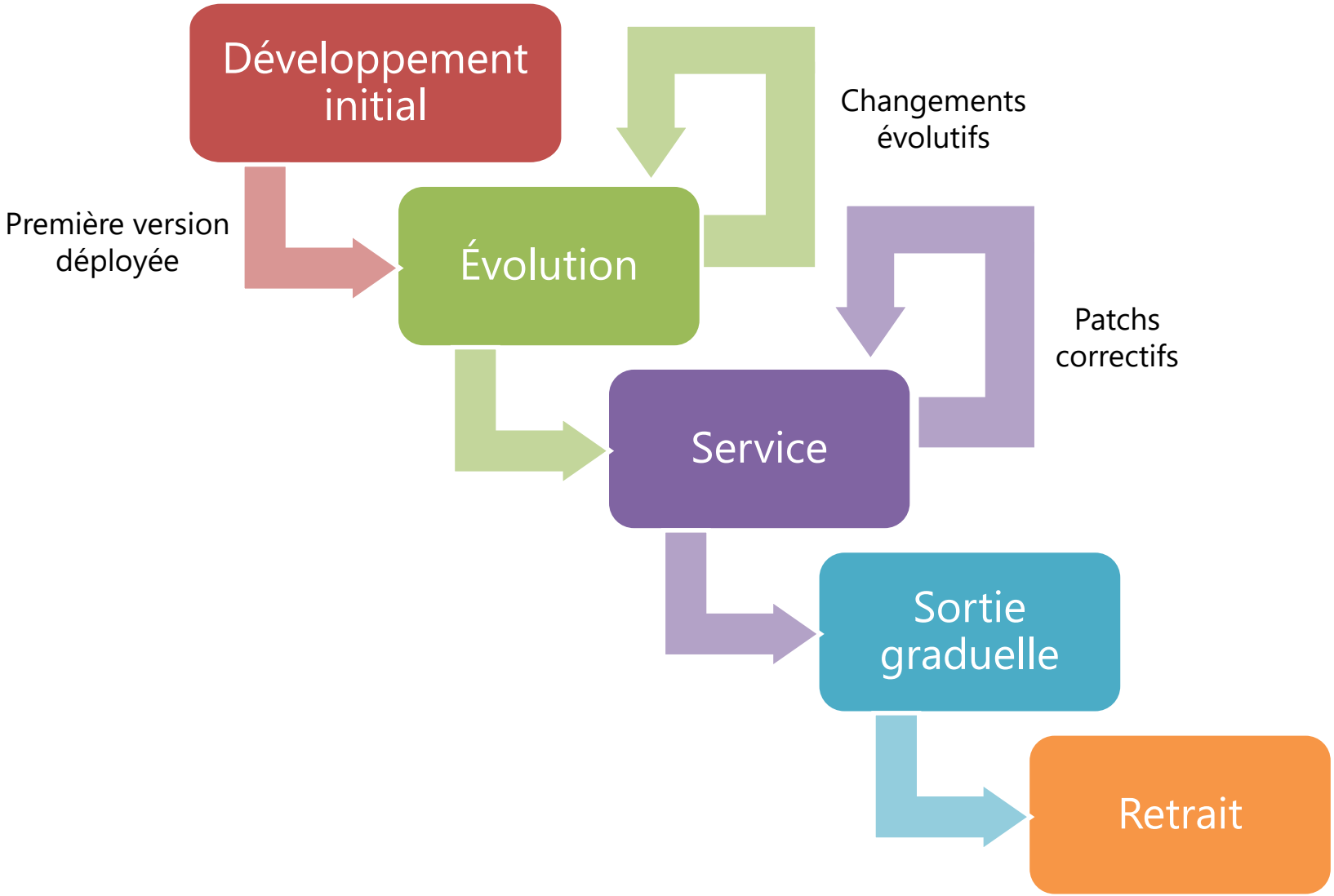
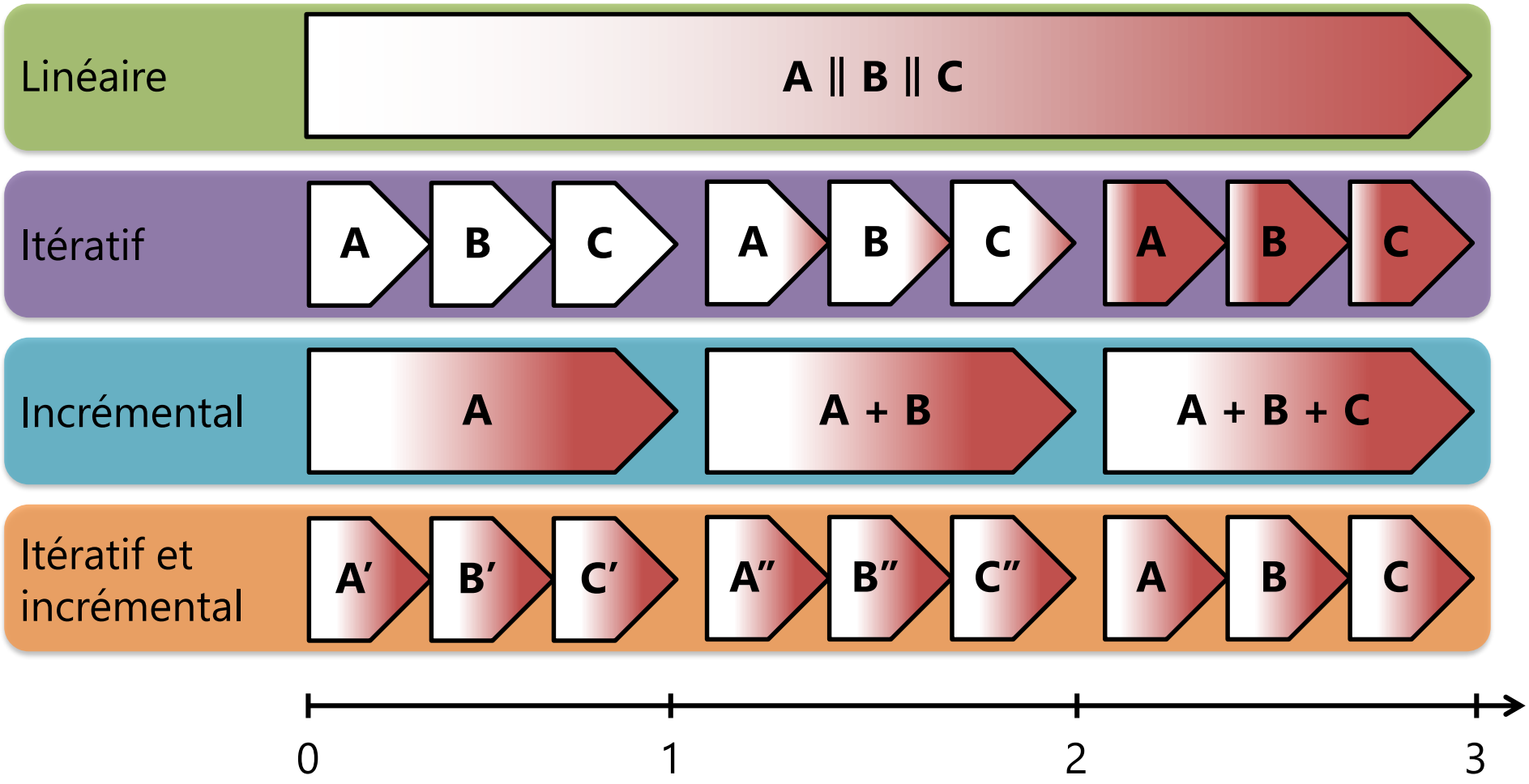
Évolution d'un logiciel



