UML
1/5 – Contexte, méthodes et processus

Aurélien Tabard
Département Informatique
Université Claude Bernard Lyon 1
2014

Basé sur le cours de Yannick Prié

Aspects pratiques

Aurélien Tabard - http://tabard.fr/cours.html

4 Cours sur UML :

- .: Aujourd'hui
- .: Vendredi 21 novembre
- .: Lundi 24 novembre
- .: Vendredi 28 novembre

4 TPs en décembre

Un examen de 45 minutes le 17 décembre à 14h

Objectifs du cours global

Notions de méthodes de conception
de Système Informatique (SI)

Plan des 4 séances :

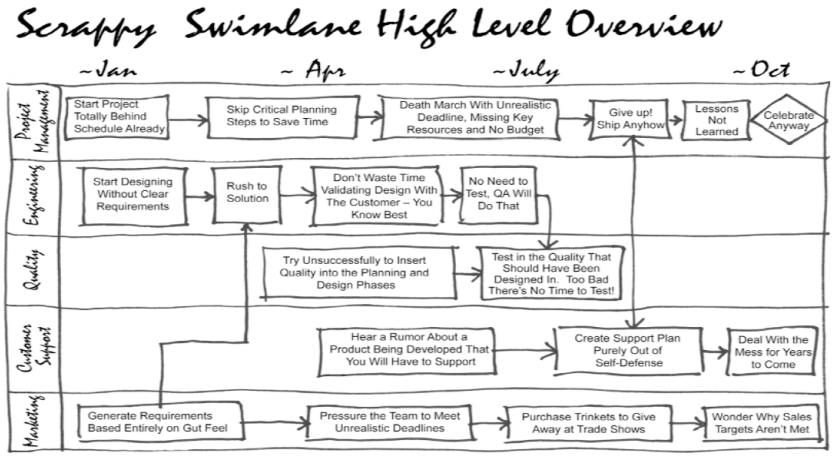
- .: Généralités sur les méthodes
- .: Processus unifié
- .: UML : Cas d'utilisation
- .: UML : Diagrammes de séquence
- .: UML : Diagrammes d'état
- .: Processus agiles

Plan de ce cours

Avant-propos
Génie logiciel
Méthodes
Activités
Outils
Documentation de projet

Ce qu'il faut éviter

(Kimberly Wiefling)



La chose à retenir

1. Les systèmes informatique sont complexes et demandent des méthodes de conception pour être menés à bien.

2. Ces méthodes doivent combiner rigueur et adaptation à l'inconnu et au changement.

UML : No silver bullet

- .: Connaître UML n'est pas suffisant pour réaliser de bonnes conceptions
  - .: UML n'est qu'un langage
- .: Il faut en plus
  - .: savoir penser / coder en termes d'objets
  - .: maîtriser des techniques de conception et de programmation objet
- .: Méthodes de conception
  - .: propositions de cheminements à suivre pour concevoir
  - .: pas de méthode ultime non plus
    - .: certains bon principes se retrouvent cependant partout

(B. Morand)

Ce qu'il faut aimer pour arriver à concevoir

- .: Être à l'écoute du monde extérieur
- .: Dialoguer et communiquer avec les gens qui utiliseront le système
- .: Observer et expérimenter : une conception n'est jamais bonne du premier coup
- .: Travailler sans filet : en général, il y a très peu de recettes toutes faites
- .: Abstraire
- .: Travailler à plusieurs : un projet n'est jamais réalisé tout seul
- .: Aller au résultat : le client doit être satisfait, il y a des enjeux financiers

Avertissement

.: Beaucoup de concepts dans ce cours
  .: proviennent du domaine du développement logiciel
    .: ancien (plusieurs décennies)
    .: plus récent
.: Tout l'enjeu est de comprendre
  .: ce qu'ils décrivent / signifient
  .: comment ils s'articulent
.: Méthode
  .: construire petit à petit une compréhension globale
    .: lire et relire
    .: chercher de l'information par soi même
    .: poser des questions
    .: pratiquer

Plan

(O. Boissier)

Génie logiciel

Définition

.: ensemble de méthodes, techniques et outils pour la production et la maintenance de composants logiciels de qualité

Pourquoi ?

.: logiciels de plus en plus gros, technologies en évolution, architectures multiples

Principes

.: rigueur et formalisation, séparation des problèmes, modularité, abstraction, prévision du changement, généricité, incréments

(O. Boissier)

Qualité du logiciel (1)

Facteur externes (usages)

.: Correction (validité)
  .: aptitude à répondre aux besoins et à remplir les fonctions définies dans le cahier des charges
.: Utilisabilité
  .: facilité d'apprentissage et d'utilisation, de préparation des données, de correction des erreurs d'utilisation, d'interprétation des retours
.: Robustesse (fiabilité)
  .: aptitude à fonctionner dans des conditions non prévues au cahier des charges, éventuellement anormales

(O. Boissier)

Qualité du logiciel (2)

Facteurs externes (environnement)

- .: Compatibilité
  - .: facilité avec laquelle un logiciel peut être combiné avec d'autres
- .: Extensibilité
  - .: facilité avec laquelle de nouvelles fonctionnalités peuvent être ajoutées
- .: Efficacité
  - .: utilisation optimale des ressources matérielles (processeur, mémoires, réseau, …)
- .: Intégrité (sécurité)
  - .: aptitude d'un logiciel à se protéger contre des accès non autorisés.

(O. Boissier)

Qualité du logiciel (3)

Facteurs internes (concepteur)

- .: Ré-utilisabilité
  - .: aptitude d'un logiciel à être réutilisé, en tout ou en partie, pour d'autres applications
- .: Vérifiabilité
  - .: aptitude d'un logiciel à être testé (optimisation de la préparation et de la vérification des jeux d'essai)
- .: Portabilité
  - .: aptitude d'un logiciel à être transféré dans des environnements logiciels et matériels différents
- .: Lisibilité
- .: Modularité

(O. Boissier)

Projet en général (1/2)

Définition

- .: ensemble d'actions à entreprendre afin de répondre à un besoin défini (avec une qualité suffisante), dans un délai fixé, mobilisant des ressources humaines et matérielles, possédant un coût.

