RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITÉ IBN KHALDOUN-TIARET

FACULTÉ DES MATHÉMATIQUES
ET DE L'INFORMATIQUE

Cours de Génie Logiciel

(avec exercices résolus)

Destiné aux étudiants de la :

1^{ere} Année Master

Spécialité : Génie Logiciel

Réalisé par :

Dr. **Boudjemaa BOUDAA**

Expertisé par :

Pr. **Youcef DAHMANI** - Université Ibn Khaldoun, Tiaret.

Pr. **Sidi Mohammed BENSLIMANE** - ESI, Sidi Bel-Abbès.

Avant-propos

Ce polycopié de cours est le fruit de six années d'expérience dans l'enseignement du module « Génie Logiciel » en faveur les étudiants de la 1ere année Master (spécialité GL) au niveau du département informatique de l'université de Tiaret. Il contient neuf (09) chapitres détaillant les connaissances de base ainsi que les concepts avancés de l'ingénierie des logiciels. D'autres part, il est enrichi par une variété de questions et d'exercices proposés avec solutions de la plupart d'entre eux.

Apres fixer les objectifs visés, chaque chapitre commence par une introduction avant d'aborder les thèmes y afférant. Et il se termine par un rappel des points clés retenus et une série d'exercices avec leurs solutions.

Aussi, et pour un accès rapide et permanent, les chapitres de ce polycopié sont disponibles en ligne via la plateforme Moodle de l'université de Tiaret (http://moodle.univ-tiaret.dz/course/view.php?id=37).

Ce document a été élaboré en se basant sur les célèbres références dans le domaine du génie logiciel, tel que les livres de Sommerville, Pressman, Shari et Jacques Lonchamp, et plusieurs d'autres sources intéressantes et disponibles sur Internet (Springer, par exemple).

Le présent polycopié pourrait servir un grand public de lecteurs :

- En premier lieu, les étudiants en génie logiciel vont y trouver les connaissances théoriques et les bonnes pratiques nécessaires pour apprendre la discipline de l'ingénierie des logiciels.
- Egalement, les enseignants peuvent en bénéficier pour préparer soit des cours ou des fiches de travaux dirigés (TD) à travers les exercices et les solutions proposés.
- Enfin, ce polycopié s'adresse à toute personne intéressée par la conception et le développement des logiciels de manière professionnelle.

Bonne lecture.

Tiaret, le 02/02/2020

Sommaire

Chapitre I

Introduction au Génie Logiciel

Objectifs

- Comprendre le génie logiciel et pourquoi il est important;
- Comprendre que le développement de différents types de systèmes de logiciels peut nécessiter des techniques de génie logiciel;
- Comprendre certains enjeux éthiques et professionnels qui sont importants pour les ingénieurs des logiciels.

Themes couverts

- Histoire du Génie Logiciel;
- Développement des logiciels professionnels;
- Éthiques de l'Ingénierie du Logiciel;
- Etudes de cas.

Chapitre 1:

Introduction au Génie Logiciel

1. Introduction

Pourquoi le Génie logiciel?

- ✧ Les économies de tous les pays développés dépendent sur des logiciels.
- ✧ De plus en plus, les systèmes sont pilotés par des logiciels
- ✧ Le génie logiciel est intéressé par les **théories**, les **méthodes** et les **outils de développement** des logiciels professionnels.
- ✧ Les dépenses sur les logiciels représentent une fraction significative du PNB (Le produit national brut) de tous les pays développés.

Les coûts des logiciels

- ✧ Les coûts des logiciels **dominent** souvent les coûts d'un système informatique. Les coûts des logiciels sur un ordinateur sont souvent plus élevés que le coût du matériel.
- ✧ Logiciel coûte plus cher à maintenir que d'en développer. Pour les systèmes avec une longue durée de vie, les coûts de maintenance peuvent être de plusieurs fois des coûts de développement.
- ✧ Le génie logiciel est préoccupé par le développement de logiciels rentables.

Échec du projet logiciel

- ✧ Accroissement de la complexité du système:
  - Au fur et à mesure que de nouvelles techniques d'ingénierie logicielle nous permettent de construire des systèmes plus grands et plus complexes. Les systèmes doivent être construits et livrés plus rapidement; doivent avoir de nouvelles capacités qui auparavant étaient considérées comme impossibles.

- ✧ Défaut d'utiliser les méthodes d'ingénierie logicielle:
  - Il est assez facile d'écrire des programmes informatiques sans utiliser de méthodes et de techniques d'ingénierie logicielle. Beaucoup d'entreprises ont dérivé du développement de logiciels à mesure que leurs produits et services ont évolué. Ils n'utilisent pas les méthodes d'ingénierie logicielle dans leur travail quotidien. Par conséquent, leur logiciel est souvent plus coûteux et moins fiable qu'il ne le devrait être.

2. Histoire du Génie Logiciel

Naissance du Génie Logiciel

- La notion de «Génie Logiciel» a été proposé en 1968 lors de la conférence «Garmisch-Partenkirchen, Germany, 7th to 11th October 1968 » pour discuter de ce qui était alors appelé la «Crise du Logiciel».

- Il est devenu clair que les approches individuelles au développement du programme n'ont pas pu développer les grands et complexes systèmes logiciels et qui restent:
  - non fiables et ne satisfont pas leurs cahiers des charges
  - coûtent plus chers que prévu,
  - et ont été livrés en retard.

- Tout au long des années 1970 et 1980, une variété de nouvelles techniques et méthodes de génie logiciel ont été développés. Outils et notations standards ont été élaborés et sont maintenant largement utilisés.

- L'initiative viendra de la division des affaires scientifiques de l'OTAN (NATO)[1], qui organise en octobre 1968 sur la suggestion de **F. L. Bauer**, professeur à l'université technique de Munich, une conférence de travail sur les difficultés de la production de logiciel et les moyens de les surmonter.

- Intitulée Working Conference on Software Engineering, elle est considérée comme l'événement fondateur de cette nouvelle discipline et c'est elle qui popularise le terme de software engineering, traduit en français par « génie logiciel ».

- Bauer donne la définition suivante du terme GL : "***Establishment and use of sound engineering principles to obtain economically software that is reliable and works on real machines efficiently***"

Objectifs du Génie Logiciel

- Le génie logiciel (software engineering) est une science de génie industriel qui étudie les méthodes de travail et les bonnes pratiques des ingénieurs qui développent des logiciels. Le génie logiciel s'intéresse en particulier aux procédures systématiques qui permettent d'arriver à ce que des logiciels de grande taille correspondent aux:
  - attentes du client,
  - soient fiables,
  - aient un coût d'entretien réduit et
  - de qualité et de bonnes performances tout en respectant les délais et les coûts de construction.

☞ Le génie logiciel est préoccupé par le développement des logiciels professionnels rentables.

[1] L'*Organisation du traité de l'Atlantique Nord* (*OTAN* ou *Otan* ; en anglais : North Atlantic Treaty Organization)

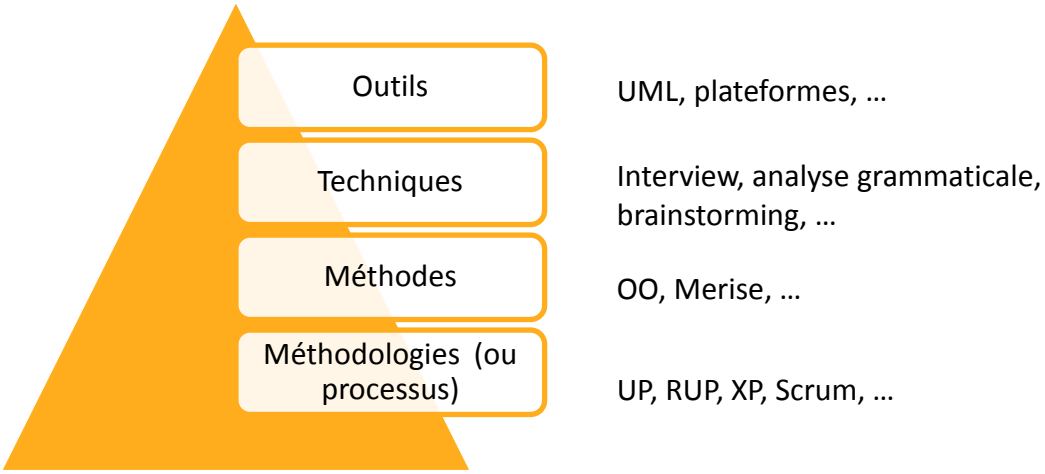
3. Développement de logiciel professionnel

Qu'est-ce que le génie logiciel ?

- **Définition 1**: « Le génie logiciel est une discipline d'ingénierie qui s'occupe de tous les aspects de la production de logiciels ». Le génie logiciel est intéressé par les théories, les méthodes et les outils de développement de logiciels professionnels.

