IFT 2255 - Genie Logiciel

Maintenance

Situation post-déploiement

- Supposons qu'un logiciel ait été développé pour 2M $
- Quand le temps de le maintenir est venu, on fait face à certaines difficultés

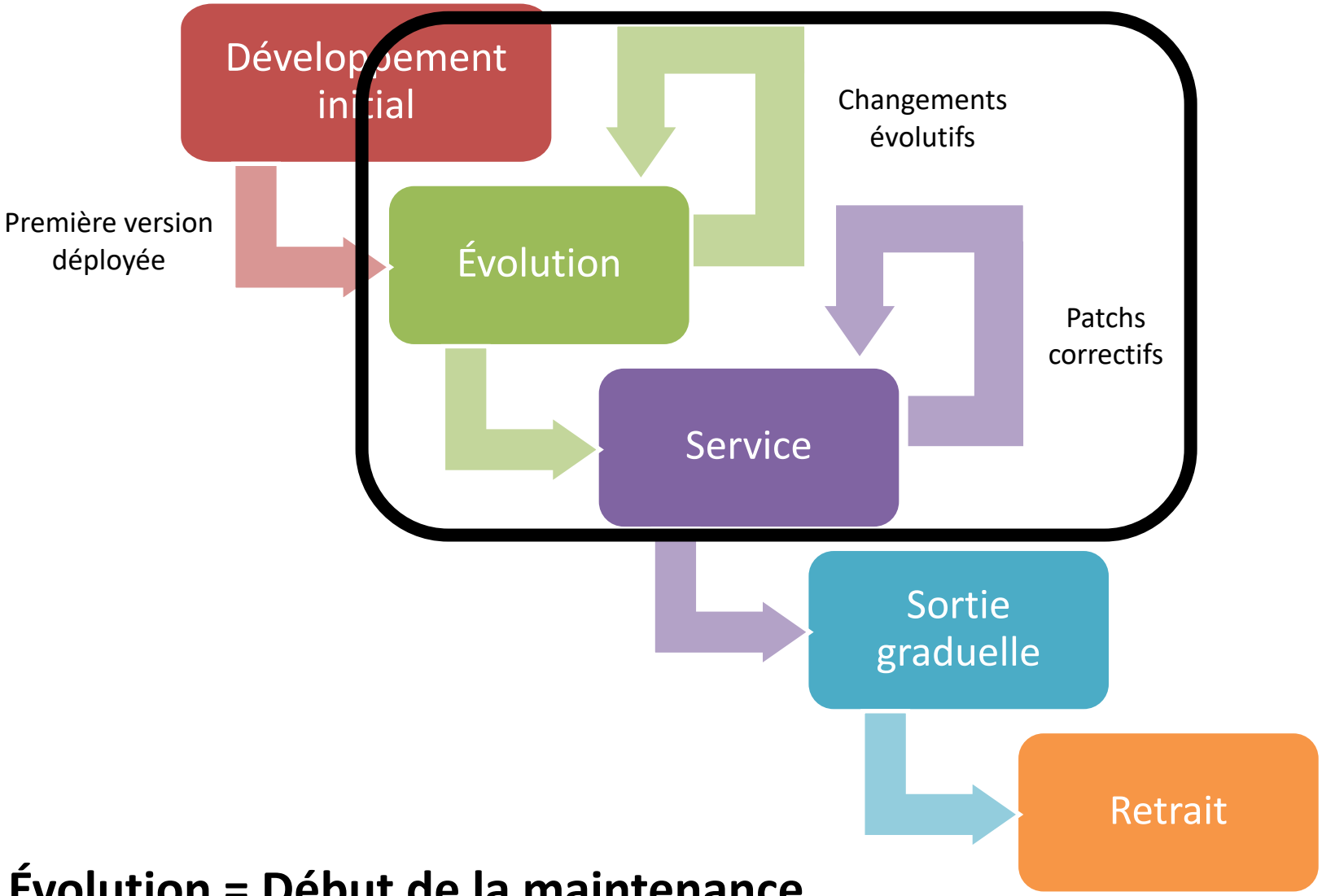
1. Disque utilisé par la base de donnée est plein
   - Compagnie qui a développé le disque n'est plus en service
     - Besoin de trouver un nouveau fournisseur de disque avec plus d'espace
     - Changements pour surmonter les incompatibilités matérielles vont coûter 100K $
2. Développeurs initiaux ont quitté la compagnie
   - Changement à effectuer par une équipe de maintenance qui n'a jamais vu le produit auparavant
3. Le processus de développement a changé
   - Avant cascade, maintenant agile

Maintenir ou redévelopper ?