Maître d'ouvrage

- .: personne physique ou morale propriétaire de l'ouvrage. Il détermine les objectifs, le budget et les délais de réalisation.

Maître d'oeuvre

- .: personne physique ou morale qui reçoit mission du maître d'ouvrage pour assurer la conception et la réalisation de l'ouvrage.

(O. Boissier)

Projet en général (2/2)

Conduite de projet

- .: organisation méthodologique mise en oeuvre pour faire en sorte que l'ouvrage réalisé par le maître d'oeuvre réponde aux attentes du maître d'ouvrage dans les contraintes de délai, coût et qualité.

Direction de projet

- .: gestion des hommes : organisation, communication, animation
- .: gestion technique : objectifs, méthode, qualité
- .: gestion de moyens : planification, contrôle, coûts, délais
- .: prise de décision : analyse, reporting, synthèse

Projet logiciel

Problème

.: comment piloter un projet de développement logiciel ?

Solution

.: définir et utiliser des méthodes

.: spécifiant des processus de développement

.: organisant les activités du projet

.: définissant les artefacts du projet

.: se basant sur des modèles

(O. Boissier)

Cycle de vie d'un logiciel

Cycle de vie d'un logiciel

.: débute avec la spécification et s'achève sur les phases d'exploitation et de maintenance

Modèles de cycle de vie

.: organiser les différentes phases du cycle de vie pour l'obtention d'un logiciel fiable, adaptable et efficace

.: guider le développeur dans ses activités techniques

.: fournir des moyens pour gérer le développement et la maintenance

.: ressources, délais, avancement, etc.

Deux types principaux de modèles

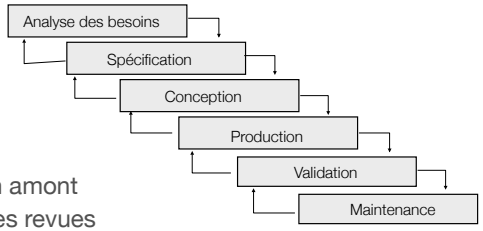
.: Modèle linéaires

.: en cascade et variantes

.: Modèles non linéaires

.: en spirale, incrémentaux, itératifs

Modèle en cascade



Années 70

Linéaire, flot descendant

Retour limité à une phase en amont

Validation des phases par des revues

Échecs majeurs sur de gros systèmes

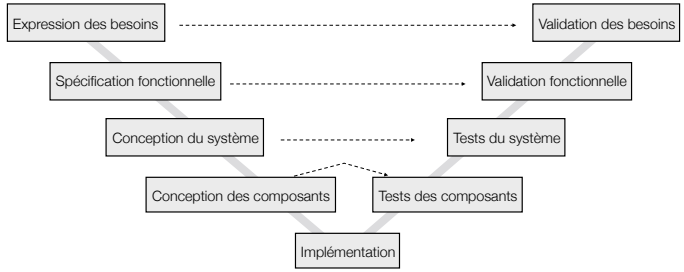
.: délais longs pour voir quelque chose qui tourne

.: test de l'application globale uniquement à la fin

.: difficulté de définir tous les besoins au début du projet

Bien adapté lorsque les besoins sont clairement identifiés et stables

Modèle en V



Variante du modèle en cascade

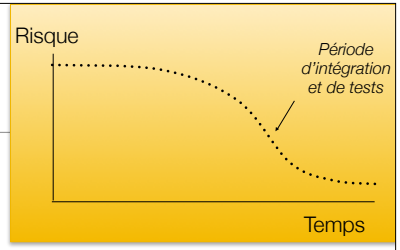
Tests bien structurés

Hiérarchisation du système (composants)

Validation finale trop tardive (très coûteuse s'il y a des erreurs)

Variante : W (validation d'un maquette avant conception)

Les problèmes du cycle en cascade



.: Risques élevés et non contrôlés
  .: identification tardive des problèmes
  .: preuve tardive de bon fonctionnement
.: Grand laps de temps entre début de fabrication et sortie du produit
.: Décisions stratégiques prise au moment où le système est le moins bien connu
.: Non-prise en compte de l'évolution des besoins pendant le cycle

Échecs des cycles en cascades

.: 25 % des exigences d'un projet type sont modifiées (35-50 % pour les gros projet) (Larman 2005)
.: 45% de fonctionnalités spécifiées ne sont jamais utilisées (Larman 2005, citant une étude 2002 sur des milliers de projets)
.: le développement d'un nouveau produit informatique n'est pas une activité prévisible ou de production de masse
.: la stabilité des spécifications est une illusion
.: Distinction entre activités trop stricte
.: modèle théoriquement parfait, mais inadapté aux humains

Anti-cascade

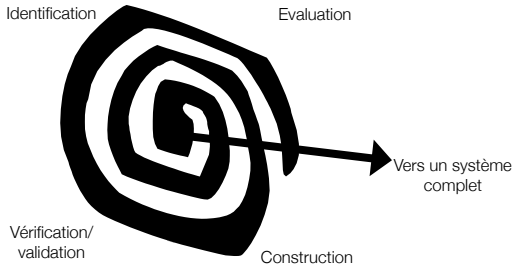
Nécessité de reconnaître que le changement est une constante (normale) des projets logiciels

  .: feedback et adaptation : décision tout au long du processus
  .: convergence vers un système satisfaisant

Idées

  .: construction du système par incréments
  .: gestion des risques
  .: passage d'une culture produit à une culture projet
  .: souplesse de la démarche

Modèle en spirale



Incréments successifs → itérations
Approche souvent à base de prototypes
Nécessite de bien spécifier les incréments
Figement progressif de l'application
Gestion de projet pas évidente

Les méthodes objet en dérivent

Plan de ce cours

Avant-propos
Génie logiciel
Méthodes
Activités
Outils
Documentation de projet

Qu'est ce qu'une méthode ?