- **Définition 2**: selon l'arrêté du 30 décembre 1983: « ensemble des activités de conception et de mise en œuvre des produits et des procédures tendant à rationaliser la production du logiciel et de son suivi »

- **Définition 3**: « Le génie logiciel est un domaine des sciences de l'ingénieur dont l'objet d'étude est la conception, la fabrication, et la maintenance des systèmes informatiques complexes».



Qu'est-ce qu'un système ?

- **Définition 1**: ensemble d'éléments en interaction dynamique, dont les éléments sont organisés et coordonnés en vue d'atteindre un objectif, qui évolue dans un environnement.

- **Definition 2**: Un système est un ensemble d'éléments interagissant entre eux suivant un certains nombres de principes et de règles dans le but de réaliser un objectif.

- L'environnement est la partie du monde extérieure au système. Un système est souvent hiérarchisé à l'aide de sous-systèmes.

- Un système complexe se caractérise par :
  - sa dimension, qui nécessite la collaboration de plusieurs personnes;
  - son évolutivité.

- **Exemples** : Une fourmilière, l'économie mondiale, le noyau Linux, . . .

  ☞ **De plus en plus, les systèmes sont pilotées par des logiciels**

Qu'est-ce qu'un logiciel ?

- ✧ **Définition 1**: Les programmes informatiques et la documentation associée. Les produits logiciels peuvent être développés pour un client particulier ou peuvent être développés pour un marché général.

- ✧ **Définition 2**: « Un logiciel est un ensemble d'entités nécessaires au fonctionnement d'un processus de traitement automatique de l'information».
  - Parmi ces entités, on trouve par exemple :
    - des programmes (en format *code source* ou exécutables);
    - des documentations d'utilisation;
    - des informations de configuration.

- ✧ **Définition 3**: selon l'arrêté du 22 décembre 1981: ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitements de l'information.

FAQ sur le génie logiciel

Question	Réponse
Qu'est ce qu'un logiciel?	Les programmes informatiques et la documentation associée. Les produits logiciels peuvent être développés pour un client particulier ou peuvent être développés pour un marché général.
Quelles sont les caractéristiques d'un bon logiciel?	Un bon logiciel doit offrir la fonctionnalité et les performances requises pour l'utilisateur et doit être maintenable, fiable et utilisable.
Qu'est-ce que le génie logiciel?	Le génie logiciel est une discipline d'ingénierie qui s'occupe de tous les aspects de la production de logiciels.
Quelles sont les activités fondamentales de génie logiciel?	Spécification des logiciels, développement des logiciels, validation des logiciel et l'évolution des logiciels.
Quelle est la différence entre le génie logiciel et de l'informatique?	L'informatique focalise sur la théorie et les principes fondamentaux; génie logiciel est concerné par les pratiques de développement et de la réalisation des logiciels utiles.
Quelle est la différence entre le génie logiciel et de l'ingénierie de système?	Ingénierie des systèmes s'intéresse à tous les aspects du développement des systèmes à base d'ordinateur, y compris le matériel, les logiciels et l'ingénierie des processus. Génie logiciel fait partie de ce processus plus général.

Les produits logiciels

- ✧ **Les produits génériques**:
  - Systèmes autonomes qui sont commercialisés et vendus à un client qui souhaite les acheter.
  - Exemples : logiciel de PC tels que les programmes graphiques, les outils de gestion de projet; les logiciels de CAO; logiciels pour des marchés spécifiques tels que les systèmes de rendez-vous pour les dentistes.

- ✧ **Les produits commandés (ou sur mesure, personnalisés):**
  - Le logiciel qui est commandé par un client spécifique pour répondre à leurs propres besoins.
  - Exemples: systèmes embarqués de contrôle, logiciel de contrôle du trafic aérien, les systèmes de surveillance (monitoring) du trafic.

Spécification du produit

- ✧ **Les produits génériques:**
  - La spécification de ce que le logiciel doit faire est détenue par le développeur du logiciel et les décisions de modification (changement) sur les logiciels sont faites par le développeur.

- ✧ **Les produits sur mesure:**
  - La spécification de ce que le logiciel doit faire est détenue par le client du logiciel et qui prend des décisions de changement sur les logiciels qui sont nécessaires.

Les caractéristiques essentielles pour un bon logiciel (Comment concevoir un logiciel de qualité ?)

Caractéristique	Description
Maintenabilité (maintainability)	Le logiciel doit pouvoir évoluer pour s'adapter aux besoins changeants des clients. Il s'agit d'un attribut essentiel parce que le changement de logiciel est une exigence inévitable dans un environnement commercial en évolution.
Fiabilité et sécurité (Dependability and security)	La Fiabilité du Logiciel comprend un éventail de caractéristiques, la disponibilité (availability), la sécurité (security) et la sûreté (safety). Un logiciel fiable ne devrait pas causer des dommages physiques ou économiques en cas de défaillance du système. Les utilisateurs malveillants ne devraient pas être en mesure d'accéder ou endommager le système.
Efficacité ou performance (Efficiency or performance)	Les logiciels ne doivent pas gaspiller les ressources système telles que la mémoire et les cycles du processeur. L'efficacité inclut donc la réactivité, le temps de traitement, l'utilisation de la mémoire, etc.
Acceptabilité ou utilisabilité (acceptability or usability)	Le logiciel doit être acceptable par les utilisateurs pour lesquels il est conçu. Cela signifie qu'il doit être documenté, compréhensible, utilisable et compatible avec d'autres systèmes qu'ils utilisent.

Génie logiciel

✧ Le génie logiciel est une discipline d'ingénierie qui s'occupe de tous les aspects de la production de logiciels dès les premières étapes de spécification du système jusqu'à la maintenance du système après qu'il a été mis en usage.

✧ Discipline d'ingénierie :
  - Utilisation des théories et des méthodes appropriées pour résoudre les problèmes en tenant compte de l'organisation et les contraintes financières.

✧ Tous les aspects de la production de logiciels

- N'est pas seulement le processus technique de développement. Aussi, la gestion de projet et le développement d'outils, de méthodes, etc… pour soutenir la production de logiciels.

Activités du processus logiciel

✧ **Spécification du logiciel**, où les clients et les ingénieurs définissent le logiciel qui doit être produit et les contraintes sur son fonctionnement.

✧ **Développement de logiciel**, où le logiciel est conçu et programmé.

✧ **Validation du logiciel**, où le logiciel est vérifié pour s'assurer que c'est ce que le client demande.

✧ **L'évolution du logiciel**, où le logiciel est modifié pour tenir compte de l'évolution des besoins des clients et du marché.

Issues générales affectant la plupart des logiciels

✧ **Hétérogénéité**
- De plus en plus, les systèmes doivent fonctionner comme des systèmes distribués dans des réseaux qui regroupent différents types d'ordinateurs et d'appareils mobiles.

✧ **Economie et changement social**
- L'économie et la société changent incroyablement vite avec l'émergence de nouvelles technologies. Ils doivent être capables de changer leurs logiciels existants et de développer rapidement de nouveaux logiciels.

✧ **Sécurité et confiance**
- Comme le logiciel est intimement lié à tous les aspects de nos vies, il est essentiel que nous puissions faire confiance à ce logiciel.

✧ **Echelle**
- Le logiciel doit être développé dans une très large gamme d'échelles, à partir de très petits systèmes embarqués dans des appareils mettables (wearable) ou portables, jusqu'à des systèmes basés sur l'Internet, basés sur le cloud, qui desservent une communauté mondiale.

Diversité de génie logiciel

✧ Il y a beaucoup de différents types de systèmes de logiciels et il n'y a pas de techniques logicielles universelles qui sont applicables à tous ces systèmes de logiciels.

✧ Les méthodes de génie logiciel et les outils utilisés dépendent du type d'application en cours d'élaboration, des exigences de la clientèle et l'esprit de l'équipe de développement.

Types d'applications

1) Applications autonomes (*Stand-alone applications*)
- Ce sont des systèmes d'application s'exécutant sur un ordinateur local, tel qu'un PC. Elles comprennent toutes les fonctionnalités nécessaires et n'ont pas besoin d'être connecté à un réseau. Ex. applications bureautiques.

2) Applications basés sur les transactions interactives (*Interactive transaction-based applications)*

- Les applications qui s'exécutent sur un ordinateur distant et qui sont accessibles par les utilisateurs à partir de leurs propres ordinateurs ou terminaux. Il s'agit notamment des applications Web telles que les applications de e-commerce. Ex. applications d'entreprise (business systems), services en nuage (cloud-based services)

3) Systèmes embarqués de contrôle

- Ce sont des systèmes logiciels de contrôle qui contrôlent et gèrent les périphériques matériels. Numériquement, il y a probablement plus de systèmes embarqués que n'importe quel autre type de système. Ex. logiciel dans un téléphone portable, logiciel contrôlant le freinage dans une voiture (anti-lock braking) et logiciel dans un four à micro-ondes permettant de contrôler le processus de cuisson.

4) Les systèmes de traitement par lots

- Ce sont des systèmes de gestion qui sont conçus pour traiter des données en grandes séries. Ils traitent un grand nombre de différentes entrées pour créer des sorties correspondantes. Ex. systèmes de facturation périodique.

