

A project process for UML

(Conception de systèmes avec UML)

Justin Templemore-Finlayson

justin.templemore@ece.fr

Bibliographie

- **Rumbaugh et al**, *Modélisation et conception orientées objet*
- **Fowler+Scott**, *Tout en Poche : UML*
- **Roques**, *UML par la pratique*
- Lai, *Penser objet avec UML et Java*
- Booch+Rumbaugh+Jacobson, *Le Guide de l'utilisateur UML*
- Larman, *UML et les Design Patterns*
- Eckel, *Penser en Java*
- Unilog, *Produits analyse et conception*
- Pierre-Alain Muller, *Modélisation objet avec UML*

Sommaire

- Approche objet vs approche classique
- Les métiers
- Architecture 4+1
- Maquette et prototype
- *Unified Process* (Processus unifié)

Sommaire

- **Approche objet vs approche classique**
- Les métiers
- Architecture 4+1
- Maquette et prototype
- *Unified Process* (Processus unifié)

Approche objet vs approche classique

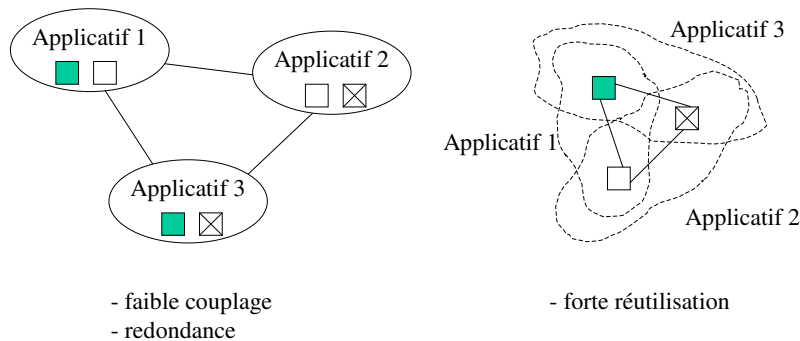
- Le but de l'approche objet est de concevoir le SI non plus comme une juxtaposition d'applicatifs à faible couplage,
- Mais comme un ensemble organisé de fonctionnalités pertinentes
 - non redondantes
 - fortement réutilisées
 - constituent la réponse à l'ensemble des besoins de l'entreprise.

Conception - 03 - Projet

5

Approche objet vs approche classique

- Plutôt que de chercher à définir l'action que vous visez, construisez l'objet qui exécutera cette action



Conception - 03 - Projet

6

Approche objet vs approche classique

Schéma de base d'un applicatif utilisant les classes *Comptable*, *Facture*, *Magasinier*, *Casier*, *Fichier article* et *Calcullette*

1. *Facture*, établis-toi, dit le *Comptable*.
 2. *Magasinier*, dis-moi quels sont les articles commandés, dit la *Facture*.
 3. *Casier d'article 1*, dis-moi combien d'unités de tu viens de sortir, dit le *Magasinier*.
 4. *Magasinier*, répondit le *Casier article 1*, j'ai sorti 5 pièces.
 5. ...
 6. *Facture*, voilà la réponse à ta question, dit le *Magasinier*.
 7. *Fichier article 1*, quel est le libellé et quel est le prix de l'article 1, dit la *Facture*.
 8. *Facture*, c'est un "Table Dum" à 20 euros l'unité, répond le *Fichier article 1*.
 9. ...
 10. *Calcullette*, quel est le prix de 5 "Table Dum" à 20 euros la pièce, continue la *Facture*.
 11. ...
 12. *Facture*, c'est 100 euros, répond la *Calcullette*.
- Et la facture fut !

Conception - 03 - Projet

7

Approche objet vs approche classique

- Un **objet** est un composant autonome et réactif, regroupant données et comportement qui lui permet de remplir les fonctions attendues de l'entité qu'il représente
- Il fonctionne aussi bien de façon unitaire, qu'en collaboration avec d'autres objets
 - *CompteCourant* maintient son solde à jour et peut communiquer avec *CompteEpargne* pour effectuer un virement.