Définitions
- .: guide plus ou moins formalisé
- .: démarche reproductible permettant d'obtenir des solutions fiables à un problème donné

Capitalise
- .: l'expérience de projets antérieurs
- .: les règles dans le domaine du problème

Une méthode définit
- .: des concepts de modélisation (obtenir des modèles à partir d'éléments de modélisation, sous un angle particulier, représenter les modèles de façon graphique)
- .: une chronologie des activités (construction de modèles)
- .: un ensemble de règles et de conseils pour tous les participants

Description d'une méthode
- .: des gens, des activités, des résultats

Méthodes en génie logiciel

Grandes classes de méthodes (Bézivin)
- .: méthodes pour l'organisation stratégique
- .: méthodes de développement
- .: méthodes de conduite de projet
- .: méthodes d'assurance et de contrôle qualité

Méthodes de développement pour
- .: construire des systèmes opérationnels
- .: organiser le travail dans le projet
- .: gérer le cycle de vie complet
- .: gérer les coûts
- .: gérer les risques
- .: obtenir de manière répétitive des produits de qualité constante

Processus

Pour décrire qui fait quoi à quel moment et de quelle façon pour atteindre un certain objectif

Définition
- .: ensemble de directives et jeu partiellement ordonné d'activités (d'étapes) destinées à produire des logiciels de manière contrôlée et reproductible, avec des coûts prévisibles, présentant un ensemble de bonnes pratiques autorisées par l'état de l'art

Deux axes
- .: développement technique
- .: gestion du développement

Artefacts d'un projet

Tout élément d'information utilisé ou généré pendant un cycle de développement d'un système logiciel

.: morceau de code, commentaire, spécification statique d'une classe, spécification comportementale d'une classe, jeu de test, programme de test, interview d'un utilisateur potentiel du système, description du contexte d'installation matériel, diagramme d'une architecture globale, prototype, rapport de réalisation, modèle de dialogue, rapport de qualimétrie, manuel utilisateur…

Notation et artefacts

Notation = formalisme graphique de représentation (ex: UML)

.: pour représenter de façon uniforme l'ensemble des artefacts logiciels produits ou utilisés pendant le cycle de développement

.: pour faciliter la communication, l'organisation et la vérification

Méthode / processus

.: types d'artefacts + notation + démarche (+ outils)

.: façon de modéliser et façon de travailler

Évolution des méthodes

4 vagues :

.: Méthodes de modélisation par les fonctions

.: Méthodes de modélisation par les données

.: Méthodes de modélisation orientée-objet

.: Méthodes agiles

1ère génération
Modélisation par les fonctions

Approche dite « cartésienne »

Décomposition d'un problème en sous-problèmes

Analyse fonctionnelle hiérarchique : fonctions et sous-fonctions

.: avec fonctions entrées, sorties, contrôles (proche du fonctionnement de la machine)

.: les fonctions contrôlent la structure : si la fonction bouge, tout bouge

.: données non centralisées

Méthodes de programmation structurée

.: IDEF0 puis SADT

Points faibles

.: focus sur fonctions en oubliant les données, règles de décomposition non explicitées, réutilisation hasardeuse

2ème génération
Modélisation par les données

Approches dites « systémiques »

SI = structure + comportement

Modélisation des données et des traitements

.: privilégie les flots de données et les relations entre structures de données (apparition des SGBD)

.: traitements = transformations de données dans un flux (notion de processus)

Exemple : MERISE

.: plusieurs niveaux d'abstraction

.: plusieurs modèles

Points forts

.: cohérence des données, niveaux d'abstraction bien définis.

Points faibles

.: manque de cohérence entre données et traitements, faiblesse de la modélisation de traitement (mélange de contraintes et de contrôles), cycles de développement trop figés (cascade)

génération actuelle
Modélisation orientée-objet

Mutation due au changement de la nature des logiciels

.: gestion > bureautique, télécommunications

Approche « systémique » avec grande cohérence données/traitements

Système

.: ensemble d'objets qui collaborent

.: considérés de façon statique (ce que le système est : données) et dynamique (ce que le système fait : fonctions)

.: évolution fonctionnelle possible sans remise en cause de la structure statique du logiciel

Démarche

.: passer du monde des objets (du discours) à celui de l'application en complétant des modèles (pas de transfert d'un modèle à l'autre)

.: à la fois ascendante et descendante, récursive, encapsulation

.: abstraction forte

.: orientée vers la réutilisation : notion de composants, modularité, extensibilité, adaptabilité (objets du monde), souples

génération émergente (depuis 2000)
Méthodes agiles

Méthodes adaptatives (vs. prédictives)

.: itérations courtes

.: lien fort avec le client

.: fixer les délais et les coûts, mais pas la portée

Insistance sur les hommes

.: les programmeurs sont des spécialistes, et pas des unités interchangeables

.: attention à la communication humaine

.: équipes auto-organisées

Processus auto-adaptatif

.: révision du processus à chaque itération

Plan de ce cours

Développement logiciel et activités

Cinq grandes activités qui ont émergé de la pratique et des projets

- .: spécification des besoins
- .: analyse
- .: conception
- .: implémentation
- .: tests

Activités : spécification des besoins

Fondamentale mais difficile

Règle d'or

- .: les informaticiens sont au service du client, et pas l'inverse

Deux types d'exigences

- .: Exigences fonctionnelles
  - .: à quoi sert le système
  - .: ce qu'il doit faire
- .: Exigences non fonctionnelles
  - .: performance, sûreté, portabilité, etc.
  - .: critères souvent mesurable

Notion de conception participative

Besoins : modèle FURPS

Fonctionnalités

- .: fonctions, capacité et sécurité

Utilisabilité

- .: facteurs humains, aide et documentation

Fiabilité (Reliability)

- .: fréquence des pannes, possibilité de récupération et prévisibilité

Performance

- .: temps de réponse, débit, exactitude, disponibilité et utilisation des ressources

Possibilité de prise en charge (Supportability)

- .: adaptabilité, facilité de maintenance, internationalisation et configurabilité

Besoins : modèle FURPS+

FURPS mais aussi :

- .: implémentation : limitation des ressources, langages et outils, matériel, etc.
- .: interface : contraintes d'interfaçage avec des systèmes externes
- .: exploitation : gestion du système dans l'environnement de production
- .: conditionnement
- .: aspects juridiques : attribution de licences, etc.