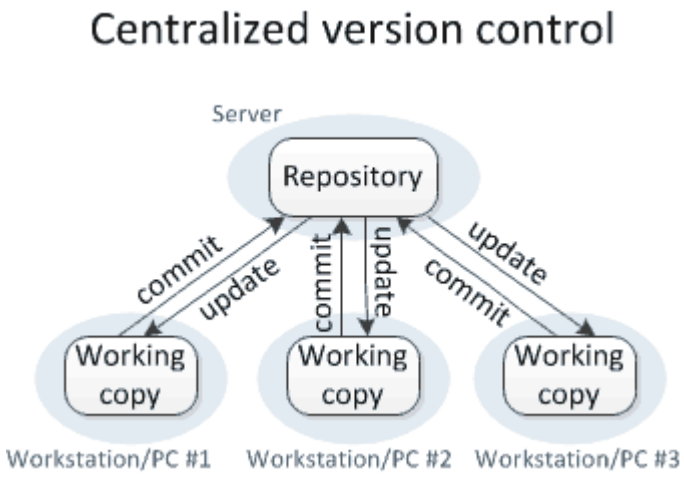
Modèles de développement logiciel

- Projet de 3 mois avec 3 composants (A, B, C) à livrer

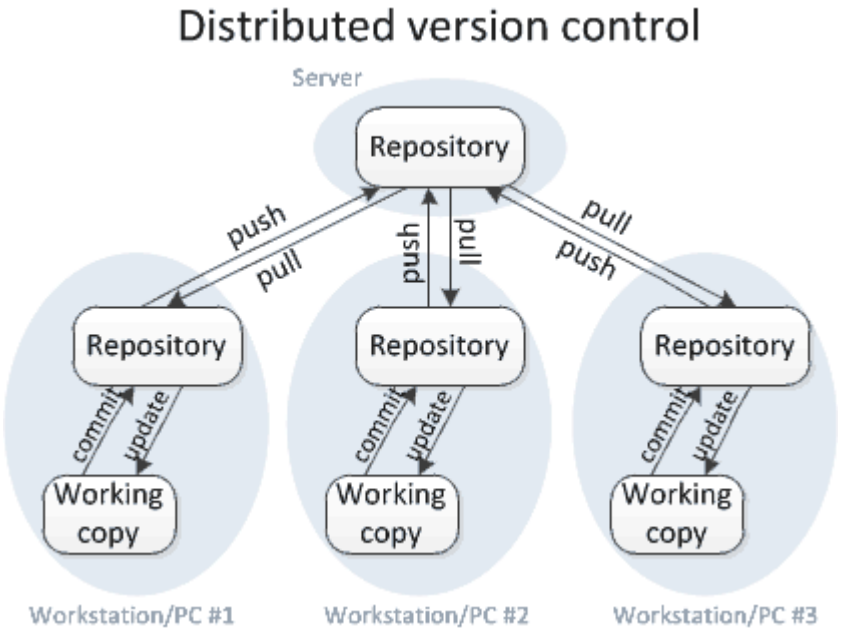


Systèmes de gestion de révisions

Centralisé
SVN, CVS

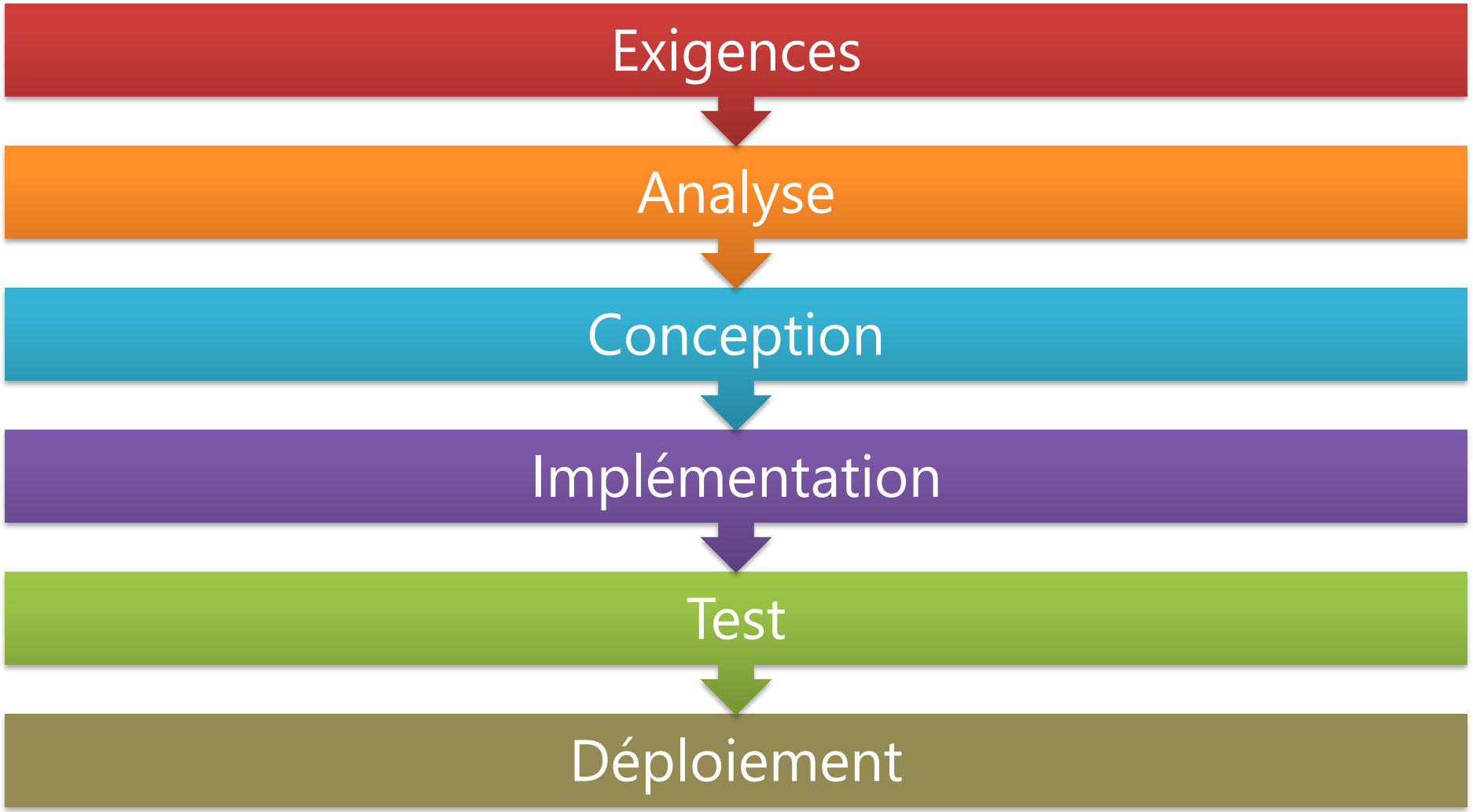


Décentralisé
GIT, Mercurial



Flux de travail *(Workflows)*

Activités en continu



Exigences

- Besoins fonctionnels
- Contraintes non fonctionnelles
- Flux d'affaires
- Données et formats
- Interfaces utilisateur
- Interface avec d'autres systèmes

Prototypage des exigences

- Maquette démonstrative, première étude de faisabilité
- Identification de besoins conflictuels, omis ou mal saisis
- Prototype **jetable**
  - Porter attention sur les besoins moins bien compris
- Prototype **évolutif**
  - Porter attention sur les besoins les mieux compris

Cas d'utilisation

Cas d'utilisation: Dépôt de fonds

But: Le client a l'intention de déposer des fonds dans son compte bancaire.

Acteurs: client, logiciel bancaire (secondaire)

Préconditions:

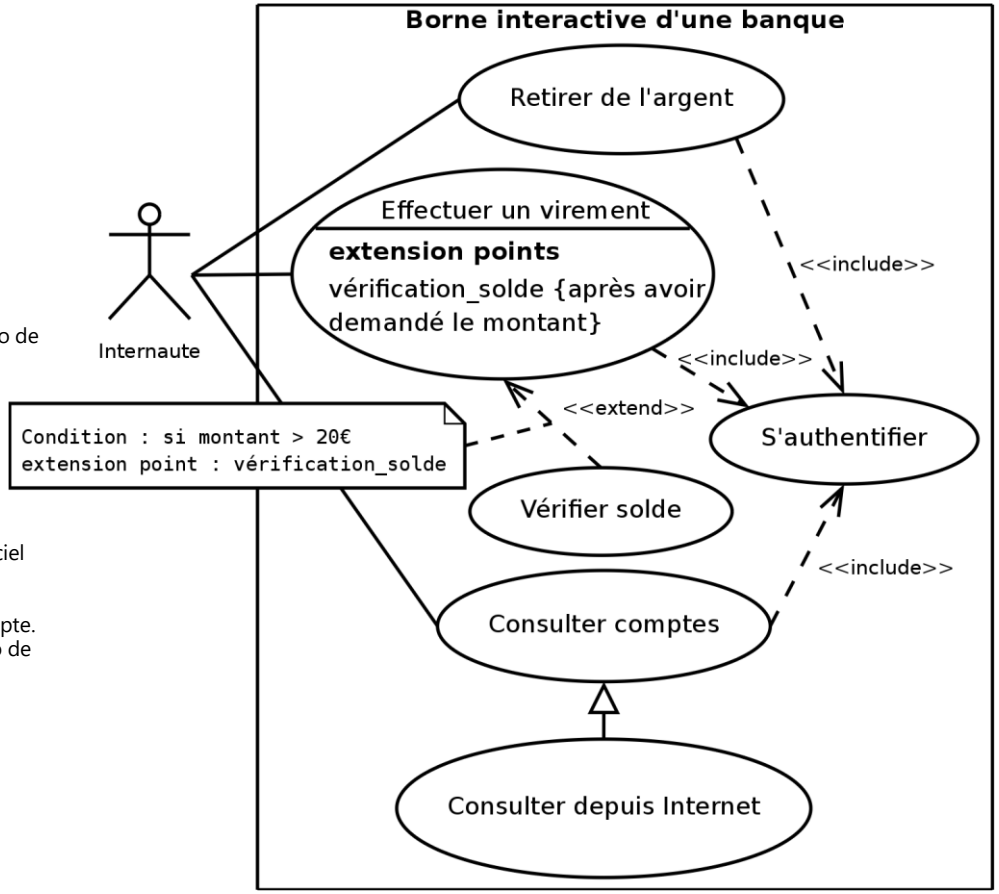
Scénario principal :

1. Lorsque le système détecte une carte, il invite le client à entre son NIP.
2. Le système valide l'authentification.
    2.1. Le système communique avec le logiciel bancaire pour valider le numéro de carte et le NIP.