Conception - 03 - Projet

8

Approche objet vs approche classique

- La création d'applicatif devient alors principalement une réutilisation d'objets existants.
- **Composition** : Réutilisation des attributs et/ou du comportement d'objets existants
 - Approche « lego »
 - E.g. : Création d'un applicatif "Guichet Transferts" à partir des classes *Client*, *CompteCourant*, *CompteEpargne* et *Transaction*
- **Héritage** : Réutilisation de la forme d'un objet existant, avec spécialisation de certains éléments
 - E.g. : Définition d'un nouveau type de compte *CompteCashBack* à base de *CompteCourant*.

Sommaire

- Approche objet vs approche classique
- **Les métiers**
- Architecture 4+1
- Maquette et prototype
- *Unified Process* (Processus unifié)

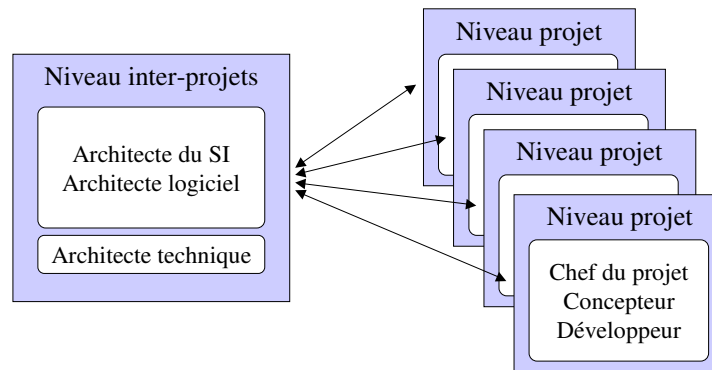
Les métiers

- L'importance de la réutilisation induit un redécoupage des métiers liés aux projets informatiques:
 - Au **niveau projet** on trouve les personnes chargées de la réalisation des applicatifs, les utilisateurs des classes du SI
 - Au **niveau inter-projets** on trouve les personnes garantes de la cohérence du SI, fournisseurs des classes du SI

Les métiers

- Il est impossible de confier le développement et l'évolution des classes du SI à une équipe projet:
 - La richesse de la bibliothèque de classes du SI peut constituer un frein au développement, l'effort pour en prendre conscience pouvant être rebutant
 - L'équipe projet peut n'avoir qu'une connaissance insuffisante de l'existant applicatif
 - L'équipe projet a un objectif à court terme de mis à disposition d'un applicatif, en conflit avec l'objectif moyen terme de développement de composants réutilisables du SI.

Les métiers



Niveau projet

- Le **chef de projet** a la responsabilité de la conformité des fournitures produites aux fournitures prévues :
 - il a en charge le pilotage des actions de conception et de développement du logiciel ;
 - il assure la planification détaillée et le suivi en charges et en délai du planning du projet ;
 - il dialogue avec les utilisateurs ;
 - il dialogue avec les intervenants inter-projets.
- Le **concepteur** réalise sous la responsabilité du chef de projet l'étude préalable ainsi que la conception du SI sur l'ensemble du domaine d'étude.

Niveau projet

- Le **développeur** conçoit et réalise les logiciels.
- Se contente de faire communiquer des classes déjà existant, sur lesquelles il n'a aucun droit d'intervention.
- Un programme applicatif n'est plus qu'une suite d'activation de méthodes et est le plus souvent totalement dépourvu d'instructions procédurales.
- Toute modification de classe doit avoir l'accord préalable l'équipe inter-projets.
- Le développeur peut seulement être amené à développer les classes considérées comme non réutilisables.