Activités : analyse / conception

Un seule chose est sûre :

.: l'analyse vient avant la conception

Analyse

.: plus liée à l'investigation du domaine, à la compréhension du problème et des besoins, au quoi

.: recherche du bon système

Conception

.: plus liée à l'implémentation, à la mise en place de solutions, au comment

.: construction du système

Frontière floue entre les deux activités

.: certains auteurs ne les différencient pas

.: et doutent qu'il soit possible de distinguer

.: d'autres placent des limites

.: ex. : analyse hors technologie / conception orientée langage spécifique

Activités : implémentation / tests

Implémentation

.: dans un ou plusieurs langage(s)

.: activité la plus coûteuse

Tests

.: tests unitaires

.: classe, composant

.: test du système intégré

.: non régression

.: ce qui était valide à un moment doit le rester

.: impossible à réaliser sans outils

Plan de ce cours

Avant-propos

Génie logiciel

Méthodes

Activités

Outils

Documentation de projet

Outils et processus

Une méthode spécifie

.: des activités

.: des artefacts à réaliser

Il est souvent vital de disposer d'outil(s) soutenant le processus en

.: pilotant / permettant les activités

.: gérant les artefacts du projet

Les outils peuvent être plus ou moins

.: intégrés à la méthode

.: inter-opérables

.: achetés / fabriqués / transformés…

Des outils pour gérer un projet (1)

Outils de planification
Outils de gestion des versions
Outils de gestion de documentation
Outils de maquettage
Outils de gestion des tests

Des outils pour gérer un projet (2)

Outils de modélisation
Ateliers de développement logiciel
Outils de vérification
Outils de communication
…

Plan de ce cours

Avant-propos
Génie logiciel
Méthodes
Activités
Outils
Documentation de projet
.: Spécification fonctionnelles
.: Document d'Architecture logicielle

Cahier des charges

Spécification d'un système à réaliser
.: Fonctionnalités, description du contexte, contraintes de délais, de prix, préférences…
Document très important
.: Permet de se mettre d'accord en interne sur le système à construire
.: Permet de lancer un appel d'offre
.: Est une partie du contrat entre le demandeur et le prestataire
.: Sert de base et de guide au chef de projet
.: …

Cahier des charges fonctionnel

Préciser les orientations et le champ du domaine étudié,

Analyser l'existant au niveau organisation, documents utilisés, traitements effectués, données manipulées,

Proposer des solutions d'organisation, fonctionnelles et techniques répondant aux exigences et besoins exprimés,

Cahier des charges fonctionnel

Obtenir une description globale du système (organisationnelle, fonctionnelle, technique, contraintes majeures de sécurité, de performance, interfaces avec d'autres systèmes...),

Vérifier la faisabilité organisationnelle et technique,

Aboutir à un choix argumenté d'une solution type de développement.

Spécifications fonctionnelles

Besoins fonctionnels métier

- .: Liste de fonctions que le système devra remplir
- .: Scénarios
- .: Cas d'utilisation
- .: ...

.: Ne pas parler de réalisation technique

Spécifications techniques

Comment réaliser les choses

- .: Liste de fonctions à coder
- .: Architecture
- .: ...

Attention

- .: le mélange technique / métier est facile à faire !
- .: Un exemple
  - .: http://www.pcinpact.com/actu/news/63190-hadopi-specification-fonctionnelles.htm

Architecture ?

Difficile à définir

.: Ex. bâtiment : plombier, électricien, peintre, ne voient pas la même chose.

Définitions

.: Software architecture is not only concerned with structure and behavior, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints and tradeoffs, and esthetics. (RUP, 98)

.: A Technical Architecture is the minimal set of rules governing the arrangement, interaction, and interdependance of the parts or elements that together may be used to form an information system. (U.S. Army 1996)

Architecture

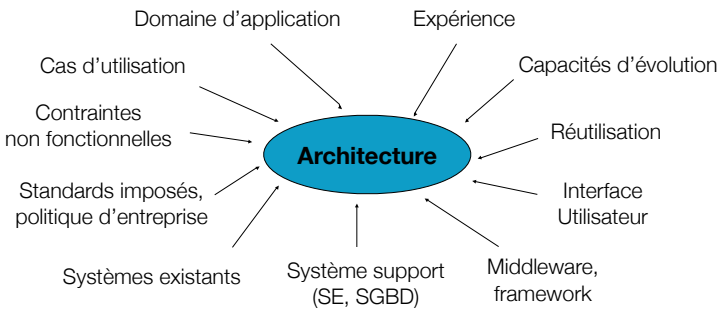
Définition pour ce cours

.: art d'assembler des composants en respectant des contraintes, ensemble des décisions significatives sur

- .: l'organisation du système
- .: les éléments qui structurent le système
- .: la composition des sous-systèmes en systèmes
- .: le style architectural guidant l'organisation (couches…)

.: ensemble des éléments de modélisation les plus signifiants qui constituent les fondations du système à développer

Facteurs influençant l'architecture



Points à considérer

Performances, qualité, testabilité, convivialité, sûreté, disponibilité, extensibilité, exactitude, tolérance aux changements, robustesse, facilité de maintenance, fiabilité, portabilité, risque minimum, rentabilité économique…

Axes pour considérer l'architecture

Architecture logicielle (ou architecture logique) :

.: organisation à grande échelle des classes logicielles en packages, sous-systèmes et couches

- .: architectures client/serveurs en niveaux (tiers)
- .: architectures en couches
- .: architecture à base de composants

Architecture de déploiement

.: décision de déploiement des différents éléments

.: déploiement des fonctions sur les postes de travail des utilisateurs (entreprise : central/départemental/local)

Existence de patterns architecturaux

Description de l'architecture (1)

L'architecture doit être une vision partagée

.: sur un système très complexe

.: pour guider le développement

.: tout en restant compréhensible

Il faut donc mettre en place une description (ou documentation) explicite de l'architecture

.: qui servira de référence jusqu'à la fin du cycle (et après)

.: qui doit rester aussi stable que l'architecture de référence

(Larman, 2004)

Description de l'architecture (2)

Comment décrire l'architecture ?