3. Le système affiche le menu au client qui peut sélectionner une option.
4. Le client choisit «Fonds de dépôt».
5. Le système affiche les comptes dans lesquels le dépôt peut être effectué.
6. Le client sélectionne un compte.
7. Le client entre le montant à déposer et confirme le montant.
8. Le système ouvre la fente de dépôt et attend que le client insère une enveloppe contenant le montant sous forme de chèque ou en espèces.
9. Lors de la réception du montant, le système communique la transaction au logiciel bancaire.
    9.1. Le système reçoit confirmation du dépôt.
    9.2. Le système confirme au client que le montant a été déposé dans le compte.
10. Le système imprime un reçu indiquant la date, le montant du dépôt, le numéro de compte, et le solde après le dépôt.
11. Le système éjecte la carte.

Scénarios alternatifs :

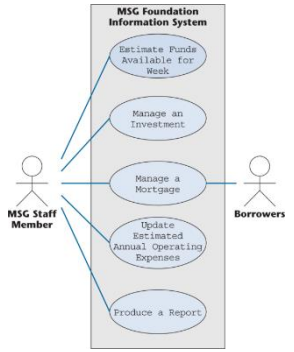
2a. NIP incorrect a été saisi
    2a.1. Le système informe le client.
    2a.2. Le cas d'utilisation continue à l'étape 11.
9a. Le logiciel bancaire rejette la transaction.
    9a.1. Le système informe le client que la transaction n'a pas abouti.
    9a.2. Le système retourne l'enveloppe au client.
    9a.3. Le cas d'utilisation continue à l'étape 3.