Niveau inter-projets

- **L'architecte du SI** a pour mission d'assurer la cohérence fonctionnelle des projets :
 - il tient à jour la cartographie du système d'information ;
 - il identifie, décrit et administre les classes et les données ;
 - il contrôle le respect des normes et standards garantissant la cohérence du système d'information ;
 - il définit les plans d'évolution du système d'information ;
 - il garantit la pérennité du système d'information.
- Son objectif, entre autres, est de constituer un langage commun pour l'entreprise et de dégager des classes invariantes.
- L'architecture du système d'information a pour objet de constituer le référentiel du SI de l'entreprise.

Niveau inter-projets

- **L'architecte logiciel** assure la pertinence, la cohérence et l'accessibilité des classes mises à la disposition des développeurs d'applicatifs et développe les classes réutilisables.
- Il a en charge
 - les **classes fonctionnelles** qui modélisent chacune une entité liée au métier de l'entreprise,
 - les **classes techniques** qui modélisent chacune une entité liée aux aspects techniques de l'automatisation, comme la fenêtre d'affichage.
- Également chargé de l'élaboration de la charte graphique.

Niveau inter-projets

- **L'architecte technique** conçoit et met en oeuvre des solutions techniques nouvelles, en garantissant la fiabilité et l'intégration dans les architectures existantes, et
- conduit les projets d'homologation des nouveaux produits.
- Il existe également d'autres métiers qui participent:
 - l'intégrateur, qui intègre et qualifie les différentes versions de logiciel développées par les équipes projet,
 - des intervenants méthode, qualité et outils,
 - l'administrateur de bases de données.

Sommaire

- Approche objet vs approche classique
- Les métiers
- **Architecture 4+1**
- Maquette et prototype
- *Unified Process* (Processus unifié)

Architecture 4+1

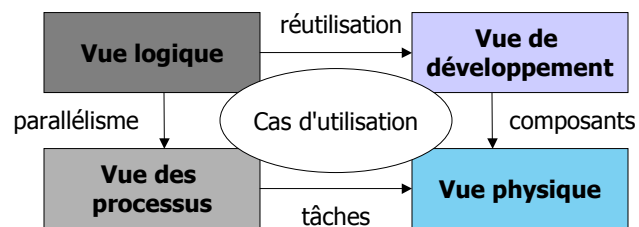
- L'architecture d'un système décrit sa forme.
- Les systèmes d'aujourd'hui sont trop larges, trop complexes pour que l'on puisse en décrire l'architecture dans un seul diagramme ou vue.
- **Kruchten*** propose un modèle de 4+1 vues concurrents pour mieux décrire les différents aspects de l'architecture d'un système.

*Phillipe Kruchten, *Le modèle 4+1 vues*, IEEE Software, novembre 1995.

Architecture 4+1

- **Vue logique**
 - Le modèle objet du système
- **Vues des processus**
 - Identification des tâches; synchronisation
- **Vue de développement**
 - Organisation du modèle logique en composants réutilisable
- **Vue physique**
 - Description de l'architecture matérielle et l'implantation des composants et tâches

Architecture 4+1



- Chaque vue présente un sous ensemble d'éléments sélectionnés de l'architecture du SI.
- Ils sont concurrents et obligatoirement cohérent.
- Les cas d'utilisation sont la colle entre les 4 vues de l'architecture

Cas d'utilisation

- Les cas d'utilisation sont utilisés pour exprimer les exigences fonctionnelles d'un SI.
- Les 4 vues de l'architecture sont conçues avec l'objectif de soutenir ces besoins fonctionnels.
- Notamment, les cas d'utilisation sont utilisés comme référence pour :
 - motiver les décisions prises lors de l'élaboration de l'architecture;
 - illustrer les relations entre les vues; et
 - valider l'architecture développée (test).
- Les **descriptions des cas** décrivent les utilisateurs (*acteurs*) du système et les fonctions (cas) exigés.

Architecture logique

- La vue logique exprime les abstractions clés du domaine du problème sous forme de classes et d'objets communicants.
- Un **diagramme de classes** montrent des classes et des relations entre eux.
- Un **diagramme d'états-transitions** décrit le comportement interne d'une classe d'objets.
- Un **diagramme d'interaction** (collaboration ou séquence) montrent la façon dont un sous-ensemble des objets collaborent pour réaliser une fonctionnalité (cas).