.: décrire les facteurs qui influencent l'architecture

.: facteurs architecturaux

.: décrire les choix qui ont été faits

.: mémos techniques

.: décrire l'architecture

.: document d'architecture du logiciel

(Larman, 20

Description de l'architecture : Facteurs architecturaux

Facteurs qui ont une influence significative sur l'architecture

.: fonctionnalités, performance, fiabilité, facilité de maintenance, implémentation et interface, etc.

Un facteur architectural doit être identifié et décrit dans une fiche

Nom
ex. : « fiabilité, possibilité de récupération »
Mesures et scénarios de qualité
ce qu'il doit se passer et comment le vérifier
ex. : « si problème, récupération dans la minute »
Variabilité
souplesse actuelle et évolutions futures
ex. : «pour l'instant service simplifiés acceptables en cas de rupture, évolution : services complets »
Impacts
pour les parties prenantes, l'architecture…
ex. : « fort impact, rupture de service non acceptable »
Priorité
ex. : élevée
Difficulté ou risque
ex. : moyen

(Larman, 20

Description de l'architecture Mémos techniques

Les choix architecturaux doivent prendre en compte les facteurs architecturaux

Il est important de décrire les solutions choisies et leur motivation

.: assurer la traçabilité des décisions architecturales

.: raisons des choix

.: alternatives étudiées, etc.

.: Les « mémos techniques » décrivent les choix architecturaux

.: texte + diagrammes

Mémo technique :	Solution
Nom du problème étudié	Motivation
Résumé de la solution	Problèmes non résolus
Facteurs architecturaux	Autres solutions envisagées

(Larman, 20

Description de l'architecture

Vue architecturale

- .: vue de l'architecture du système depuis un point de vue particulier
  - .: texte + diagrammes
- .: se concentre sur les informations essentielles et documente la motivation
  - .: « ce que vous diriez en 1 minute dans un ascenseur à un collègue »
- .: description a posteriori

guidelines.html

DAL : Document d'architecture du logiciel

= récapitulatif des décisions architecturales

- .: Résumé rapide de l'architecture
  - .: 1 diagramme + texte
- .: Facteurs architecturaux
  - .: quels sont les points qui ont eu une influence importante sur les choix d'architecture ?
- .: Ensemble de vues architecturales
  - .: description du matériel et du logiciel utilisés
- .: Mémos techniques
  - .: quelles sont les décisions qui ont été prises, pourquoi, etc.

http://www.codingthearchitecture.com/pages/book/software-architecture-document-guidelines.html

DAL : Contenu possible

- .: Context
- .: Functional View
- .: Process View
- .: Non-functional View
- .: Constraints
- .: Principles
- .: Logical View
- .: Interface View
- .: Design View
- .: Infrastructure View
- .: Deployment View
- .: Operational View
- .: Security View
- .: Data View
- .: Technology Selection
- .: Architecture Justification

On a vu :

Avant-propos
Génie logiciel
Méthodes
Activités
Outils
Documentation de projet

UML
2/5 – Processus Unifié

Aurélien Tabard
Département Informatique
Université Claude Bernard Lyon 1
2014

Basé sur le cours de Yannick Prié

Objectifs du cours global

Notion de méthode de conception
de Système Informatique (SI)

Plan des 4 scéances :

- .: Généralités sur les méthodes
- **.: Processus unifié**
- .: UML : Cas d'utilisation
- .: UML : Diagrammes de séquence
- .: UML : Diagrammes d'état
- .: Processus agiles (si assez de temps)

Plan

Trame du processus unifié

Caractéristiques essentielles

Plan

Trame du processus unifié

Caractéristiques essentielles

Unified Software Development Process

.: Développé dans les années 90 par Rumbaugh, Booch, Jacobson (les concepteurs originaux d'UML)

.: Purement objet

.: Gérer un projet logiciel de bout en bout

.: Les généralités présentées sur USDP s'appliquent à beaucoup de méthodes objets

Unified Software Development Process

Un processus capable de

.: dicter l'organisation des activités de l'équipe

.: diriger les tâches de chaque individu et de l'équipe dans son ensemble

.: spécifier les artefacts à produire

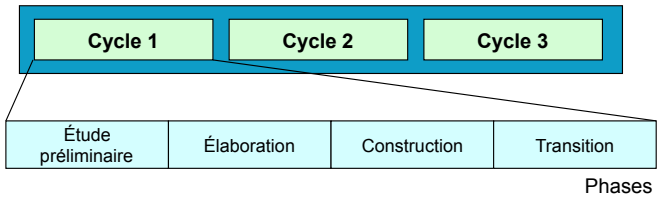
.: proposer des critères pour le contrôle de produits et des activités de l'équipe

Regroupement de bonnes pratiques, mais

.: non figé

.: générique (hautement adaptable : individus, cultures, …)

Les 4 phases du cycle de vie



.: Considérer un produit logiciel par rapport à ses versions

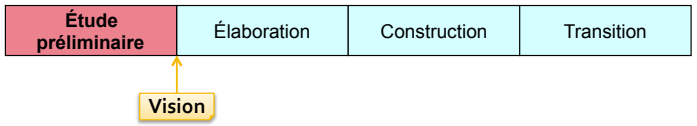
.: un cycle produit une version

.: Gérer chaque cycle de développement comme un projet ayant quatre phases

.: vue gestionnaire (manager)

.: chaque phase se termine par un point de contrôle (ou jalon) permettant aux chefs de projet de prendre des décisions

Phase 1 : Etude préliminaire



.: Déterminer

.: que fait le système ?