5) Systèmes de divertissement

- Ce sont des systèmes qui sont principalement pour un usage personnel et qui sont destinées à divertir l'utilisateur. Ex. Games.

6) Systèmes de modélisation et de simulation

- Ce sont des systèmes qui sont développés par des scientifiques et des ingénieurs pour modéliser des processus ou des situations physiques, qui comprennent de nombreux objets séparés et en interaction.

7) Systèmes de collecte de données

- Ce sont des systèmes de collecte de données à partir de leur environnement, en utilisant un ensemble de capteurs et d'envoyer ces données à d'autres systèmes de traitement.

8) Systèmes de systèmes

- Ce sont des systèmes qui sont composés d'un certain nombre d'autres systèmes logiciels.

Remarque: les limites entre ces types de systèmes sont floues. Ex. Les systèmes de traitement par lots sont souvent utilisés avec des systèmes Web. Par exemple, dans une entreprise, les demandes de remboursement de frais de déplacement peuvent être soumises via une application Web, mais traitées dans une application de traitement par lots pour un paiement mensuel.

Principes Fondamentaux de Génie Logiciel

✧ Certains principes fondamentaux s'appliquent à tous les types de système de logiciels, quelles que soient les techniques de développement utilisées:

- Les systèmes doivent être développés **en utilisant un processus de développement** réussi et compréhensible. L'organisation qui développe le logiciel doit planifier le processus de développement et avoir une idée claire de ce qui sera produit et du moment où il sera terminé. Bien entendu, différents processus sont utilisés pour différents types de logiciels.
- La **fiabilité et la performance** sont importantes pour tous les types de systèmes. Les

logiciels doivent se comporter comme prévu, sans défaillance et doivent être disponibles pour une utilisation en cas de besoin. Son fonctionnement doit être sûr (savety) et, dans la mesure du possible, protégé contre les attaques externes (security). Le système doit fonctionner efficacement et ne doit pas gaspiller de ressources.

- Comprendre et **gérer les spécifications logicielles et les exigences** (ce que le logiciel doit faire) sont important. Vous devez savoir ce que les différents clients et utilisateurs du système attendent de celui-ci et gérer leurs attentes afin qu'un système utile puisse être livré dans les limites du budget et du calendrier.
- Vous devez **utiliser le plus efficacement possible les ressources existantes**. Cela signifie que, le cas échéant, vous devez réutiliser un logiciel déjà développé plutôt que d'écrire un nouveau logiciel.

Génie logiciel basé sur le Web

✧ Systèmes basés sur le Web sont des systèmes distribués complexes mais les principes fondamentaux du génie logiciel discutés précédemment s'appliquent aussi à eux car ils sont parmi les types de systèmes.

✧ Les idées fondamentales de l'ingénierie logicielle discutés s'appliquent au logiciel basé sur le Web de la même manière qu'ils s'appliquent à d'autres types de systèmes logiciels.

✧ Réutilisation des logiciels

- Réutilisation de logiciels est l'approche dominante pour la construction de systèmes basés sur le Web. Lors de la construction de ces systèmes, vous pensez à comment vous pouvez les assembler à partir de composants et de systèmes logiciels préexistants.

✧ Développement incrémental et agile

- Systèmes basés sur le Web devraient être élaborés et exécutés progressivement (par l'incrémentation). Maintenant, Il est admis généralement qu'il est impossible de spécifier toutes les exigences de ces systèmes à l'avance.

✧ Systèmes orientés services (Service-oriented systems)

- Le logiciel peut être implémenté à l'aide d'une ingénierie logicielle axée sur les services, où les composants logiciels sont des services Web autonomes.

✧ Les interfaces riches

- Les technologies telles que AJAX et HTML5 permettent la création des interfaces riches au sein d'un navigateur Web, mais sont encore difficiles à utiliser. Les formulaires Web avec les scripts locaux sont plus communément utilisés.

4. Éthiques de l'ingénierie du logiciel

✧ Génie logiciel implique des responsabilités plus larges que la simple application des compétences techniques.
✧ Les ingénieurs logiciels doivent se comporter de façon responsable, honnête et éthique s'ils veulent être respectés en tant que professionnels.
✧ Le comportement éthique est plus que simplement faire respecter la loi, mais consiste à la suite d'une série de principes qui sont moralement corrects.

Les issues de responsabilité professionnelle

✧ **Confidentialité**
- Les ingénieurs devraient normalement respecter la confidentialité de leurs employeurs ou clients, indépendamment de si oui ou non un accord formel de confidentialité a été signé.

✧ **Compétence**
- Les ingénieurs ne devraient pas dénaturer leur niveau de compétence. Ils ne doivent pas accepter le travail qui est en dehors de leur compétence.

✧ **Droits de propriété intellectuelle**
- Les ingénieurs devraient être au courant des lois locales régissant l'utilisation de la propriété intellectuelle telle que les brevets, droits d'auteur, etc. Ils devraient faire attention à ce que la propriété intellectuelle des employeurs et des clients est protégé.

✧ **Mauvaise utilisation de l'ordinateur**
- Les ingénieurs logiciels ne doivent pas utiliser leurs compétences techniques pour abuser les ordinateurs d'autres personnes. Mauvaise utilisation de l'ordinateur varie de relativement trivial (jeu en jouant sur la machine de l'employeur, par exemple) au extrêmement graves (diffusion de virus).

Code d'éthique ACM/IEEE

✧ Les sociétés professionnelles publient des codes de conduite définissant les normes de comportement attendues de leurs membres.

✧ Exemple de code: ACM/IEEE code d'éthique (voir le livre de sommerville 2015).

5. Étude de Cas

✧ **Une pompe à insuline personnel**
- Un système embarqué dans une pompe à insuline utilisée par les diabétiques pour maintenir le contrôle glycémique.

✧ **Un système de gestion des patients des cas de santé mentale**
- MentCare: Un système utilisé pour tenir les enregistrements des personnes recevant des soins pour des problèmes de santé mentale.

✧ **Une station météorologique de désert (zones de nature sauvage)**
- Un système de collecte de données qui recueille des données sur les conditions météorologiques dans les régions éloignées.

✧ **iLearn: un environnement d'apprentissage numérique**
- Un système de soutien à l'apprentissage dans les écoles

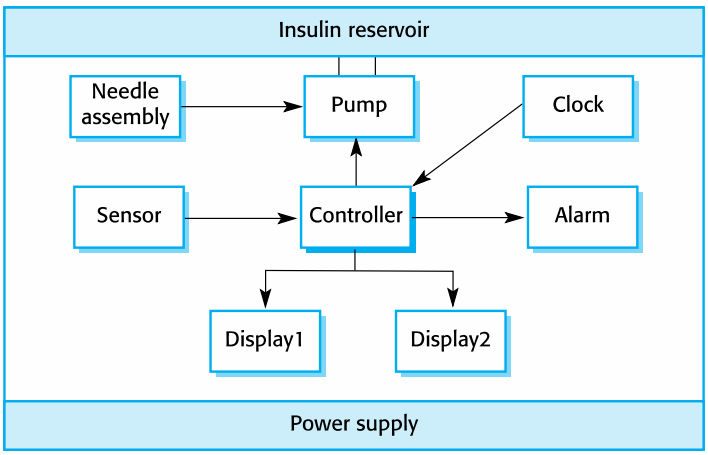
1) Système de contrôle de la pompe à insuline

✧ Collecte des données provenant d'un capteur de glucose sanguin et calcule la quantité d'insuline nécessaire pour être injecté.

✧ Calcul basé sur la vitesse de variation du taux de sucre dans le sang.

✧ Envoie des signaux à une micro-pompe pour délivrer la dose correcte d'insuline.

✧ Un système de sécurité critique parce que les faibles taux de glycémie peuvent entraîner un mauvais fonctionnement du cerveau, le coma et la mort; et des taux élevés de sucre dans le sang ont des conséquences à long terme, tels que des lésions oculaires et rénales.



Exigences essentielles de haut niveau

✧ Le système doit être disponible pour fournir de l'insuline si nécessaire.

✧ Le système doit fonctionner de manière fiable et de fournir la bonne quantité d'insuline pour contrer le niveau actuel des taux de sucre sanguin.

✧ Le système doit donc être conçu et mis en œuvre pour s'assurer que le système répond toujours à ces exigences.

✧ **Mentcare: Un système d'information sur les patients pour soins de santé mentale**
Un système d'information patient pour soutenir les soins de santé mentale est un système d'information médicale qui gère les informations sur les patients souffrant de problèmes de santé mentale et les traitements qu'ils ont reçus.

✧ La plupart des patients de santé mentale ne nécessitent pas de traitement de l'hôpital dédié, mais ont besoin d'assister à des cliniques spécialisées régulièrement où ils peuvent rencontrer un médecin qui a une connaissance approfondie de leurs problèmes.