Architecture des processus

- On définit:
 - **tache**: un flot de contrôle (*thread*) qui exécute des activités sur un nœud physique.
 - **processus**: un ensemble de taches qui forme une unité d'exécution.
- La vue des processus
 - identifie les tâches du SI et la répartition des éléments de l'architecture logique entre ces tâches
 - spécifie la parallélisme et synchronisation des tâches au sein d'un processus
- Un **diagramme de tâches** est un diagramme de classes stéréotypé ne montrant que les *tâches* (classes actives) et leurs relations.
- Un **diagramme d'activités** permet de décrire un processus, avec la séquence des planification des activités de chaque tâche.

Architecture de développement

- Décrit l'organisation/décomposition du système dans l'environnement de développement.
- Un logiciel complexe sera décomposé en une hiérarchie de composants (programmes principaux, sous-programmes, BDDs, processus distribués, bibliothèques, ...)
 - chaque composant regroupe un sous-ensemble des éléments du modèle logique
- Objectif: Modéliser un système comme un ensemble de composants réutilisables et assembler le système à partir de composants existants.
- Un **diagramme de composants** permet de décrire ces composants et leurs relations de dépendance.

Architecture physique

- Un SI consiste normalement de plusieurs composants implanté dans une architecture matérielle distribuée.
- L'architecture physique décrit les ressources matérielles, leur interconnexion et la répartition des composants et tâches du SI dans cette déploiement.
- Un **diagramme de déploiement** permet de décrire les ressources matérielles (nœuds) et leurs connexions.

Architecture 4+1

- Notez que toutes les vues ne sont pas exigés. On n'utilise que ce dont on a besoin pour le SI sous étude.
- La vue logique est favorisé pendant la phase d'analyse; les autres vue coulent de celle-ci pendant la phase de conception.

Document d'architecture

Titre

Histoire de changements

Table de matières

Table d'illustrations

1. Contexte
2. Références
3. Objectifs et contraintes
4. Cas d'utilisation
5. Architecture
 - 5.1 Architecture logique
 - 5.2 Architecture des processus
 - 5.3 Architecture de développement
 - 5.4 Architecture physique
- A. Glossaire et dictionnaire

Sommaire

- Approche objet vs approche classique
- Les métiers
- Architecture 4+1
- **Maquette et prototype**
- *Unified Process* (Processus unifié)

Maquette et prototype

- Définitions normalisées par l'ISO
 - "La **maquette** est un livrable jetable qui concrétise les spécifications dans un environnement technique qui peut être différent de l'environnement de production."
 - "Le **prototype** est un livrable réutilisable qui met en oeuvre l'exhaustivité de certaines fonctionnalités dans un environnement technique identique à l'environnement de production."
- On distingue trois principaux types de maquettes ou de prototypes :
 - validation de l'ergonomie de l'application ;
 - vérification de la pertinence des choix techniques ;
 - validation des fonctionnalités de l'application.

Maquette ou prototype ?

- Le choix entre maquette et prototype est fonction, entre autres, du niveau d'avancement dans le projet.
- On aura tendance à privilégier la maquette lorsque les choix sont encore peu assurés puisqu'elle ne nécessite que peu d'effort de développement.
- On préférera le prototype lorsque l'avancée du projet donne de bonnes chances aux possibilités de réutilisation des livrables intermédiaires.

Sommaire

- Approche objet vs approche classique
- Les métiers
- Architecture 4+1
- Maquette et prototype
- ***Unified Process (Processus unifié)***

Processus de développement

- L'objet est très adapté aux types de cycles de vie moderne grâce à :
 - **la puissance** des IDE et des 4GL dont la logique interne est à base d'objets.
 - Utilisent et fournissent objets réutilisable, e.g. Fenêtre.
 - **sa facilité d'évolution** - les classes se prêtent aisément à un codage général des fonctionnalités qui seront détaillées ultérieurement.
 - La livraison de prototypes intermédiaire et réutilisable est donc simple à mettre en oeuvre.
 - L'affinement ultérieur se fait par ajout de membres aux classes initiales, ou par spécialisation en utilisant l'héritage.