.: à quoi pourrait ressembler l'architecture ?

.: quels sont les risques ?

.: quel est le coût estimé du projet ? Comment le planifier ?

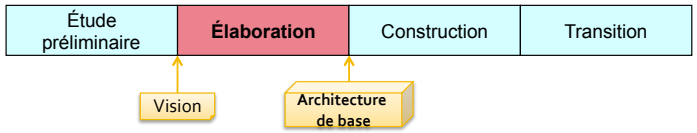
.: Jalon

.: « vision du projet » = document

.: Accepter le projet ?

.: Coût faible

Phase 2 : Elaboration



.: Spécifier la plupart des cas d'utilisation

.: Concevoir l'architecture de base (squelette du système)

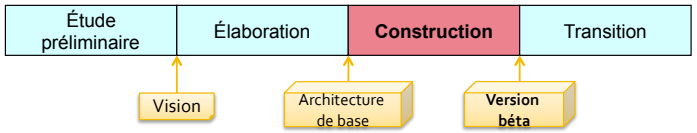
.: Mettre en œuvre cette architecture

.: Faire une planification complète

.: Jalon

- .: « architecture du cycle de vie » = premier prototype qui démontre la validité des choix architecturaux
- .: Peut-on passer à la construction ? (besoins, architecture, planning stables ? Risques contrôlés ?)

Phase 3 : Construction



.: Développer par incréments

- .: l'architecture reste stable malgré des changements mineurs
- .: le produit contient au final tout ce qui avait été planifié, il reste quelques erreurs

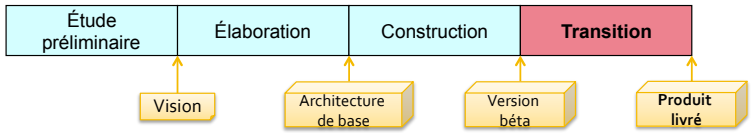
.: Jalon

- .: « capacité opérationnelle initiale » = version béta
- .: Le produit est-il suffisamment correct pour être installé chez le / un client ?

.: Phase la plus coûteuse

- .: > 50% du cycle
- .: englobe conception, codage, tests, intégration, etc.

Phase 4 : Transition



.: Transition

- .: livrer / déployer le produit
- .: corriger le reliquat d'erreurs
- .: améliorer le produit
- .: former les utilisateurs
- .: lettre en place l'assistance en ligne

.: Jalon

- .: « livraison du produit » = produit déployé chez le client
- .: tests suffisants ? Produit satisfaisant ? Manuels prêts ?

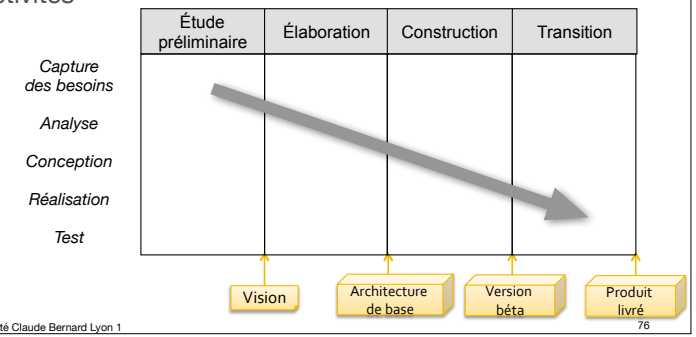
Phases et Activités

Un cycle met en jeu des activités

.: vue du développement sous l'angle technique (développeur)

.: les activités sont réalisées au cours des phases

.: les activités



Plan

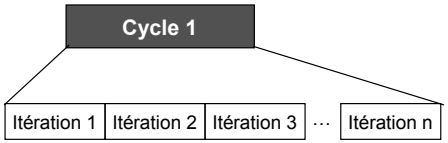
Trame du processus unifié

Processus unifié : caractéristiques essentielles

- **.: itératif et incrémental**
- .: piloté par les besoins
- .: piloté par les risques
- .: centré sur l'architecture

Itérations et incréments

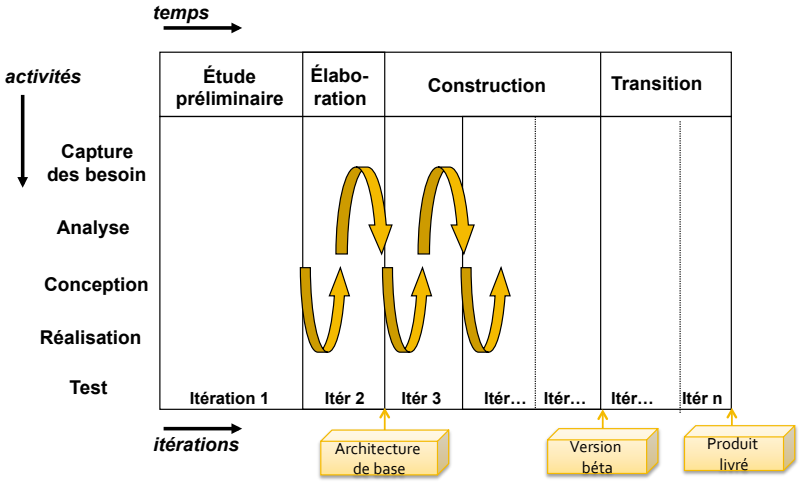
- .: Anti-cascade, spirale
- .: Construction progressive du système
  - .: processus incrémental
- .: Chaque itération permet de maîtriser une partie des risques et apporte une preuve tangible de faisabilité
  - .: produit un système partiel opérationnel (exécutable, testé et intégré) avec une qualité égale à celle d'un produit fini (alpha)
  - .: qui peut être évalué : permet de savoir si on va dans la bonne direction ou non
- .: Série de prototypes qui vont en s'améliorant
  - .: de plus en plus de fonctionnalités disponibles
  - .: retours utilisateurs