- Devrait-on développer un nouveau logiciel ?
- Devrait-on effectuer de la maintenance sur celui présent ?
- Dans ce cas, la maintenance serait beaucoup moins coûteuse
  - 5% du prix d'un nouveau logiciel

- C'est toujours plus « facile » de développer un nouveau logiciel, car on en fera un meilleur, mais à quel prix ?
- Dans la majorité des cas, il est bien plus préférable d'évoluer un logiciel (pour des raisons économiques)

Évolution d'un logiciel, post-déploiement



Évolution = Début de la maintenance

Maintenance

- Tout changement de n'importe quel artéfact du logiciel une fois déployé
  - Code source, tests, documents
- Maintenabilité doit être pensée à même le logiciel dès le début
  - Pour réduire l'effort, le temps et les coûts durant la phase d'évolution
  - Ne doit pas être compromise par le processus de développement
- C'est comme si **tout ce qu'on a vu dans ce cours était dévoué à la maintenance**

Types de changements

- **Perfectif** (51%) : accroître la valeur du logiciel suite à des changements demandés par le client
  - Ajouter des fonctionnalités
  - Améliorer la performance
- **Adaptatif** (24%) : préserver la valeur du logiciel en répondant aux changements de l'environnement
  - Transférer vers un nouveau compilateur, système d'exploitation, matériel
  - Changement légal (ex: taux d'imposition)
  - Changement de norme (ex: immatriculation contient des chiffres et lettres, nouvelle devise)

Lientz, B. & Swanson, E. *Software maintenance management: A study of the maintenance of computer application software in 487 data processing organizations*. Reading, MA: Addison-Wesley. 1980.

Types de changements

- **Correctif** (22%) : corriger les défauts restants
  - Analyse, conception, implémentation, documentation, …
- **Préventif** (3%) : protège le logiciel d'une manière proactive en facilitant les changements futurs par anticipation
  - Augmenter sa maintenabilité
  - Invisible à l'utilisateur
    - Améliorer la structure interne (*refactoring*)
    - Transférer de technologie pour augmenter le temps de vie du logiciel

Lientz, B. & Swanson, E. *Software maintenance management: A study of the maintenance of computer application software in 487 data processing organizations*. Reading, MA: Addison-Wesley. 1980.

Maintenance en pratique

- Plus de **60% du coût total**
- **Source majeure de revenu**
  - Certaines compagnies en font leur principal modèle d'affaire
- Néanmoins, plusieurs organisations assignent les tâches de maintenance à des :
  - Développeurs débutants, sans supervision
  - Programmeurs moins compétents
- Maintenance est un des aspects les plus difficiles de la production de logiciel
  - Incorpore **tous** les autres workflows

Travail d'un programmeur de maintenance

- Suppose qu'il reçoit un rapport de défauts
  - Défaut = faute, bogue
- Quelle est la cause ?
  - Il n'y a rien de faux
  - Le manuel utilisateur est incorrect, pas le code
  - Souvent, il y a une faute dans le code

Maintenance corrective

- De quels outils dispose le programmeur de maintenance pour trouver la faute ?
  - Le **rapport de défauts** produit par un utilisateur
  - Le code source
  - Et souvent, rien d'autre (la documentation ?)

- Il doit donc avoir des talents de débogage extraordinaires
  - **La faute peut être n'importe où** dans le produit
  - La cause de la faute peut être située dans une exigence ou un document de conception aujourd'hui inexistant

Problèmes de régression

- Supposons qu'on a localisé la faute
- Comment la corriger sans (ré)introduire une faute?
  - Une faute dans une partie apparemment non-reliée
- Comment minimiser les fautes de régression ?
  - Consulter la documentation complète du produit
  - Consulter la documentation détaillée de chaque module
- Souvent, la documentation est inexistante, incomplète ou erronée
- Il doit déduire du code même toute l'information pour éviter d'introduire une faute de régression

Vérification

- Tester que les modifications fonctionnent correctement
  - En utilisant des **cas de test** spécifiquement conçus pour celles-ci
- Effectuer des **tests de régression**
  - Ré-exécuter les test en utilisant les données de test existantes
- Ajouter les nouveaux cas de test à ces données pour qu'ils fassent parties de tests de régression **futurs**
- **Documenter** tous les changements

Assurer la maintenance

- Maintenance n'est pas un effort ponctuel
- Il faut la planifier tout au long du cycle de vie du logiciel
  - Workflow de conception : utiliser les techniques de dissimulation d'information et d'implémentation
  - Workflow d'implémentation : choisir des noms de variables qui compréhensibles aux programmeurs de maintenance
  - Documentation doit être complète, juste et refléter la version courante de tous les artéfacts
- Ne pas compromettre la maintenabilité durant la maintenance