Processus de développement

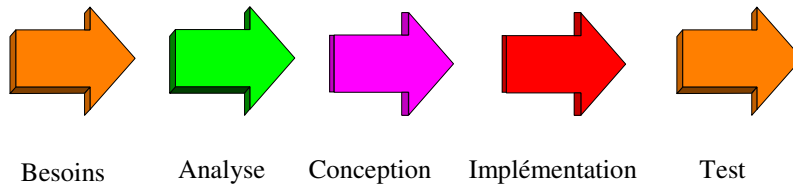
- Le **processus unifié** est un processus de développement logiciel.
- Un processus de développement définit une séquence d'étapes pour obtenir un système logiciel ou faire évoluer un système existant.
- Son but est de guider une équipe de projet dans la production d'un logiciel de haute-qualité qui réponde aux exigences utilisateurs dans un coût et un délai prévisible, avec un minimum de risque.

Unified Process

- Le **processus unifié** est un processus de développement :
 - **construit sur UML** : graphique, orienté-objet;
 - **piloté par les cas d'utilisation** : le système développé répond aux exigences de l'utilisateur;
 - **centré sur l'architecture** : l'architecture de référence structure l'implémentation et réduit les risques
 - **itératif et incrémental** : on construit le produit petit à petit, par ajouts successif de fonctionnalités
 - **gérer par les risques** : les risques fonctionnelles et techniques sont levés au plus tôt

Unified Process

- A chaque itération, on identifie et spécifie les cas d'utilisation pertinents à développer et on enchaînera les activités du processus:



Conception - 03 - Projet

37

Piloté par les cas d'utilisation

- Le pilotage par les cas d'utilisation permet d'assurer la conformité du logiciel aux besoins utilisateurs et garanti la prise en compte et la gestion des exigences utilisateurs de façon structurée.
- Les cas d'utilisation pilotent le processus de développement et encadre les modèles d'architecture et ainsi garantissent la cohérence du système.
- A chaque itération, ce sont les cas d'utilisation qui pilotent les enchaînements des différents modèles produits.

Conception - 03 - Projet

38

Centré sur l'architecture

- Le processus unifié recherche la forme générale d'un système dès le début.
 - le code n'est qu'un "effet secondaire"
- L'architecture du produit logiciel dépend de deux facteurs :
 - des contraintes fonctionnelles des utilisateurs (exigences), et
 - des contraintes techniques liées à l'environnement d'implémentation.
- Dans le processus unifié, ces contraintes sont adressés dès le début par la spécification détaillé de l'architecture du système.
 - cela réduit le risque d'échec pendant la phase d'implémentation

Centré sur l'architecture

- Une bonne architecture :
 - Facilite l'implémentation en parallèle ;
 - en identifiant les composants et fixant leurs interfaces
 - Réduit le travail de révision en cas de changements ;
 - par la réduction de l'inter-dépendance de composants
 - Utilise et produit des composants ré-utilisable ;
 - Présente un documentation compréhensible du système;
 - Réalise les exigences fonctionnelles du client;
 - Réduit les risques d'échec.