Une itération

- .: est un mini-projet :
  - .: plan pré-établi et objectifs pour le prototype, critères d'évaluation,
- .: comporte toutes les activités (mini-cascade)
- .: est terminée par un point de contrôle
- .: ensemble de modèles agréés, décisions pour les itérations suivantes
- .: conduit à une version montrable implémentant un certain nombre de Cas d'Utilisation (CU).
- .: dure entre quelques semaines et 9 mois (au delà : danger)
- .: butée temporelle qui oblige à prendre des décisions

Itérations et phases



Avantages d'un processus itératif & incrémental

Gestion de la complexité

.: pas tout en même temps, étalement des décisions importantes

Maîtrise des risques élevés précoce

.: diminution de l'échec

.: architecture mise à l'épreuve rapidement (prototype réel)

Intégration continue

.: progrès immédiatement visibles

.: maintien de l'intérêt des équipes (court terme, prototypes vs documents)

Avantages d'un processus itératif & incrémental

Prise en compte des modifications de besoins

.: feedback, implication des utilisateurs et adaptation précoce

Apprentissage rapide de la méthode

.: amélioration de la productivité et de la qualité du logiciel

Adaptation de la méthode

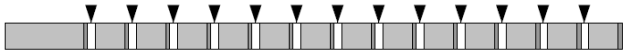
.: possibilité d'explorer méthodiquement les leçons tirées d'une itération (élément à conserver, problèmes, éléments à essayer…)

Mais gestion de projet plus complexe : planification adaptative

Résumé

Démarche "en bloc"

Démarche itérative



Plan

Trame du processus unifié

Processus unifié : caractéristiques essentielles

.: itératif et incrémental

.: piloté par les besoins

.: piloté par les risques

.: centré sur l'architecture

Un processus piloté par les besoins

Objectif du processus

.: construction d'un système qui réponde à des besoins

.: par construction complexe de modèles

Cas d'utilisation = expression / spécification des besoins

.: CU portée système  / objectifs utilisateurs 

CU utilisés tout au long du cycle

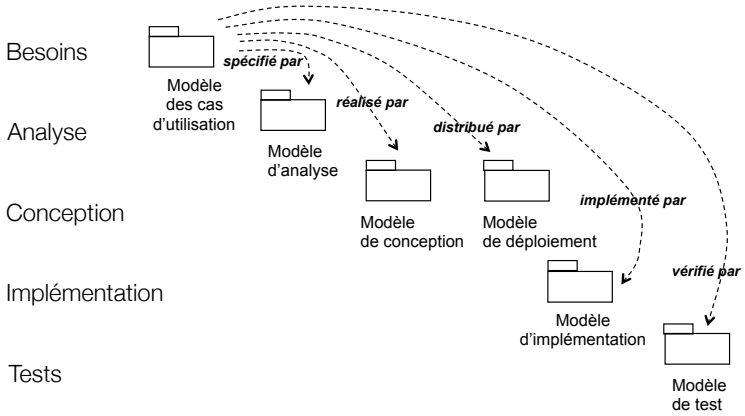
.: validation des besoins / utilisateurs

.: point de départ pour l'analyse (découverte des objets, de leurs relations, de leur comportement) et la conception (sous-systèmes)

.: guide pour la construction des interfaces

.: guide pour la mise au point des plans de tests

Les CU lient les modèles



Avantages des cas d'utilisation

.: Centrés utilisateurs

.: support de communication en langue naturelle entre utilisateurs et concepteurs basé sur les scénarios (et non liste de fonctions)

.: dimension satisfaction d'un objectif utilisateur

.: également pour les utilisateurs informaticiens (administration)

.: besoins fonctionnels, pour acteurs humains et non humains à identifier précisément

.: Assurent la traçabilité par rapports aux besoins de toute décision de conception sur l'ensemble du projet

.: tout modèle peut se référer in fine à un CU

.: Fournissent une vision commune aux participants du projet

.: management, conception, qualité, marketing, etc.

.: Attention :

.: créer de bons CU est un art (voir prochain cours)

Plan

Trame du processus unifié

Processus unifié : caractéristiques essentielles

.: itératif et incrémental

.: piloté par les besoins

.: piloté par les risques

.: centré sur l'architecture

Risque ?

Risque que le projet de construction du système soit un échec

.: Différentes natures de risques pour un projet

.: besoins / technique / autres

Exemples

.: le système construit n'est pas le bon

.: architecture inadaptée, utilisation de technologies mal maîtrisées, performances insuffisantes

.: personnel insuffisant, problèmes commerciaux ou financiers (risques non techniques, mais bien réels)

Gérer les risques

Identifier et classer les risques par importance

Agir pour diminuer les risques

.: ex. changer les besoins, confiner la portée à une petite partie du projet,
faire des test pour vérifier leur présence et les éliminer

S'ils sont inévitables, les évaluer rapidement

.: tout risque fatal pour le projet est à découvrir au plus tôt

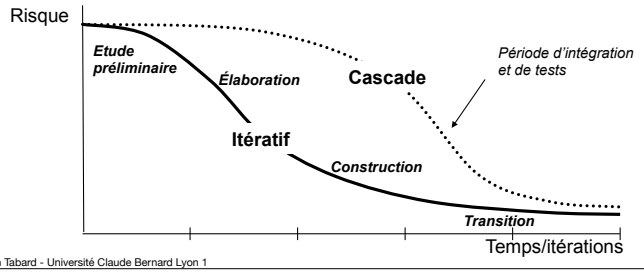
Pilotage par les risques et itérations

.: Identifier, lister et évaluer la dangerosité des risques

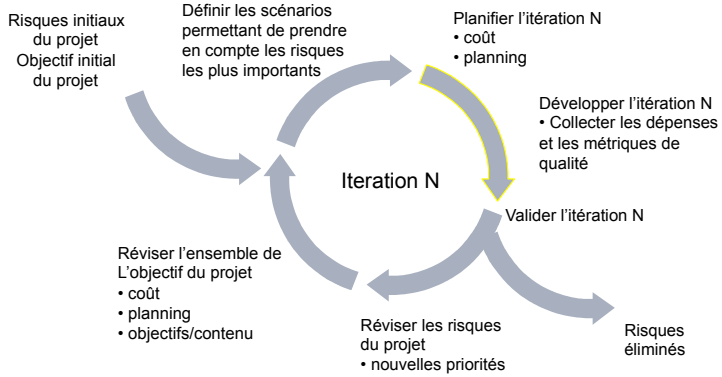
.: Construire les itérations en fonction des risques : s'attaquer en priorité aux risques les plus importants qui menacent le plus la réussite du projet