✧ Pour aider les patients à y assister, ces cliniques n'existent pas seulement dans les hôpitaux. Ils peuvent également être organisés dans les cabinets médicaux locales ou dans les centres communautaires.

✧ Le Mentcare est un système d'information qui est destiné à être utilisé dans les cliniques.

✧ Il fait usage d'une base de données centralisée de l'information du patient, mais a également été conçu pour fonctionner sur un PC, de sorte qu'il peut être consulté et utilisé à partir des sites qui n'ont pas de connectivité réseau sécurisée.

✧ Lorsque les systèmes locaux disposent d'un accès réseau sécurisé, ils utilisent l'information du patient dans la base de données, mais ils peuvent télécharger et utiliser des copies locales des dossiers des patients lorsqu'ils sont déconnectés.

Buts de Mentcare:

✧ Pour générer des informations de gestion qui permettent aux gestionnaires de services de santé d'évaluer le rendement par rapport aux objectifs locaux et gouvernementaux.

✧ De fournir au personnel médical de l'information opportune pour soutenir le traitement des patients.

Principales caractéristiques du Mentcare

✧ La gestion de la prise en charge individuelle
- Les cliniciens peuvent créer des dossiers pour les patients, modifier les informations dans le système, voir l'historique du patient, etc. Le système prend en charge des résumés de données afin que les médecins peuvent apprendre rapidement les problèmes et les principaux traitements qui ont été prescrits.

✧ La surveillance du patient
- Le système surveille les dossiers des patients qui sont impliqués dans le traitement et émet des avertissements si les éventuels problèmes sont détectés.

✧ Rapports administratifs
- Le système génère des rapports de gestion mensuels indiquant le nombre de patients traités dans chaque clinique, le nombre de patients qui sont entrés et sortis du système de soins, le nombre de patients sélectionnés, les médicaments prescrits et leurs coûts, etc.

Préoccupations Mentcare

✧ Vie privée
- Il est essentiel que l'information du patient est confidentielle et n'est jamais divulguée à personne en dehors de personnel médical autorisé et le patient lui-même.

✧ Sécurité
- Certaines maladies mentales provoquent les patients à devenir suicidaire ou un danger pour d'autres personnes. Chaque fois que possible, le système doit avertir le personnel médical sur les patients potentiellement suicidaires ou dangereux.
- Le système doit être disponible en cas de besoin par ailleurs, la sécurité peut être compromise et il peut être impossible de prescrire le bon médicament aux patients.

2) Station météorologique sauvage

✧ Le gouvernement d'un pays avec de grandes zones de nature sauvage décide de déployer plusieurs centaines de stations météorologiques dans les régions éloignées.

✧ Les stations météorologiques collectent des données à partir d'un ensemble d'instruments qui mesurent la température et la pression, le soleil, la pluie, la vitesse du vent et direction du vent.
- La station météorologique comprend un certain nombre d'instruments qui mesurent les paramètres météorologiques tels que la vitesse et direction du vent, les températures du sol et de l'air, la pression barométrique et les précipitations sur une période de 24 heures. Chacun

de ces instruments est contrôlé par un système de logiciel qui effectue les lectures de paramètres périodiquement et gère les données collectées à partir des instruments.

Système d'information météorologique

✧ Le système de la station météo
- Ceci est responsable de la collecte de données météorologiques, la réalisation d'un certain traitement de données initial et de la transmettre au système de gestion de données.

✧ La gestion de données et le système d'archivage
- Ce système recueille les données de toutes les stations météorologiques de nature sauvage, effectue le traitement des données et l'analyse et archive les données.

✧ Le système de maintenance de la station
- Ce système peut communiquer par satellite avec toutes les stations météorologiques de désert pour surveiller la santé de ces systèmes et de fournir des rapports de problèmes.

Fonctionnalité supplémentaire du logiciel

✧ Surveiller les instruments, le matériel électrique et la communication et signaler les défauts du système de gestion.

✧ Gérer l'alimentation du système, veiller à ce que les batteries sont chargées à chaque fois que les conditions environnementales le permettent, mais aussi si les générateurs sont arrêtés à cause des conditions météorologiques potentiellement néfastes, tels que des vents forts.

✧ Soutenir la reconfiguration dynamique où des parties du logiciel sont remplacés par de nouvelles versions et où les instruments de sauvegarde sont commutés dans le système en cas de défaillance du système.

3) iLearn: Un environnement d'apprentissage numérique

✧ Un environnement d'apprentissage numérique est un cadre de travail (framework) dans lequel un ensemble d'outils d'apprentissage à usage général et spécialement conçus pour l'apprentissage peut être embarqués avec un ensemble d'applications adaptées aux besoins des apprenants utilisant le système.

✧ Les outils inclus dans chaque version de l'environnement sont choisis par les enseignants et les apprenants en fonction de leurs besoins spécifiques.
- Ceux-ci peuvent être des applications générales telles que des feuilles de calcul, des applications de gestion de l'apprentissage comme un environnement d'apprentissage virtuel (VLE) pour gérer la soumission et l'évaluation des devoirs, les jeux et les simulations.

Systèmes orientés services

✧ Le système est un système axé sur les services avec tous les composants du système considérés comme un service remplaçable.

✧ Cela permet au système d'être mis à jour progressivement à mesure que de nouveaux services sont disponibles.

✧ Il permet également de configurer rapidement le système pour créer des versions de l'environnement pour différents groupes tels que les très jeunes enfants qui ne peuvent pas lire, les étudiants seniors, etc.

Services d'iLearn

✧ Services d'utilité qui fournissent des fonctionnalités basiques indépendantes de l'application et qui peuvent être utilisées par d'autres services du système.

✧ Services d'application qui fournissent des applications spécifiques telles que les courriels, les conférences, le partage de photos, etc., et l'accès à des contenus éducatifs spécifiques tels que des films scientifiques ou des ressources historiques.

✧ Les services de configuration utilisés pour adapter l'environnement à un ensemble spécifique de services d'application et définir comment les services sont partagés entre les étudiants, les enseignants et leurs parents.

Service iLearn d'integration

✧ Les services intégrés sont des services proposant une API (interface de programmation d'applications) et auxquels les autres services peuvent accéder par l'intermédiaire de cette API. La communication directe entre les services est donc possible.

✧ Les services indépendants sont des services auxquels on accède facilement via une interface de navigateur et qui fonctionnent indépendamment d'autres services. L'information ne peut être partagée qu'avec d'autres services grâce à des actions explicites de l'utilisateur telles que copier et coller; une ré-authentification peut être requise pour chaque service indépendant.

6. Points clés

- ✓ Les ingénieurs en logiciel ont des responsabilités à la profession d'ingénieur et de la société. Ils ne doivent pas simplement être concernés par les questions techniques.
- ✓ Les associations professionnelles publient des codes de conduite qui énoncent les normes de comportement attendues de leurs membres.
- ✓ Trois études de cas sont présentées pour les utilisés dans les futurs chapitres.
- ✓ Le génie logiciel est une discipline d'ingénierie qui s'occupe de tous les aspects de la production de logiciels.
- ✓ Les caractéristiques essentielles des logiciels sont la maintenabilité, la fiabilité et la sécurité, l'efficacité et l'acceptabilité.
- ✓ Les activités de haut niveau de spécification, de développement, de validation et d'évolution font partie de tous les processus de logiciels.
- ✓ Les notions fondamentales du génie logiciel sont universellement applicables à tous les types de développement des systèmes.
- ✓ Il y a beaucoup de différents types de systèmes et chacun requiert des outils et des techniques de génie logiciel appropriés pour leur développement.
- ✓ Les idées fondamentales du génie logiciel sont applicables à tous les types de système de logiciel.
- ✓ Les ingénieurs logiciels ont des responsabilités à l'égard de la profession d'ingénieur et de la société. Ils ne devraient pas seulement se préoccuper des problèmes techniques.
- ✓ Les sociétés professionnelles publient des codes de conduite définissant les normes de comportement attendues de leurs membres.

7. Exercices

I) Quiz :

1: Quelle question ne concerne plus l'ingénieur logiciel moderne?

a. Pourquoi le matériel informatique coûte très cher?
b. Pourquoi le logiciel prend-il beaucoup de temps pour terminer?
c. Pourquoi coûte tellement le développement d'un logiciel?
d. Pourquoi les erreurs de logiciel ne peuvent pas être retirées des produits avant la livraison?

2: Le logiciel est un produit qui peut être fabriqué en utilisant les mêmes technologies utilisées pour d'autres artefacts d'ingénierie.

a. Vrai
b. Faux

3: Le logiciel se détériore plutôt que s'use parce que

a. Le logiciel souffre d'une exposition à des environnements hostiles
b. Les défauts sont plus susceptibles de se produire après que le logiciel a été souvent utilisé.
c. Les demandes multiples de changement introduisent des erreurs dans les interactions des composants.
d. Les parties de rechange des logiciels deviennent plus difficiles à commander.