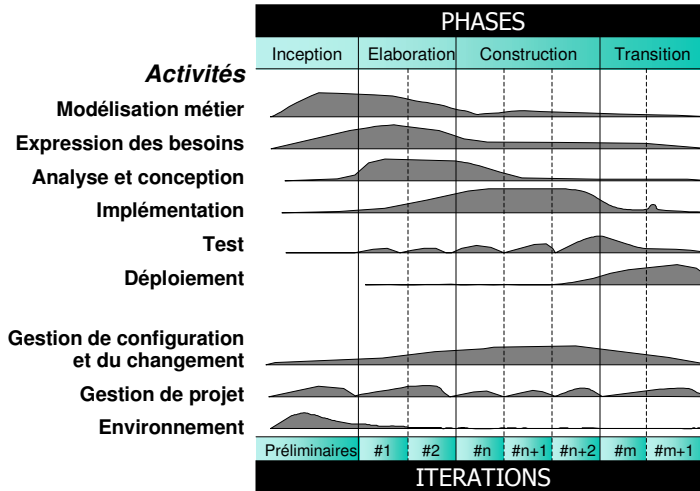
Itératif et incrémental

- L'idée est de faire un découpe fonctionnel du processus de développement.
 - Une itération correspond à un "sous travail" et produit un code exécutable
- A chaque itération, on identifie et spécifie les cas d'utilisation pertinents à développer et on enchaîne les activités (analyse, conception, implémentation, test) du processus.
- Les itérations sont planifiées à partir des cas d'utilisation selon deux critères :
 - l'intérêt de l'incrément livré en fonction des pré-conditions entre les cas d'utilisation,
 - la réalisation prioritaire des cas d'utilisation permettant d'éliminer les risques.

Itératif et incrémental

- Chaque itération produit un code exécutable sous forme d'un composant correspondant aux cas d'utilisation sélectionnés pour cette itération.
- La forme de l'exécutable dépend de la phase d'avancement du projet:
 - maquette, prototype, sous-système ou bien produit final
 - Chaque nouvel incrément livré réutilise et enrichit progressivement le système.
- L'approche itérative encourage la participation des utilisateurs finales, qui sont présentés avec un exécutable à chaque itération.
 - Ceci réduit évidemment la probabilité d'échec.

Unified Process



Conception - 03 - Projet

43

Unified Process

- La vie d'un logiciel est divisé en plusieurs cycles de développement, chacun produisant une nouvelle génération (version majeur) du produit.
- UP divise un cycle en phases. Chaque phase a un but précis et termine par l'arrivée à un jalon important.
- Une phase peut définir et planifier plusieurs itérations à partir des cas d'utilisation.
- A chaque itération on enchaînera les activités du processus.
- L'importance de l'activité dans une itération dépend de la phase d'avancement du cycle.

Conception - 03 - Projet

44

Modélisation métier

- C'est la modélisation du domaine entreprise indépendamment de l'application.
- Il s'agit de décrire le domaine dans lequel fonctionne un entreprise (*Domain Model*) ou de modéliser des procédures en vigueur dans l'entreprise (*Business Model*).
- Ces modèles permet à l'ingénieur de mieux comprendre le domaine et ses activités - le contexte dans lequel le système va devoir fonctionner – avant de s'intéresser aux fonctionnalités du système.
- On utilise un ensemble de diagrammes UML stéréotypé.

Exigences

- C'est ici que l'on modélise les exigences utilisateurs sous forme de cas d'utilisation.
- Un *cas d'utilisation* décrit une fonctionnalité que le SI fournira à un utilisateur ou *acteur*.
- Chaque cas décrit les variations de comportement désirables entre les acteurs et le système.

Analyse

- Partant des exigences, l'analyste construit un architecture logique.
- Le modèle d'analyse est un abstraction précise et concise du but de l'application, non la façon dont elle sera bâtie.
- Les objets identifiées sont les concepts du domaine de l'application, non des objets informatiques telles que les structures de données.
- Un bon modèle d'analyse est lisible par les experts du domaine du problème (non-informaticiens).

Conception

- Le concepteur reprend le modèle d'analyse et y impose les contraintes techniques de l'implémentation:
 - l'architecture logique est découpée en sous-systèmes et sous-tâches,
 - une architecture physique est proposée.
 - on décide les caractéristiques de performance et d'optimisation, la stratégie d'implémentation, et l'allocation de ressources.
 - on détail les classes en ajoutant les objets de soutien purement informatique
 - arbres, tables d'adressage, plateformes de communication, etc.
- On fait usage des *Design Patterns* pour résoudre les problèmes récurrents.