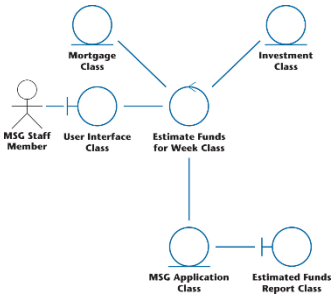
Postconditions:



Modèle d'analyse

- **Modèle fonctionnel**
  - Fonctionnalités du système
  - Cas d'utilisations, scénarios
- **Modèle d'objet**
  - Concepts individuels manipulés par le système et leurs propriétés
  - Classes, composants
- **Modèle dynamique**
  - Comportement du système
  - Activités, flux de données





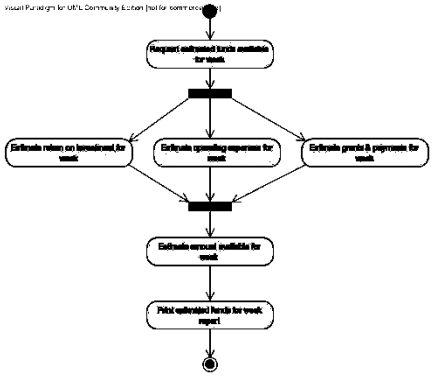
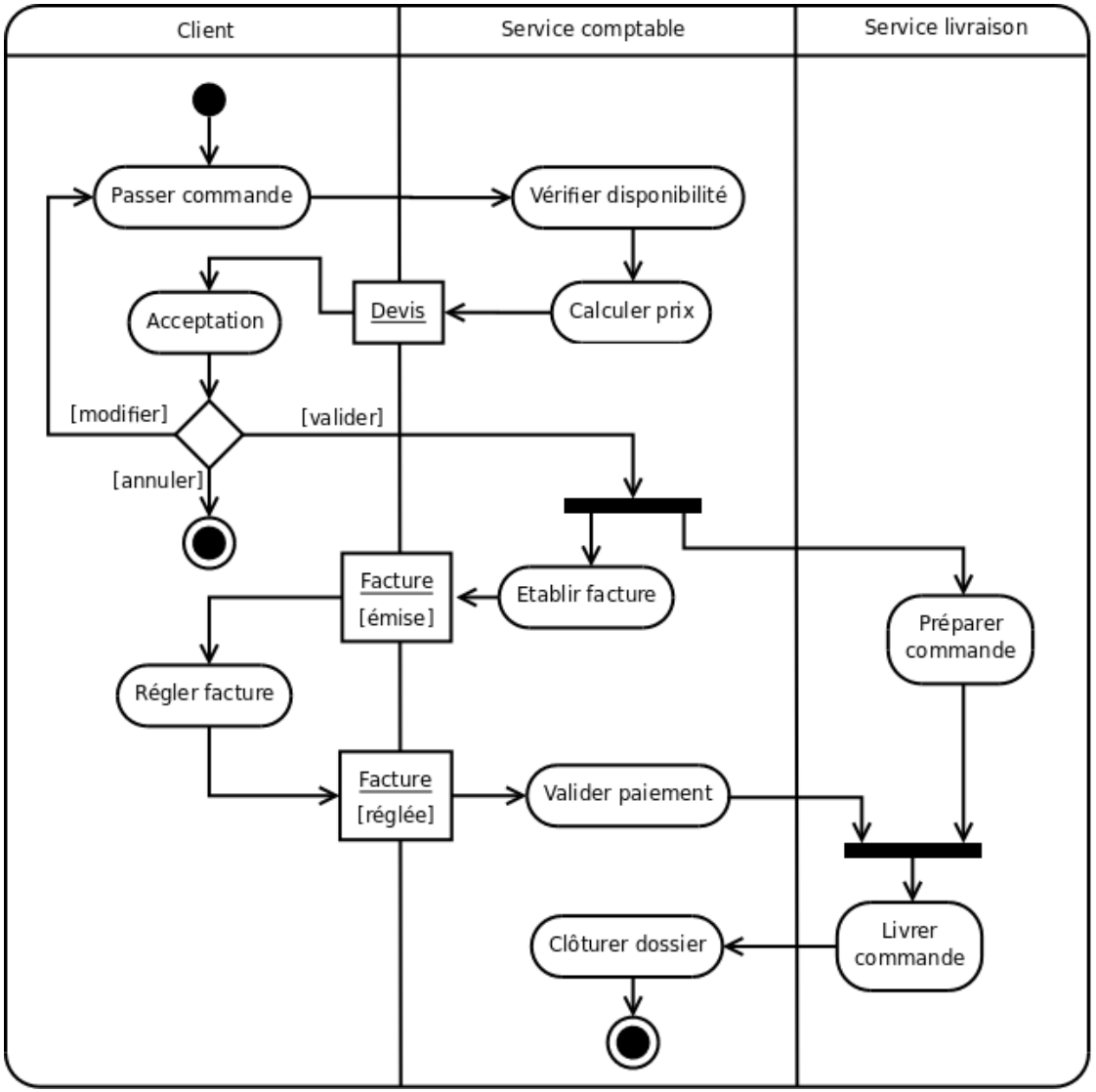