4: Les WebApps sont un mélange de publication imprimée et de développement de logiciels, rendant leur développement hors du domaine de la pratique de l'ingénierie logicielle.

a. Vrai
b. Faux

5: Il n'y a pas de différences réelles entre le développement des WebApps et MobileApps.

a. Vrai
b. Faux

6: Dans sa forme la plus simple, un dispositif (device) informatique externe peut accéder aux services de données en nuage (cloud computing) à l'aide d'un navigateur Web.

a. Vrai
b. Faux

7: Le développement du logiciel de ligne de produits (Product Line Software) dépend de la réutilisation des composants logiciels existants dans leur ingénierie logicielle.

a. Vrai
b. Faux

8: La réutilisation des logiciels réduit le coût et augmente la valeur des systèmes dans lesquels ils sont incorporés.

a. Vrai
b. Faux

9: L'essence de la pratique de l'ingénierie logicielle pourrait être décrite comme comprendre le problème, planifier une solution, exécuter le plan et examiner le résultat pour plus de précision.

a. Vrai
b. Faux

10: En général, le logiciel ne réussit que si son comportement est conforme aux objectifs de ses concepteurs.

a. Vrai
b. Faux

II) Questions de recherche :

1. Donner les différentes sortes de documents qui accompagnent un logiciel professionnel.
2. Quelle est la différence la plus importante entre le développement des produits logiciels génériques et sur mesure? Qu'est-ce que cela peut signifier dans la pratique pour les utilisateurs des produits logiciels génériques?
3. Quels sont les quatre attributs importants que tout logiciel professionnel devrait avoir? suggérer quatre autres attributs qui peuvent parfois être significatifs.
4. En donnant des exemples, définir la fiabilité et la performance des logiciels ? donner aussi leurs métriques ?

8. Solutions

I) Quiz :

1:a 2:b 3:c 4:b 5:b 6:a 7:a 8:a 9:a 10:b

II) Questions de recherche :

1. Le Logiciel professionnel est associé à une documentation pour informer le client des différentes étapes d'installation et d'utilisation et même des détails concernant le logiciel (le processus de développement) pour l'aider à bien comprendre le fonctionnement de ce dernier.
   On distingue 3 sortes de documents :
   a) Documentation concernant la spécification, l'architecture et la conception
   b) Documentation technique (code, API, Algos, scripts d'installation et de configuration)
   c) Manuel d'utilisation.

2. Il existe deux types de produits logiciels (génériques et sur mesure)

   - La différence essentielle est que dans le développement de produits logiciels génériques, la spécification est possédée par le développeur du produit. Pour le développement de produits personnalisés (sur mesure), la spécification est détenue et contrôlée par le client.

     Les implications sont importantes - le développeur peut rapidement décider de changer la spécification en réponse à un changement externe (par exemple un produit concurrent), mais, lorsque le client possède les spécifications, les changements doivent

être négociés entre le client et le développeur et peuvent avoir des conséquences contractuelles.

- Pour les utilisateurs de produits génériques, cela signifie qu'ils n'ont aucun contrôle sur la spécification de logiciels et ne peuvent donc pas contrôler l'évolution du produit. Le développeur peut décider d'inclure/exclure des fonctionnalités et changer l'interface utilisateur. Cela pourrait avoir des implications pour les processus métiers de l'utilisateur et ajouter des coûts de formation supplémentaires lorsque de nouvelles versions du système sont installées. Il peut également limiter la flexibilité du client à changer leurs propres processus métiers.

3. Les quatre caractéristiques standards sont :

   La maintenabilité (maintainability), la fiabilité (dependability or reliability), la performance ou l'efficacité (performance), et la utilisabilité (facile à utiliser, convivialité) ou acceptabilité (usability).

   - D'autres attributs peuvent être inclus:
     - La réutilisabilité (reusability) : peut être utilisé dans d'autres applications,
     - la distribuabilité (distributability) : peut être distribué sur un réseau ou sur un ensemble de processeurs,
     - la portabilité (portability) : peut fonctionner sur des plateformes multiples, et
     - l'interopérabilité (interoperability) : peut travailler avec une large gamme de systèmes logiciels.

4. La fiabilité et la performance des logiciels :

   a) Fiabilité :

   Correction, justesse, conformité : le logiciel est conforme à ses spécifications, les résultats sont ceux attendus.
   Robustesse, sureté : le logiciel fonctionne raisonnablement en toutes circonstances, rien de catastrophique ne peut survenir, même en dehors des conditions d'utilisation prévues
   - Quelques métriques:
     - **MTBF** : Mean Time Between Failures
     - **Disponibilité** (pourcentatge du temps pendant lequel le système est utilisable) et Taux d'erreur (nombre d'erreurs par KLOC)

   b) Performance :

   Les logiciels doivent satisfaire aux contraintes de temps d'exécution
   Quelques métriques: Temps d'exécution.

Chapitre II

Processus de Développement Logiciel

Objectifs

- comprendre les concepts des processus logiciels et les modèles de processus logiciel;
- Les trois modèles de processus logiciels génériques et leurs utilisations;
- connaître les activités fondamentales du processus de l'ingénierie des exigences, le développement des logiciels, les tests, et l'évolution;
- comprendre pourquoi les processus devraient être organisés pour faire face aux changements dans les exigences et la conception des logiciels;
- comprendre comment améliorer les processus de développement.

Themes couverts

- Modèles de processus logiciel
- Activités du processus
- Faire face au changement
- Amélioration des processus

Chapitre 2:

Processus de Développement Logiciel

1. Introduction

Le processus logiciel

✧ Un processus logiciel est un ensemble structuré d'activités nécessaires pour développer un système logiciel.

✧ De nombreux processus de logiciels, mais tous impliquent:
- Spécification: définir ce que le système doit faire;
- Conception et mise en œuvre: la définition de l'organisation du système et la mise en œuvre du système;
- Validation: vérifier si le système fait ce que le client veut;
- Evolution: changer le système en réponse à l'évolution des besoins des clients.

✧ Un modèle de processus logiciel est une représentation abstraite d'un processus. Il présente une description d'un processus d'une certaine perspective particulière.

Descriptions de processus logiciel

✧ Lorsque nous décrivons et discutons les processus, nous parlons généralement sur les activités de ces processus tels que la spécification d'un modèle de données, la conception d'une interface utilisateur, etc., et l'ordre de ces activités.

✧ Descriptions de processus peuvent également inclure:
- Produits, qui sont les résultats d'une activité de processus;
- Rôles, qui reflètent les responsabilités des personnes impliquées dans le processus;
- Pré- et post-conditions, qui sont des déclarations qui sont vraies avant et après une activité de processus a été adoptée ou un produit fabriqué.

Les processus planifies et agiles :

✧ Deux catégories (approches) pour les processus de développement des logiciels:

- Processus pilotés par plan (planifié): sont des processus où toutes les activités du processus sont planifiées à l'avance et les progrès sont mesurés sur ce plan.
- Processus agiles, dont la planification est progressive et il est plus facile de modifier le processus afin de refléter l'évolution des besoins des clients.

✧ Dans la pratique, les processus les plus pratiques comprennent des éléments des deux approches planifiée et agiles.

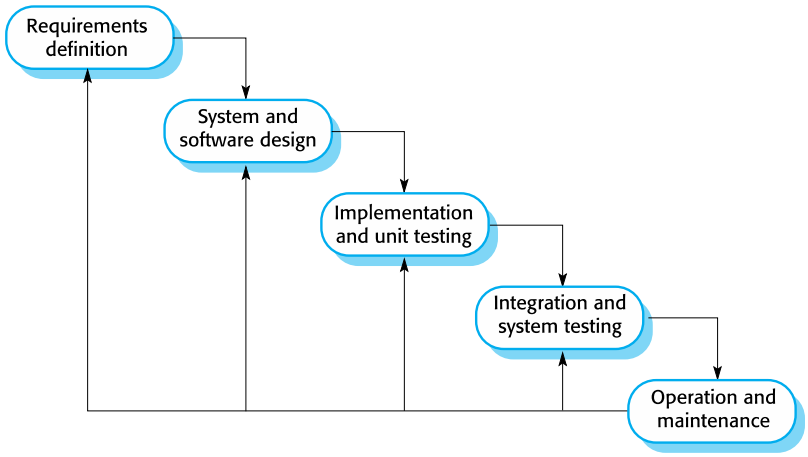
✧ Il n'y a aucun processus logiciel correct ou incorrect.

2. Modèles d'un processus logiciel

✧ Les modèles de processus génériques* (parfois appelés «processus paradigmes») sont:
- Le modèle en cascade (Royce, 1970) : modèle piloté par plan (Plan-driven), phases de spécification et de développement sont distinctes.
- Le développement incrémental : Les phases de Spécification, développement et validation sont entrelacés. Peut-être agile ou planifié.
- Intégration et configuration (réutilisation des composants) : Le système est assemblé à partir des composants configurables. Peut-être agile ou planifié.

✧ Dans la pratique, la plupart des grands systèmes sont développés en utilisant un processus qui intègre des éléments de tous ces modèles.