Implémentation

- Les objectifs de l'implémentation sont:
 - La création des composants de l'architecture (programmes, sous-programmes, fichiers aides, bases de données, processeurs embarqués...)
 - L'intégration des composants développés.
- Bien qu'important en termes de ressources, cette étape devrait être relativement mécanique puisque toutes les décisions difficiles ont été prises pendant les activités d'analyse et de conception.

Test

- Les objectifs de test sont:
 - Testes unitaire (teste d'un composant en isolation)
 - Testes d'intégration;
 - Vérifier que les exigences sont satisfaites;
 - Identifier toute bogue avant de déployer.
- Les cas d'utilisation qui définissent des cas de test.

Déploiement

- L'objectif est de déployer l'application et de tester l'application dans l'environnement utilisateur.
- Il comprend:
 - Livraison du version exécutable du logiciel;
 - Production d'un guide utilisateur;
 - L'intégration avec systèmes existants;
 - Formation des utilisateurs;
 - Planification et exécution des tests "beta".
- C'est aussi ici que l'on collectionne des informations qui permettront de décider de itérer su une nouvelle génération du logiciel.

Activités "de soutien"

- Gestion de configuration et changements
 - Contrôle de versions, synchronisation de développement en parallèle et distribué, maintenance d'intégrité des artefacts
- Gestion de projet
 - Planification des activités, allocation des ressources humaines, documentation de décisions prises
- Gestion de l'environnement
 - mis à disposition de ressources logiciels et matérielles, savoir-faire, formation

Phases

- RUP identifie quatre *phases* consécutives dans un cycle de développement:



- Chaque phase termine avec une connaissance plus approfondie du système (la fin de la phase de transition correspondant à la disponibilité du système).
- Chaque phase a un but précis – un ensemble d'objectifs à atteindre et des artefacts à produire.
- La phase termine avec un *jalon* important, où la décision de passer à la phase suivante (ou non) est prise.

Phase d'inception



- Définition du cadre du projet
 - Critères du succès;
 - Mise en évidence des risques;
 - Estimation des ressources nécessaires;
 - Identification de la situation du système dans le entreprise/domaine
 - Planning des phases avec dates des principaux jalons;
- Artefacts produits:
 - Un modèle de cas d'utilisation initial (10% à 20% complet)
 - Un glossaire
 - Planning montrant phases et itérations (diagrammes d'activités)
 - Éventuellement un *Domaine Model* et/ou *Business Model*

Phase d'élaboration



- Établissement d'un plan du projet et d'une architecture solide
- Exigences capturées et classées par ordre de priorité
 - les cas d'utilisation
- Description des différentes vues de l'architecture
 - logique, processus, développement, physique
- C'est la partie "dur" de l'ingénierie du système
- La phase termine avec la décision de procéder (ou non) avec la réalisation du système.
- Livraison de
 - Un modèle de cas d'utilisation (>80%)
 - Un document de l'architecture du logiciel
 - Un prototype exécutable
 - Un plan des activités de construction

Phase de construction

010110010
101000100
010100010
001001011
010001001
010101010
010010101
101010101

- La procédure de fabrication d'une implémentation prête à être déployée chez l'utilisateur.
- Activités:
 - Réalisation des composants du système
 - Test des composants individuels.
 - Test et intégration des composants.
 - Création du guide utilisateur pour la version actuelle.
- Les composants sont réalisés d'une façon itérative qui comprend le test unitaire.
- Les itérations peuvent être mené séquentiellement ou (mieux) en parallèle.

Phase de transition



- Fournir le système aux utilisateurs finals
 - Déploiement de la version *beta*;
 - Tests d'intégration avec d'autres systèmes existants;
 - Détection et correction de bogues;
 - Ajout de nouvelles caractéristiques absentes de la version précédente;
 - Déploiement de la version finale;
 - Formation des utilisateurs.