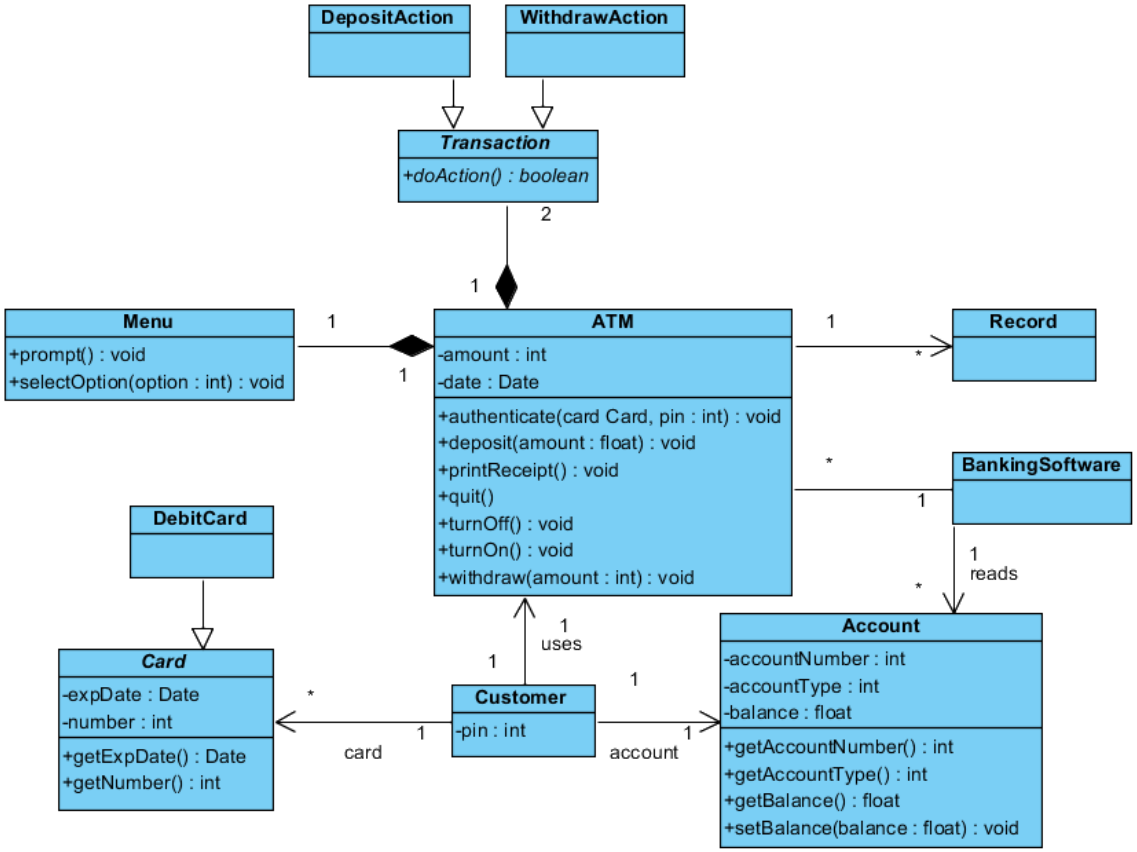
Diagramme d'activité UML

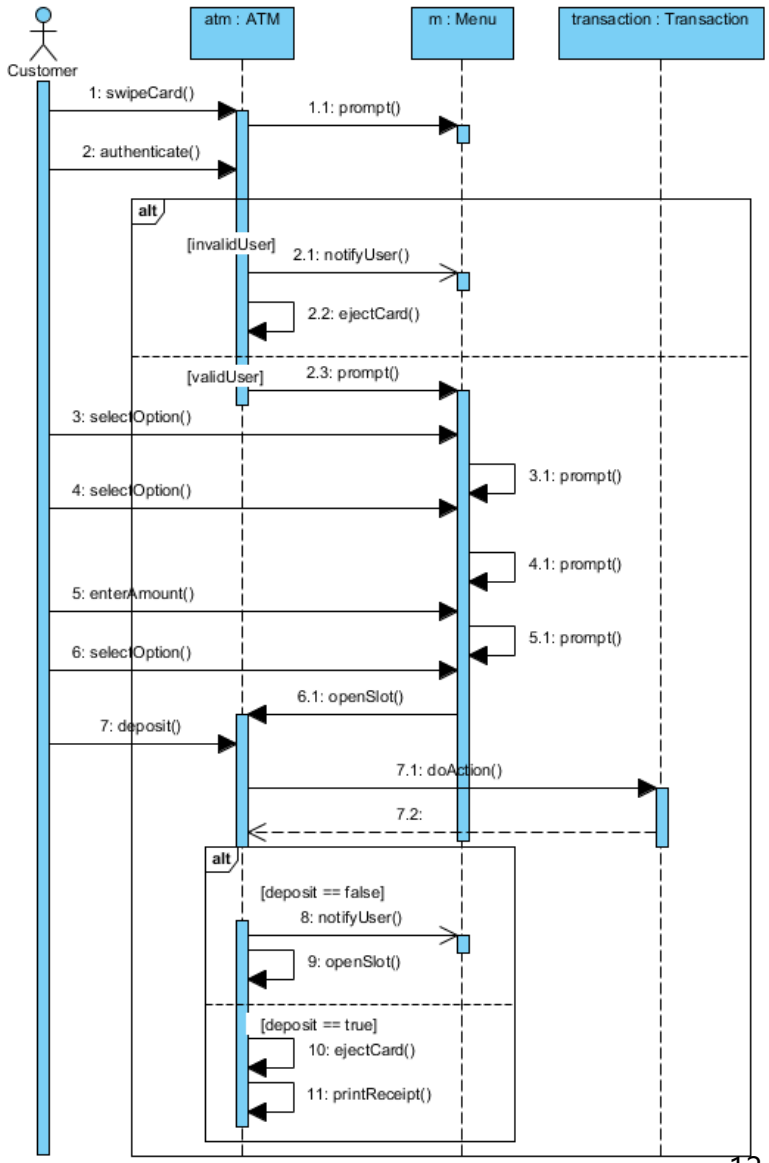


Conception

- Construction conceptuel du système
- Décomposition du système
- Modularisation

Conception


DepositAction
WithdrawAction
Transaction
+doAction() : boolean
2
1
Menu
+prompt() : void
+selectOption(option : int) : void
1
1
ATM
-amount : int
-date : Date
+authenticate(card Card, pin : int) : void
+deposit(amount : float) : void
+printReceipt() : void
+quit()
+turnOff() : void
+turnOn() : void
+withdraw(amount : int) : void
1
*
Record
*
1
BankingSoftware
1
reads
*
DebitCard
Card
-expDate : Date
-number : int
+getExpDate() : Date
+getNumber() : int
*
1
card
1
uses
1
Customer
-pin : int
1
account
Account
-accountNumber : int
-accountType : int
-balance : float
+getAccountNumber() : int
+getAccountType() : int
+getBalance() : float
+setBalance(balance : float) : void


sd Deposit
Customer
atm : ATM
m : Menu
transaction : Transaction
1: swipeCard()
1.1: prompt()
2: authenticate()
alt
[invalidUser]
2.1: notifyUser()
2.2: ejectCard()
[validUser]
2.3: prompt()
3: selectOption()
3.1: prompt()
4: selectOption()
4.1: prompt()
5: enterAmount()
5.1: prompt()
6: selectOption()
6.1: openSlot()
7: deposit()
7.1: doAction()
7.2:
alt
[deposit == false]
8: notifyUser()
9: openSlot()
[deposit == true]
10: ejectCard()
11: printReceipt()

Architecture

- Styles d'architecture
  - Pipes et filtres, en couche, MVC, client-serveur, 3-tier
  - Interactif, événementiel, base de données
- Réutilisation logiciel
  - Programmation orientée composants
  - Architectures et techniques de réutilisation
- Couplage et cohésion des modules