1) Le modèle en cascade (The waterfall model)



Phases du modèle en cascade

✧ Il y a des phases séparées identifiés dans le modèle en cascade:
- Définition et analyse des besoins
- Conception logicielle du système
- Mise en œuvre et test unitaire
- Intégration et le test du système
- Exploitation et maintenance

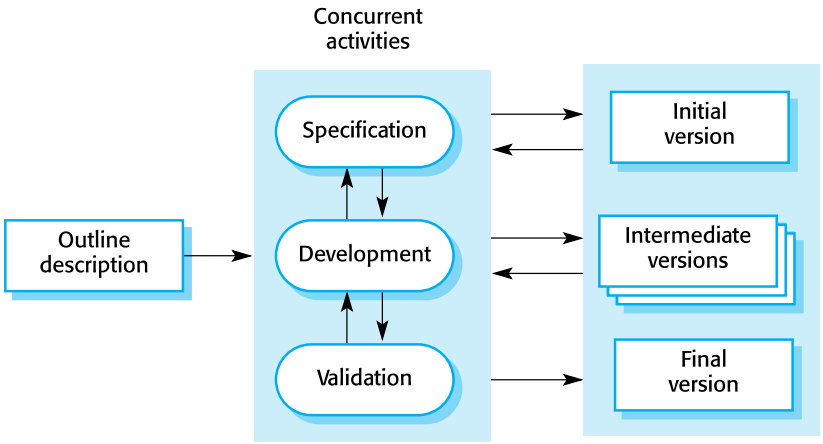
✧ Le principal inconvénient du modèle en cascade est la difficulté à accommoder le changement après que le processus est en cours. En principe, une phase doit être terminée avant de passer à la phase suivante.

Problèmes du modèle en cascade

✧ Partitionnement inflexible du projet en étapes distinctes, il est difficile de répondre à l'évolution des besoins des clients.
- Par conséquent, ce modèle ne convient que si les exigences sont bien comprises et les changements seront assez limitées au cours du processus de conception.
- Peu de systèmes d'entreprises ont des besoins stables.

✧ Le modèle en cascade est surtout utilisé pour les grands projets d'ingénierie des systèmes où un système est développé sur plusieurs sites.
- Dans ces circonstances, la nature du modèle piloté par plan du modèle en cascade permet de coordonner le travail.

2) Le développement incrémental



Avantages du développement incrémental

✧ Le coût d'accueillir des exigences changeantes des clients est réduit.
- La quantité de documentation et d'analyse qui doit être refaite est beaucoup moins que ce qui est requis par le modèle en cascade.

✧ Il est plus facile d'obtenir les commentaires des clients sur le travail de développement qui a été fait.
- Les clients peuvent faire des commentaires sur les démonstrations du logiciel et voir ce qui a été mis en œuvre.

✧ Plus de livraison rapide et déploiement de logiciels utiles au client est possible.
- Les clients sont en mesure d'utiliser et gagner de la valeur à partir du logiciel plus tôt que ce qui est possible avec un processus en cascade.

Problèmes de développement incrémental

✧ Le processus n'est pas visible.
- Les gestionnaires ont besoin des produits régulièrement livrables pour mesurer le progrès. Si les systèmes sont développés rapidement, il est pas rentable de produire des documents qui reflètent toutes les versions du système.

✧ La structure du système tend à se dégrader que les nouvelles augmentations sont ajoutées.
- En plus du temps et de l'argent dépensé sur le reconstruction pour améliorer le logiciel, le changement régulier tend à corrompre sa structure. L'intégration de nouveaux changements des logiciels devient de plus en plus difficile et coûteux.

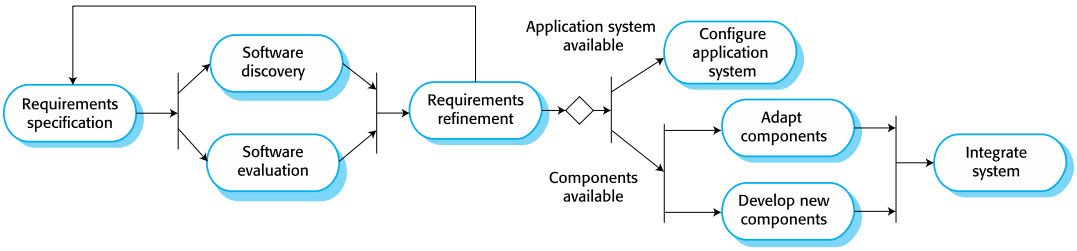
3) Intégration et configuration

✧ Basé sur la réutilisation systématique où les systèmes sont intégrés à partir de composants existants ou les systèmes COTS (Commercial-Off-The-Shelf).

✧ Les éléments réutilisés peuvent être configurés pour adapter leur comportement et leur fonctionnalité aux exigences de l'utilisateur.

✧ La réutilisation est maintenant l'approche standard pour la construction de nombreux types de système d'entreprise.

✧ Étapes clés du processus :
- Spécification des exigences
- Découverte et évaluation des logiciels
- Raffinement des exigences
- Configuration du système d'application
- Adaptation et intégration des composants



Les types des composants logiciels

✧ Systèmes autonomes de logiciels (COTS) qui sont configurés pour une utilisation dans un environnement particulier.

✧ Collections d'objets qui sont développées comme un paquet à être intégré dans un Framework de composants tels que .NET ou J2EE.

✧ Services Web qui sont développés selon les normes de service (l'architecture SOA) et qui sont disponibles pour l'invocation à distance.

Avantages et désavantages

✧ Réduction des coûts et des risques, car moins de logiciels sont développés à partir de zéro.

✧ Livraison et déploiement plus rapide du système.

✧ Mais, les compromis entre les exigences sont inévitables, de sorte que le système ne répond pas aux besoins réels des utilisateurs.

✧ Perte de contrôle sur l'évolution des éléments du système réutilisés.

Activité:

Classer les processus de développement des logiciels suivants selon les modèles étudiés:

- Le modèle en spirale de Boehm (1988)
- Le modèle RUP (Rational Unified Process)
- Le cycle en cascade
- Le cycle en V
- SCRUM
- XP (eXtreme Programming)

3. Les activités d'un processus

✧ Les processus de développement des logiciels sont des séquences d'activités techniques, collaboratives et de gestion avec l'objectif global de la spécification, la conception, la mise en œuvre (implémentation) et le test (validation et vérification) d'un système de logiciel.

✧ Ces activités fondamentales du processus sont organisées différemment dans les différents processus de développement. Dans le modèle en cascade, elles sont organisées en séquence, alors que dans le développement incrémental, elles sont entrelacées.

a) Spécification du logiciel

✧ Le processus d'établissement des services nécessaires et les contraintes sur les opérations du système et le développement.

✧ Processus d'ingénierie des exigences
- Élicitation et analyse des exigences
- Qu'est-ce que les intervenants du système exigent ou attendent du système?
- Spécification des exigences
- Définir les besoins en détail
- Validation des exigences
- Vérification de la validité des exigences

b) Conception et implémentation du logiciel

✧ Le processus de conversion de la spécification du système en un système exécutable.

✧ La conception des logiciels

- Conception d'une structure de logiciel qui réalise la spécification;

✧ Implémentation
- Traduire cette structure dans un programme exécutable;

✧ Les activités de conception et d'implémentation sont étroitement liés et peuvent être entrelacées.

Les activités de conception

✧ La création architecturale, où vous identifiez la structure globale du système, les composants principaux (parfois appelés sous-systèmes ou modules), leurs relations et la façon dont ils sont distribués.

✧ Conception de base de données, où vous concevez les structures de données du système et la façon dont ceux-ci sont à être représenté dans une base de données.

✧ Conception de l'interface, où vous définissez les interfaces entre les composants du système.

✧ Choix et conception de composants, où vous recherchez des composants réutilisables. S'il n'est pas disponible, vous concevez comment cela va fonctionner.

Implémentation du Système

✧ Le logiciel est mis en œuvre soit en développant un programme ou plusieurs programmes, soit en configurant un système d'application.

✧ La conception et l'implémentation sont des activités entrelacées pour la plupart des types de systèmes logiciels.

✧ La programmation est une activité individuelle sans processus standard.

✧ Le débogage est l'activité consistant à trouver des défauts de programme et à corriger ces défauts.

c) Validation et vérification du logiciel

✧ Vérification et validation (V & V) est destiné à montrer que le système est conforme à sa spécification et répond aux exigences de la clientèle du système.

✧ Implique la vérification et la revue des processus et le test du système.

✧ Test du système comprend l'exécution du système avec des cas de test qui sont dérivés à partir de la spécification des données réelles à traiter par le système.

✧ Le test est l'activité la plus couramment utilisée pour V & V.

Niveaux du test

✧ Test du composant:
- Les différents composants sont testés de façon indépendante;
- Les composants peuvent être des fonctions ou des objets ou des groupes cohérents de ces entités.

✧ Test du système:
- Test du système dans son ensemble. Le test des propriétés émergentes est particulièrement important.