Problème de cible mobile

- Solution apparente
  - **Geler les exigences** une fois qu'elles ont été signées, jusqu'à la livraison du produit
  - Après chaque requête de maintenance perfective, **geler les exigences** pendant (disons) 3 mois ou 1 an
- En pratique
  - Le client peut ordonner des changements le lendemain
  - Tant qu'il paye la facture, il peut faire des demandes de changement quotidiennement

« Le client est roi »

Problème de maintenance répétée

- Le **problème de cible mobile** est frustrant pour l'équipe de développement
  - Changement dans les exigences pendant que le logiciel est développé
- Des **changements fréquents ont un effet nocif** sur la maintenabilité du produit
- Ce problème est empiré pendant la maintenance
- Plus il y a de changements
  - Plus le produit dévie de sa conception originale
  - Plus il sera difficile d'intégrer de nouveaux changements
  - Moins la documentation est fiable
  - Moins les tests de régression sont à jour
  - Plus le redéveloppement semble nécessaire

Maintenance et OO

- Le paradigme OO facilite la maintenance (en série)
  - Application composée **d'unités indépendantes**
  - **Encapsulation** (indépendance conceptuelle)
  - **Dissimulation I&I** (indépendance physique)
  - Seul moyen de communication est via l'envoi de **messages**

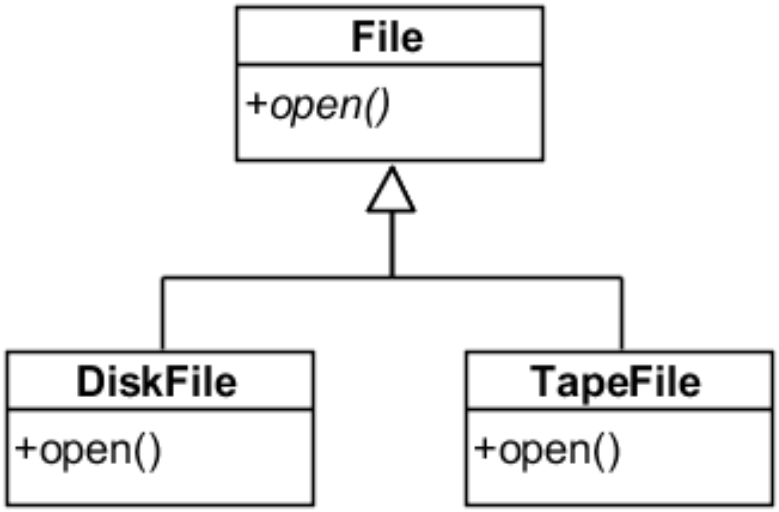
- Mais il l'entrave aussi
  - Trop grande **hiérarchie** d'héritage
  - Conséquences du **polymorphisme** et de liaisons dynamiques
  - **Fragilité** de l'héritage

Taille de la hiérarchie d'héritage

```
class UndirectedTreeClass
{
   …
   void displayNode (Node a);
   …
}// class UndirectedTreeClass

class DirectedTreeClass : public UndirectedTreeClass
{
   …
}// class DirectedTreeClass

class RootedTreeClass : public DirectedTreeClass
{
   …
   void displayNode (Node a);
   …
}// class RootedTreeClass

class BinaryTreeClass : public RootedTreeClass
{
   …
}// class BinaryTreeClass

class BalancedBinaryTreeClass : public BinaryTreeClass
{
   Node          hhh;
   displayNode (hhh);
}// class BalancedBinaryTreeClass
```

- Pour trouver ce que `displayNode()` fait dans `BalancedBinaryTreeClass`, il faut **parcourir tout l'arbre**
- L'héritage peut être dispersé à travers tout le programme
  - Contredit les « unités indépendantes »
- On peut utiliser des **outils CASE** pour faciliter ce problème

Polymorphisme et liaisons dynamiques



- Application échoue lors de l'appel à `myFile.open()`
- Quelle version de `open()` contient l'erreur ?
  - Un outil CASE ne peut pas aider (statique)
  - Il faut explorer les **traces d'exécution**

Fragilité de l'héritage

- Créer une sous-classe par héritage
  - Elle n'affecte aucune superclasse
  - N'affecte aucune autre sous-classe
- Modifier cette sous-classe
  - Aucune autre classe n'est affectée
- Modifier une superclasse
  - **Toutes les sous-classes sont affectées**