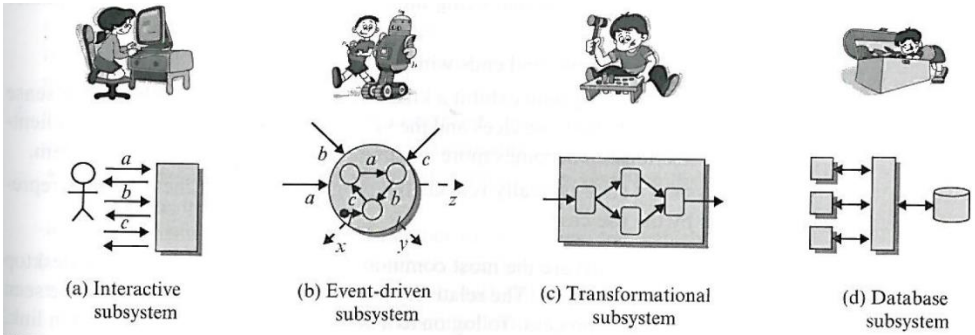
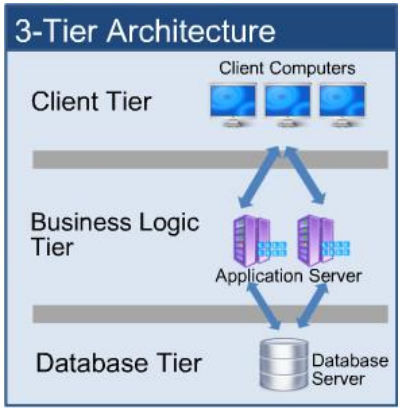


FIGURE 6.2 Four types of systems and behavior



Couplage et cohésion

- Pour un même produit, différents designs possibles
  - Compréhension?
  - Localisation des fautes?
  - Facilité à étendre ou améliorer?
  - Réutilisation dans un autre module/produit?

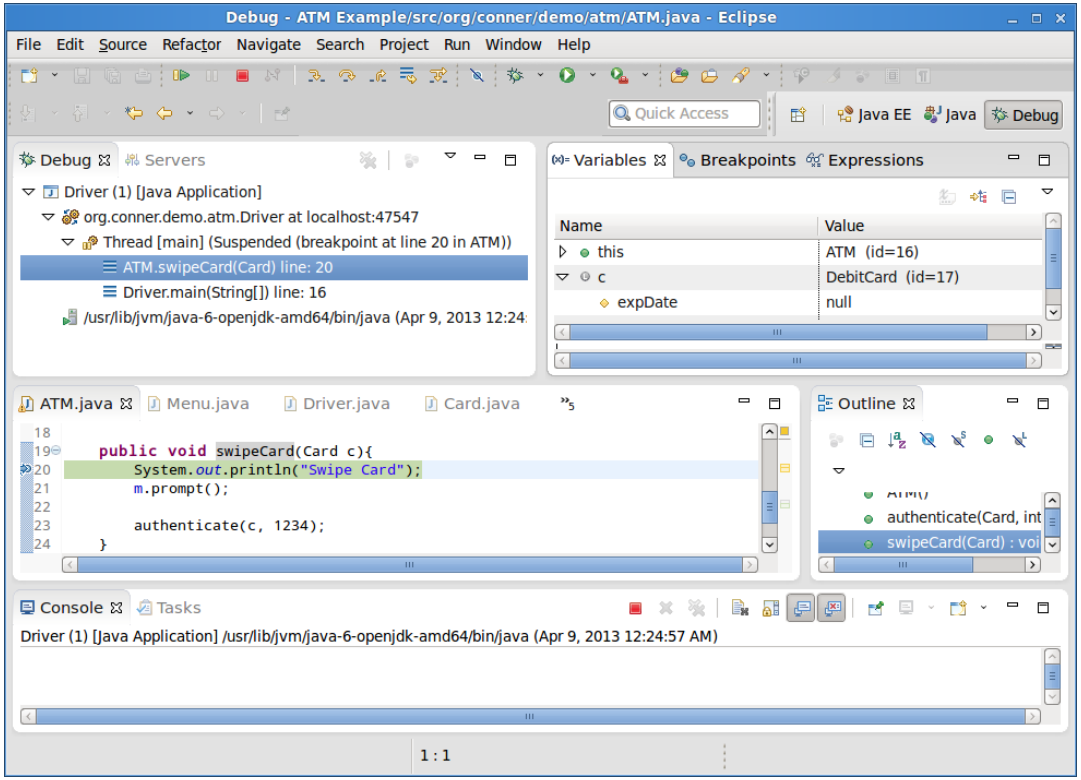
- **Cohésion** d'un module (à maximiser)
  - Degré d'interaction au sein du module
- **Couplage** d'un module (à minimiser)
  - Degré d'interaction entre les modules

Caractéristiques d'une bonne conception

- **Rigueur**
  - S'assure que toutes les exigences sont satisfaites
- **Séparation des préoccupations**
  - **Modularité**
    - Permet le travail isolé et parallèle: composants sont indépendants des autres
  - **Abstraction**
    - Permet le travail isolé et l'intégration: interfaces garantissent que les composants vont fonctionner ensemble
- **Anticipation du changement**
  - Permet d'absorber les changements sans effort
- **Généralisation**
  - Permet de réutiliser les composants à travers le système et d'autres systèmes
- **Incrémentalité**
  - Permet de développer le logiciel avec des résultats intermédiaires

Implémentation

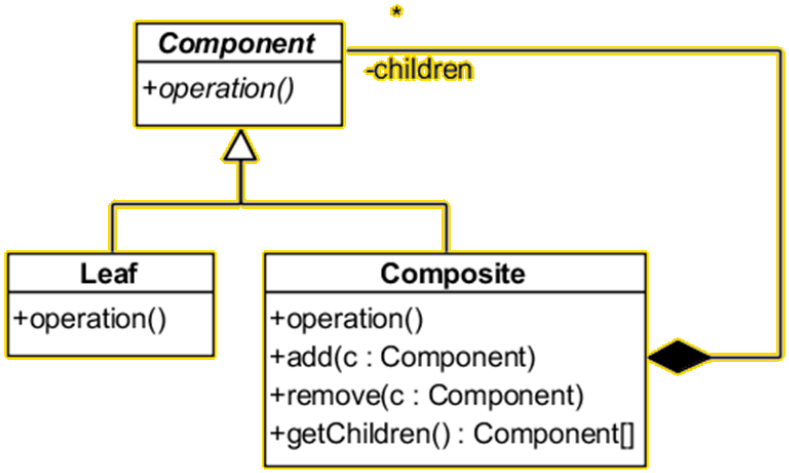
- Normes
- Choix du langage
- Techniques de débogage