✧ Test d'acceptation:
- Test avec les données du client pour vérifier que le système répond aux besoins du client.

d) Évolution du logiciel

✧ Le logiciel est, par nature, flexible et peut changer.

✧ Comme les besoins évoluent à travers l'évolution des circonstances de l'entreprise (business), le logiciel qui supporte l'entreprise doit également évoluer et changer.

✧ Bien qu'il y ait une délimitation entre le développement et l'évolution (Maintenance), cela est de plus en plus sans importance puisque de moins en moins de systèmes sont complètement nouveaux.

4. Faire face au changement

✧ Le changement est inévitable dans tous les grands projets de logiciels.
- Changements d'entreprise (business) mènent aux exigences nouvelles et modifiées du système.
- Les nouvelles technologies ouvrent de nouvelles possibilités pour améliorer l'implémentation.
- Modification des plates-formes nécessitent des changements d'application.

✧ Le changement conduit à retravailler si les coûts du changement comprennent à la fois des retraitements (par exemple ré-analyse des exigences) ainsi que les coûts de l'implémentation de nouvelles fonctionnalités.

Réduire les coûts de la reconstruction

✧ Anticipation du changement, où le processus de logiciel comprend des activités qui peuvent anticiper des changements possibles avant une significative action de retravailler(reconstruire) est nécessaire.
- Par exemple, un prototype de système peut être développé pour montrer quelques caractéristiques principales du système aux clients.

✧ Tolérance au changement, où le processus est conçu de sorte que les changements peuvent être accommodés à un coût relativement faible.
- Ceci implique normalement une certaine forme de développement incrémental. Les modifications proposées peuvent être implémentées dans des incréments qui ne sont pas encore développés. Si cela est impossible, alors un seul incrément (une petite partie du système) peut être modifiée pour incorporer le changement.

Faire face aux exigences changeantes

✧ Le prototypage du système, où une version du système ou une partie du système est développée rapidement pour vérifier les besoins du client et la faisabilité des décisions de conception. Cette approche appuie l'anticipation des changements.

✧ Livraison incrémentale, où les incréments du système sont livrés au client pour commentaires et expérimentations. Cela prend en charge à la fois l'anticipation des changements et la tolérance aux changements.

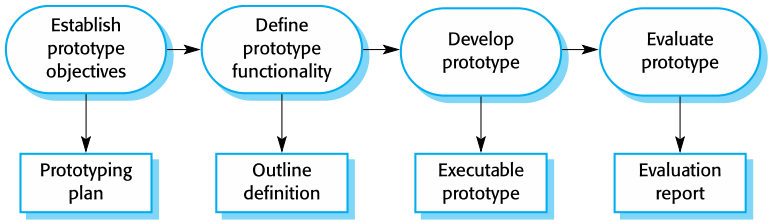
Prototypage de logiciels

✧ Un prototype est une version initiale d'un système utilisé pour démontrer les concepts et d'expérimenter les options de conception.

✧ Un prototype peut être utilisé dans:
- Le processus d'ingénierie des exigences pour aider à l'élicitation et la validation des exigences;
- Les processus de conception pour explorer les options et développer une conception de l'interface d'utilisateur (UI);
- Le processus de test pour exécuter des tests de type back-to-back.

✧ **Avantages du prototypage**
- Amélioration de la facilité d'utilisation du système.
- Une correspondance plus étroite aux besoins réels des utilisateurs.
- Amélioration de la qualité de conception.
- Amélioration de la maintenabilité.
- Réduire l'effort du développement

Le processus de développement d'un prototype



Le développement du prototype

✧ Peut être basé sur des langages de prototypage rapides ou outils.

✧ Peut impliquer la fonctionnalité de sortir
- Prototype devrait se concentrer sur les parties de produits qui ne sont pas bien comprises;
- La vérification des erreurs et la récupération ne peuvent pas être incluses dans le prototype;
- Concentrez-vous sur les exigences fonctionnelles plutôt que non-fonctionnelles tels que la fiabilité et la sécurité

Livraison incrémentale

✧ Plutôt que de livrer le système en une seule livraison, le développement et la livraison est décomposée en incrémentations, et avec chaque incrémentation, une partie de la fonctionnalité requise est livrée.

✧ Les besoins des utilisateurs sont prioritaires et les plus hautes exigences prioritaires sont incluses dans les premières incrémentations.

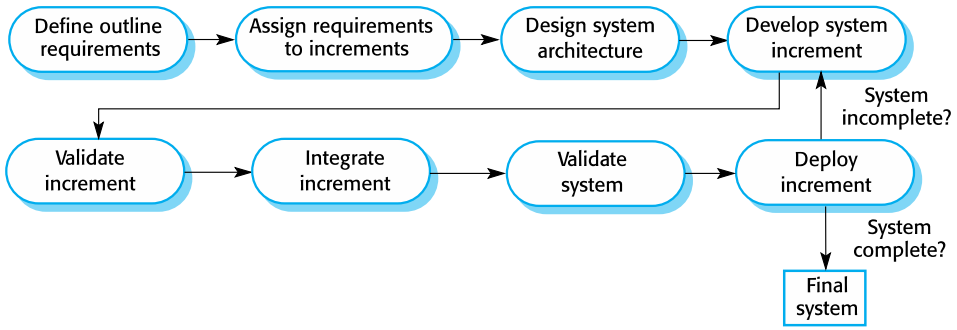
✧ Une fois le développent d'un incrément est lancé, les exigences sont gelés alors que les exigences des incrémentations ultérieures peuvent continuer évoluer.

Développement et livraison incrémentaux

✧ Développement incrémental
- Développer le système par incréments et évaluer chaque incrément avant de procéder à l'élaboration de l'incrément suivant;
- Approche normale utilisée dans les méthodes agiles;
- Évaluation faite par procuration utilisateur/client.

✧ Livraison incrémentale
- Déployer une incrémentation pour être utilisée par les utilisateurs finaux;
- Une évaluation plus réaliste sur l'utilisation pratique des logiciels;
- Difficile à implémenter pour les systèmes de remplacement parce que les incréments ont moins de fonctionnalité que le système à remplacer.

Livraison incrémentale



Avantages de la livraison incrémentale

✧ La valeur du client peut être fournie avec chaque incrément afin que la fonctionnalité du système soit disponible plus tôt.

✧ Les premiers incréments agissent comme un prototype pour aider à éliciter les exigences pour des incréments ultérieurs.

✧ Risque plus faible d'échec global du projet

✧ Les services du système de priorité élevée ont la tendance de recevoir la plupart des tests.

Problèmes de la livraison incrémentale

✧ La plupart des systèmes exigent un ensemble d'installations de base qui sont utilisés par les différentes parties du système.
- Comme les exigences ne sont pas définies en détail jusqu'à un incrément soit implémenté, il peut être difficile d'identifier les installations communes qui sont nécessaires par tous les incréments.

✧ L'essence des processus incrémentaux est que la spécification est développée en conjonction avec le logiciel.
- Cependant, cela est contraire au modèle d'approvisionnement de nombreuses organisations, où la spécification complète du système fait partie du contrat de développement du système.

5. Amélioration des processus

✧ De nombreuses entreprises de logiciels se sont tournées vers l'amélioration des processus logiciels pour améliorer la qualité de leurs logiciels, réduire les coûts ou accélérer leurs processus de développement.

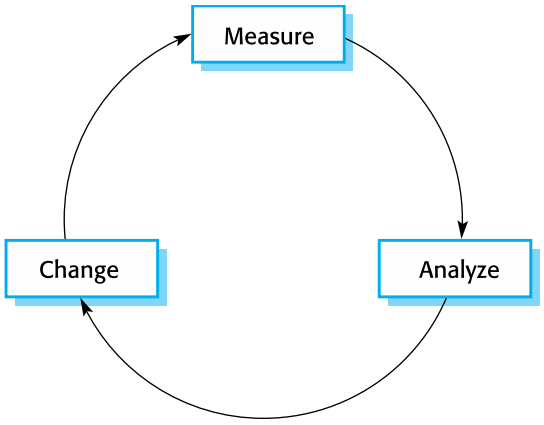
✧ L'amélioration des processus signifie comprendre les processus existants et modifier ces processus pour accroître la qualité des produits et/ou réduire les coûts et le temps de développement.

Approches d'amélioration

✧ L'approche de la maturité du processus, qui met l'accent sur le processus et l'amélioration de la gestion des projets et l'introduction de bonnes pratiques d'ingénierie logicielle.
- Le niveau de maturité des processus reflète dans quelle mesure une bonne technique et pratique de gestion ont été adoptées dans les processus de développement des logiciels organisationnels.

✧ L'approche agile, axée sur le développement itératif et la réduction des frais généraux dans le processus logiciel.
- Les principales caractéristiques des méthodes agiles sont la livraison rapide de la fonctionnalité et la réactivité à l'évolution des besoins des clients.

