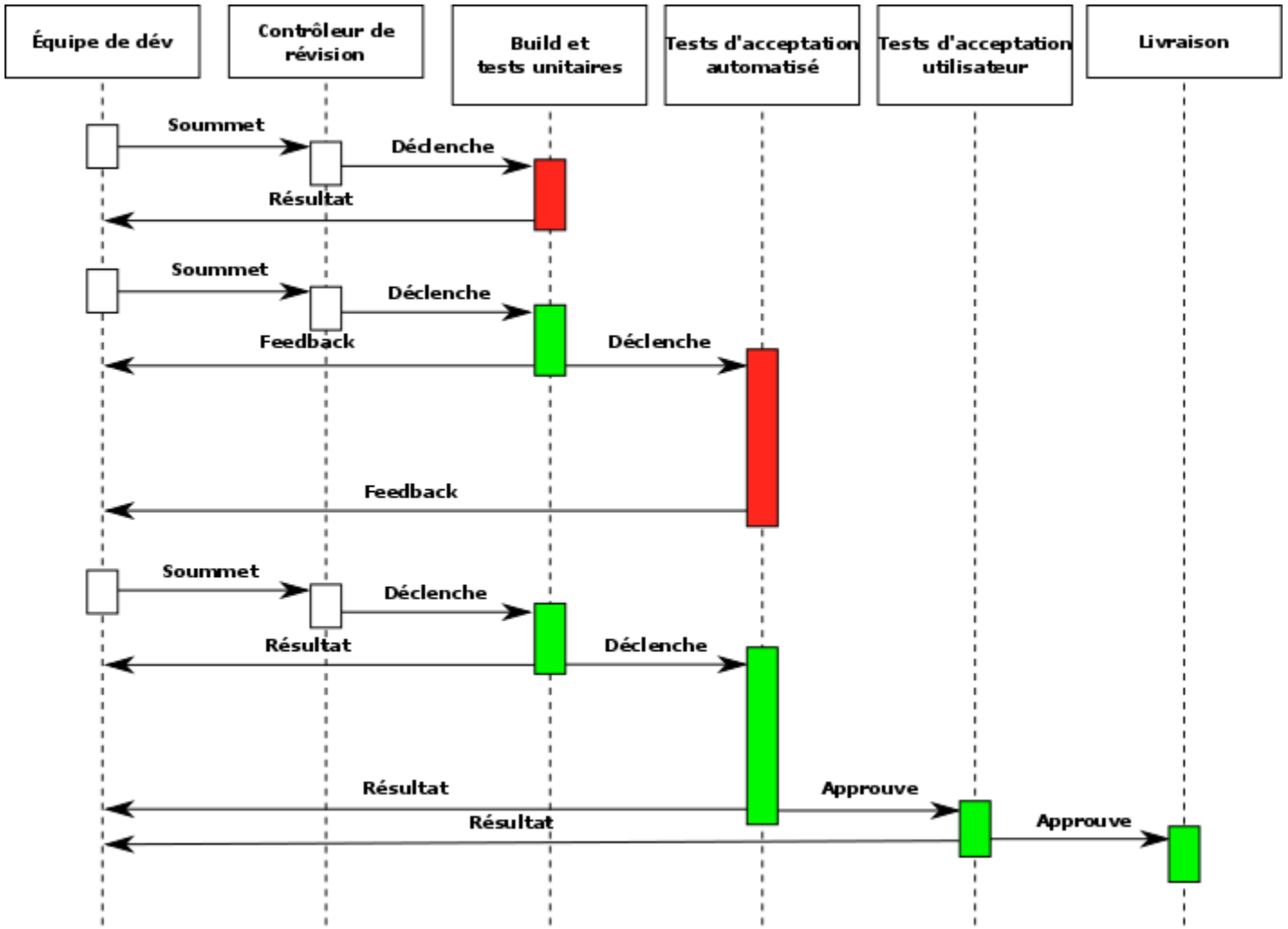
Pratiques DevOps



- Inclure les Ops tout au long du processus de développement
- Déployer en continue, tout automatiser
- Instaurer un processus de déploiement suivi par tous
- Développement de l'infrastructure doit suivre les mêmes pratiques que le code de l'application
  - Ops utilisent souvent des scripts ad hoc

Intégration continue

Architecture en pipes-et-filtres



Exemple imvu

- Développeurs soumettent le code tôt et souvent
  - En deux phases : pre-commit, commit
- Soumission déclenche l'exécution d'une **suite de tests**
  - Milliers de fichiers de tests distribués sur une 30aine de machines
  - ~9min pour exécuter une suite
- Quand le commit a passé tous les tests, il est envoyé au **déploiement**
  - Code est installé sur une 100aine de machines
  - Code n'est disponible que sur quelques machines : les canaries
- Un programme **d'échantillonnage** examine les résultats de ces machines
  - S'il y a une régression statistiquement significative, la révision est automatiquement renversée *(roll back)*
  - Sinon la grappe *(cluster)* devient actif
- Déploie du nouveau code **50 fois par jour**