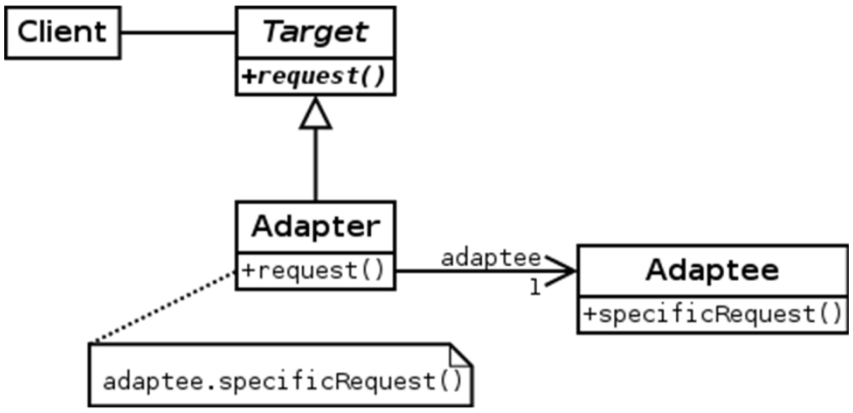



```java
package ATM;

public class ATM {

    private double balance;

    public double getBalance() {
        return balance;
    }

    public ATM(double initialBalance)
    {
        if (initialBalance > 0)
            this.balance = initialBalance;
        else
            this.balance = 0.0;
    }

    public boolean deposit(double amount)
    {
        if( amount < 0 )
            return false;
        else
            this.balance += amount;

        return true;
    }

    public boolean withdraw(double amount)
    {
        if( this.balance < amount )
            return false;
        else
            this.balance -= amount;

        return true;
    }

}
```

Patrons de conception

- Solution à un problème général récurrent
  - Promeut la réutilisation et facilite la maintenance
- Comment implémenter un patron de conception

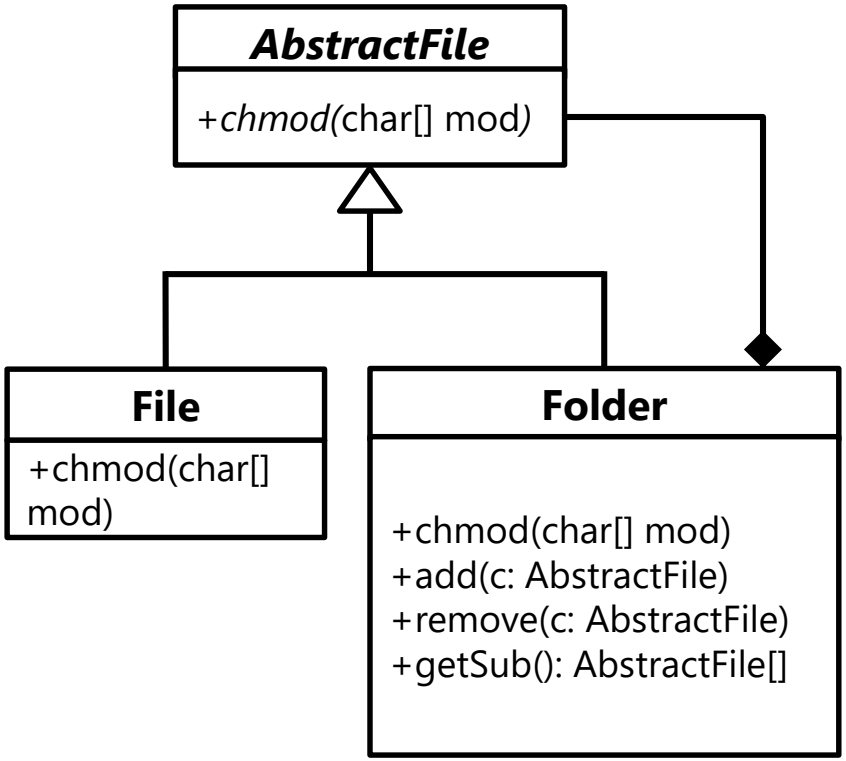




Patrons de conception

Implémentation dans un langage de programmation

```
class AbstractFile {
  public abstract void chmod(char[] mod);
  private void setOwner(char mod){ ownMod = mod; }
  private void setGroup(char mod){ groupMod = mod; }
  private void setAll(char mod){ allMod = mod; }
}
class File extends AbstractFile {
  public void chmod(char[] mod) {
      setOwner(mod[0]);
      setGroup(mod[1]);
      setAll(mod[2]);
  }
}
class Folder  extends AbstractFile {
  ArrayList<AbstractFile> sub = new ArrayList<AbstractFile>();
  public void add(c: AbstractFile) { … }
  public void remove(c: AbstractFile) { … }
  public void getSub { … }

  public void chmod(char[] mod) {
    setOwner(mod[0]);
    setGroup(mod[1]);
    setAll(mod[2]);
    foreach(af  AbstractFile : getChildren())
        af.chmod(mod);
  }
}
```



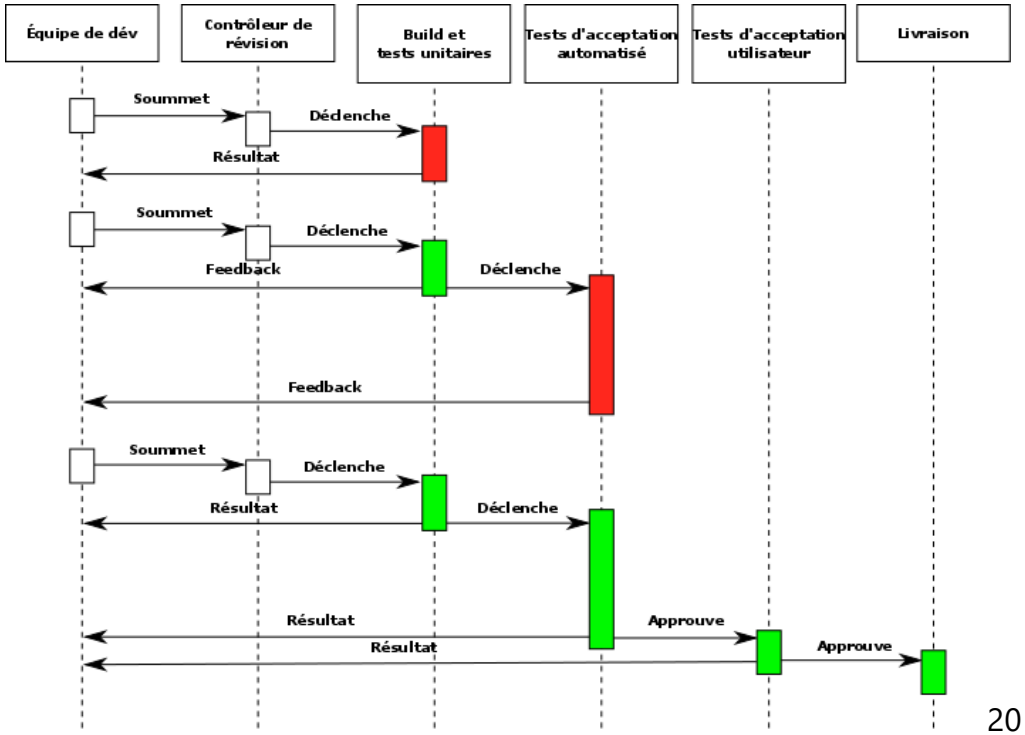
Vérification, test

- Qualité du logiciel
- Inspection
- Différents types de tests
- Tests structurels et fonctionnels
- Tests unitaires
  - Succès, échec, sanitaire
  - Couverture des cas de tests
    - Oracle