.: « provoquer des changements précoces »

.: ex. stabiliser l'architecture et les besoins liés le plus tôt possible



Itération pilotées par la réduction des risques



Itérations et risque

On ordonne les itérations à partir des priorités établies pour les cas d'utilisation et de l'étude du risque

- .: plan des itérations
- .: chaque prototype réduit une part du risque et est évalué comme tel
- .: les priorités et l'ordonnancement de construction des prototypes peuvent changer avec le déroulement du plan

Planification des itérations

- .: Planification des itérations dès l'élaboration
  - .: précise pour l'itération suivante, imprécise pour la fin
  - .: la planification générale est adaptée en fonction des risques identifiés/traités et des résultats obtenus dans les itérations
- .: Planification adaptative (vs. prédictive)
  - .: fixer des butées temporelles
  - .: gérer l'adaptation avec le client
- .: Itération
  - .: toutes les activités des besoins aux tests, résultats différents en fonction de la phase dans laquelle on se trouve
  - .: mini-projet : planning, ressources, revue
  - .: premières itérations : suivre une méthode
  - .: à la fin d'une itération : retrospective
    - .: éléments à conserver, problèmes, éléments à essayer

Plan

Trame du processus unifié

Processus unifié : caractéristiques essentielles

- .: itératif et incrémental
- .: piloté par les besoins
- .: piloté par les risques
- **.: centré sur l'architecture**

Processus centré sur l'architecture

- .: L'architecture sert de lien pour l'ensemble des membres du projet
  - .: réalisation concrète de prototypes incrémentaux qui « démontrent » les décisions prises
  - .: vient compléter les cas d'utilisation comme « socle commun »
- .: Contrainte de l'architecture
  - .: plus le projet avance, plus l'architecture est difficile à modifier
  - .: -> les risques liés à l'architecture sont très élevés, car très coûteux
- .: Objectif pour le projet
  - .: établir dès la phase d'élaboration des fondations solides et évolutives pour le système à développer, en favorisant la réutilisation
  - .: l'architecture s'impose à tous, contrôle les développements ultérieurs, permet de comprendre le système et d'en gérer la complexité
- .: L'architecture est contrôlée et réalisée par l'architecte du projet

Objectif / architecture

Construire une architecture

- .: comme forme dans laquelle le système doit s'incarner
  - .: les CU réalisés doivent y trouver leur place
  - .: la réalisation des CU suivant doit s'appuyer sur l'architecture.
- .: qui permette de promouvoir la réutilisation
- .: qui ne change pas trop
  - .: converger vers une bonne architecture rapidement

Analyse architecturale

Identifier et traiter les besoins non fonctionnels dans le contexte des besoins fonctionnels

Identifier les points de variation et d'évolution les plus probables

- .: points de variation : variations dans le système existant ou dans les besoins
- .: points d'évolution : points de variation spéculatifs, actuellement absents des besoins

Construction de l'architecture : principe (1)

Pour commencer

- .: choix d'une architecture de haut-niveau et construction des parties générales de l'application
- .: ébauche à partir
  - .: de solutions existantes
  - .: de la compréhension du domaine
  - .: de parties générales aux applications du domaine (quasi-indépendant des CU)
  - .: des choix de déploiement

Construction de l'architecture : principe (2)

Construction de l'architecture de référence :

- .: confrontation à l'ébauche des cas d'utilisation les plus significatifs (un à un)
- .: construction de parties de l'application réelle (sous-systèmes spécifiques)
- .: stabilisation de l'architecture autour des fonctions essentielles (sous-ensemble des CU).
- .: traitement des besoins non fonctionnel dans le contexte des besoins fonctionnels
- .: identification des points de variation et les points d'évolution les plus probables.

Construction de l'architecture : principe (3)

Réalisation incrémentale des cas d'utilisation

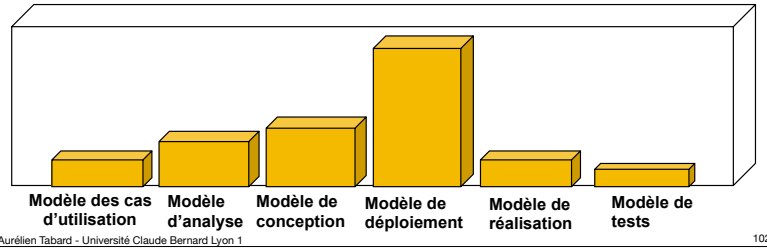
- .: itérations successives
- .: l'architecture continue à se stabiliser sans changement majeur

Remarque : maturation des cas d'utilisation

- .: plus faciles à exprimer et plus précis quand un début d'application est disponible

L'architecture est conçue et réalisée pendant la phase d'élaboration

- .: Phase d'élaboration
  - .: aller directement vers une architecture robuste, à coût limité, appelée « architecture de référence »
  - .: 10% des classes suffisent
- .: L'architecture de référence
  - .: permettra d'intégrer les CU incrémentalement pendant la construction
  - .: guidera le raffinement et l'expression des CU pas encore détaillés



Conclusion

Les 4 grandes caractéristiques du processus unifié sont liées