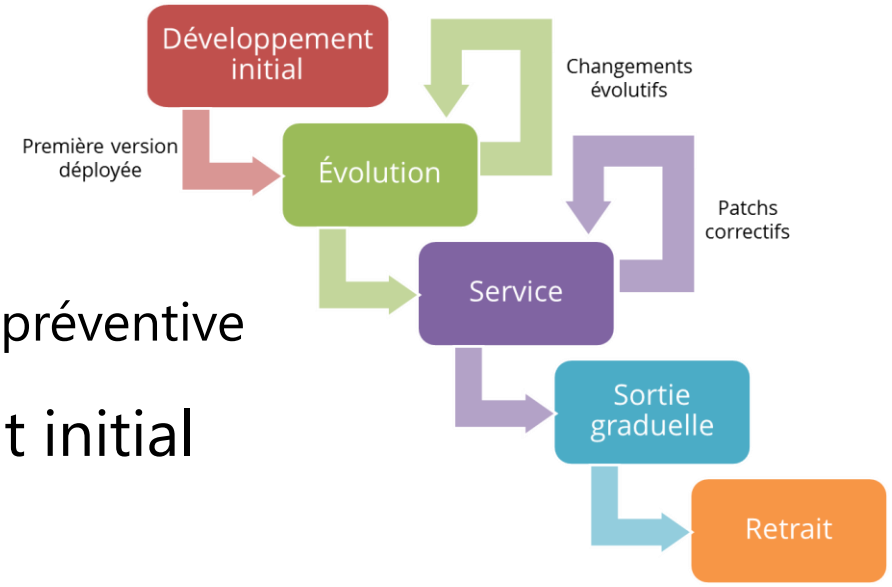
```java
package ATM;

import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class ATMTest {

    private ATM a;
    private double originalBalance;

    @Before
    public void setUp(){
        a = new ATM(200);
        originalBalance = a.getBalance();
    }

    @Test
    public void testDeposit() {
        double depositAmount = 200;

        a.deposit(depositAmount);

        assertEquals("Deposit amount is persisted", originalBalance + depositAmount, a
    }

    @Test
    public void testDepositFailure(){
        double depositAmount = -200;

        assertFalse("Cannot deposit negative amount", a.deposit(depositAmount));
    }

    @Test
    public void testWithdraw() {
        double withdrawAmount = 100;

        a.withdraw(withdrawAmount);

        assertEquals("Withdraw amount is persisted" + originalBalance,
            originalBalance - withdrawAmount, a.getBalance(), 0);
    }

    @Test
    public void testWithdrawSanity() {
        double amount = 100;
        a.deposit(amount);
        a.withdraw(amount);
        assertEquals("Deposit is the inverse of withdraw", originalBalance, a.getBalar
    }

    @Test
    public void testOverdraw(){
        double withdrawAmount = 1000;

        assertFalse("Cannot withdraw more than balance", a.withdraw(withdrawAmount));
    }
}
```

Déploiement

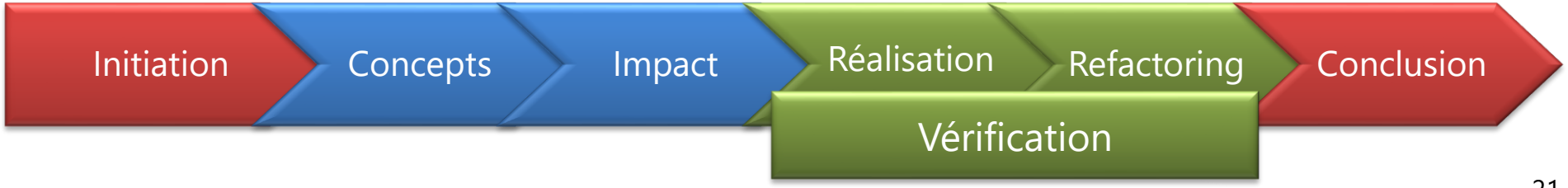
- Livraison de versions d'un logiciel
- Environnements de développement, de test, de production
- Activités de déploiement
  - Composants, assemblage, transfert, installation, activation
- Déploiement continue



Maintenance



- Types de maintenance
  - Corrective, perfective, adaptative, préventive
- Même cycle que développement initial
  - Plus courte durée
  - Répété pour chaque demande de changement / correction
- Refactoring
- Problème de maintenance inhérents au paradigme OO



Environnement logiciel utilisable

Propriétés essentielles d'utilisabilité
pour le développement de logiciels

- Unités encapsulées réutilisables
- Abstraction en patrons
- Utilisation de librairies
- Évolution de langage et rétrocompatibilité
- Performance et mise à l'échelle
- Suggestions et recommandations automatiques
- Contrôle de révisions
- Débogage
- Automatisation des tests
- Automatisation du déploiement

La suite…

Spécialisation en génie logiciel

- **IFT 3911: Analyse et conception des logiciels**
- IFT 3912: Développement, maintenance de logiciels
- IFT 3913: Qualité du logiciel et métriques
- IFT 3150: Projet d'informatique *(IFT 4055: Projet informatique honor)*
- IFT 2905: Interface personne-machine
- IFT 2935: Bases de données

- IFT6252: Méthodes empiriques en génie logiciel
- **IFT 6253: Conception dirigée par les modèles**
- IFT 6755: Analyse du logiciel

IFT 3911 : Analyse et conception des logiciels

- Emphase sur la conception approfondie
- Des exigences à la **conception**
- **Modélisation**
  - UML approfondie
  - Machines d'état
  - Réseaux de Pétri
- **Génération de code** et rétro-ingénierie
- **Patrons** de conception
- **Évaluation** qualitative de la conception
- **Analyse des propriétés** du système conçu

Conception dirigée par les modèles

- Modélisation à différents niveaux d'abstraction
- Création de langages de modélisation spécifiques au domaine
  - Méta-modélisation
  - Génération d'éditeurs de modèles
- Transformation de modèles
  - Translation, simulation, génération de code
- Synthèse d'applications complètes sans programmer