Le cycle d'amélioration des processus



Activités d'amélioration des processus

✧ Mesure de processus
- Vous mesurez un ou plusieurs attributs du processus logiciel ou du produit. Ces mesures constituent une base de référence qui vous aide à décider si les améliorations de processus ont été efficaces.

✧ L'analyse des processus

- Le processus actuel est évalué et les faiblesses des processus et les goulots d'étranglement sont identifiés. Les modèles de processus (parfois appelés cartes de processus) qui décrivent le processus peuvent être développés.

✧ Changement de processus

- Des changements de processus sont proposés pour répondre à certaines des faiblesses du processus identifiées. Ceux-ci sont introduits et le cycle reprend pour recueillir des données sur l'efficacité des changements.

6. Points clés

✓ Les processus de développement des logiciels sont les activités impliquées dans la production d'un système de logiciel. Modèles de processus logiciel sont des représentations abstraites de ces processus.

✓ Modèles génériques de processus décrivent l'organisation des processus logiciels.

- Des exemples de ces modèles génériques comprennent le modèle en cascade, le développement incrémental, et le développement intégration/configuration.

✓ L'Ingénierie des exigences est le processus d'élaboration de la spécification du logiciel.

✓ La Conception et la mise en œuvre des processus sont concernés par la transformation de la spécification des exigences dans un système de logiciel exécutable.

✓ La Validation du logiciel est le processus de vérification que le système est conforme à sa spécification et qu'il répond aux besoins réels des utilisateurs du système.

✓ L'Évolution du logiciel a lieu lorsque vous changez les systèmes logiciels existants afin de répondre aux nouvelles exigences. Le logiciel doit évoluer pour rester utile.

✓ Les processus devraient inclure des activités telles que le prototypage et la livraison incrémentale (progressive) pour faire face aux changements.

✓ Les processus peuvent être structurés pour le développement et l'exécution itérative de sorte que des **changements peuvent être effectués sans perturber le système dans son ensemble.**

✓ Les principales approches de l'amélioration des processus sont les approches agiles, axées sur la réduction des frais généraux des processus et les approches basées sur la maturité, basées sur une meilleure gestion des processus et l'utilisation de bonnes pratiques d'ingénierie logicielle.

7. Exercices

I) Quiz :

1 : Le modèle de développement des logiciels en cascade est

a. Une approche raisonnable lorsque les exigences sont bien définies.
b. Une bonne approche lorsqu'un programme de travail est requis rapidement.
c. La meilleure approche à utiliser pour les projets avec de grandes équipes de développement.
d. Un modèle ancien qui est rarement utilisé.

2: Le modèle incrémentiel de développement de logiciel est

a. Une approche raisonnable lorsque les exigences sont bien définies.
b. Une bonne approche lorsqu'un produit de base est exigé rapidement.
c. La meilleure approche à utiliser pour les projets avec de grandes équipes de développement.
d. Un modèle révolutionnaire qui n'est pas utilisé pour les produits commerciaux.

3: Modèles de processus évolutifs

a. Ils sont de nature itérative.
b. Peut facilement répondre aux changements des exigences du produit.
c. Ne produisez généralement pas de systèmes jetables.
d. Tout ce qui précède.

4: Le modèle de prototypage de développement des logiciels est

a. Une approche raisonnable lorsque les exigences sont bien définies.
b. Une approche utile lorsqu'un client ne peut pas définir clairement les exigences.
c. La meilleure approche à utiliser pour les projets avec de grandes équipes de développement.
d. Un modèle risqué qui produit rarement un produit significatif.

5: Le modèle en spirale de développement de logiciels

a. Se termine par la livraison du produit logiciel.
b. Est plus chaotique que le modèle incrémental.
c. Comprend l'évaluation des risques du projet au cours de chaque itération.
d. Tout ce qui précède.

6: Le modèle de développement concurrent est

a. Un autre nom pour l'ingénierie concurrente.
b. Définit les événements qui déclenchent les transitions d'état de l'activité d'ingénierie.
c. Utilisé uniquement pour le développement de systèmes parallèles ou distribués.
d. Utilisé chaque fois qu'un grand nombre de demandes de modification sont anticipées.
e. a et b

7: Le modèle de développement à base de composants est

a. Seulement approprié pour la conception du matériel informatique.
b. N'est pas capable de supporter le développement de composants réutilisables.
c. Dépendant de l'approche orientée objet.
d. N'est pas rentable selon les mesures logicielles quantifiables connues.

8: Le modèle des méthodes formelles de développement de logiciels utilise des méthodes mathématiques pour

a. Définir la spécification des systèmes informatiques.
b. Développer des systèmes informatiques sans défaut.
c. Vérifiez l'exactitude des systèmes informatiques.
d. Tout ce qui précède.

9: Laquelle parmi les suivantes n'est pas une phase du modèle RUP (Rational Unified Process)?

a. Phase de création
b. Phase d'élaboration
c. Phase de construction
d. Phase de validation

10: Lequel de ces éléments n'est pas une caractéristique du Processus Logiciel Personnel? (PSP : Personal Software Process)?

a. Met l'accent sur la mesure personnelle du produit de travail.
b. Le praticien exige une supervision minutieuse par le chef de projet.
c. Le praticien individuel est responsable de l'estimation et de la planification.
d. Le praticien a l'habilité de contrôler la qualité des produits logiciels.

11: Quel est l'objectif du Processus Logiciel d'Equipe (TSP :Team Software Process)?

a. Accélérer l'amélioration des processus logiciels
b. Permettre une meilleure gestion de temps par des professionnels hautement qualifiés
c. Créer des équipes de logiciels auto-dirigés
d. Montrer aux gestionnaires comment réduire les coûts et maintenir la qualité
e. b et c

12: Les outils technologiques des processus permettent aux entreprises de logiciels de compresser les plannings en ignorant les activités sans importance.

a. Vrai
b. Faux

13: Il est généralement admis que l'on ne peut pas avoir de processus logiciels faibles et créer des produits finis de haute qualité.

a. Vrai
b. Faux

II) Questions de recherche :

1. Donner les raisons de votre réponse en fonction du type de système en cours de développement, proposer le modèle de processus logiciel générique le plus appropriée qui pourrait être utilisé comme une base pour la gestion de développement des systèmes suivants:

   - Un système pour contrôler le freinage anti-blocage (*Anti-lock Braking System*) dans une voiture.
   - Un système de réalité virtuelle pour soutenir la maintenance des logiciels.
   - Un système de comptabilité de l'université qui remplace un système existant.
   - Un système interactif de planification de voyage qui aide les utilisateurs à planifier leurs voyages avec le plus bas impact sur l'environnement.

2. Expliquer pourquoi le développement incrémental est l'approche la plus efficace pour le développement des systèmes logiciels de commerce. Pourquoi ce modèle est moins approprié à l'ingénierie des systèmes de temps réel?

3. Considérons le modèle de processus basé sur l'intégration et la configuration montré dans Chapitre 2. Expliquer pourquoi il est essentiel d'avoir deux activités séparées des exigences dans le processus.

4. Proposer pourquoi il est important de faire une distinction entre le développement des exigences de l'utilisateur et le développement des exigences du système dans le processus de l'ingénierie des exigences.

5. Décrire les principales activités du processus de conception de logiciels et les sorties (résultats) de ces activités. En utilisant un diagramme, montrer les relations possibles entre les sorties de ces activités.

6. Expliquer pourquoi le changement est inévitable dans systèmes complexes et donner des exemples (en dehors de prototypage et la livraison incrémentielle) des activités de processus logiciel qui aident à prédire les changements et rendre le logiciel étant développé plus résistant aux changements.

7. Expliquer pourquoi les systèmes développés comme prototypes ne devraient, normalement, pas être utilisés comme une production de systèmes.

8. Expliquer pourquoi le modèle en spirale de Boehm est un modèle adaptable qui peut soutenir à la fois les activités de l'évitement du changement et de la tolérance au changement. Dans la pratique, ce modèle n'a pas été largement utilisé. Proposer pourquoi cela pourrait être le cas.

8. Solutions

I) Quiz :

1:a 2:b 3:d 4:b 5:c 6:e 7:c 8:d 9:d 10:b 11:e 12:b 13:a

II) Questions de recherche :

1) Les modèles de processus logiciel générique le plus appropriés pour les systèmes suivants:

   a. **Système de freinage antiblocage** : Ceci est un système de sécurité critique (safety-critical system) qui exige beaucoup d'analyse à l'avance avant la mise en œuvre. Il a certainement besoin d'une approche de développement axée sur plan avec des exigences soigneusement analysés**. Un modèle en cascade est donc le processus le plus approprié** à utiliser, peut-être avec des transformations formelles entre les différentes phases de développement.

   b. **Système de réalité virtuelle :** Ceci est un système où les besoins vont changer et il y aura des composants d'interface utilisateur (UI or GUI) étendus. Le développement incrémental avec, peut-être, certain prototypage de l'interface utilisateur (UI) est le modèle le plus approprié. **Un processus agile peut être utilisé.**

   c. **Système de comptabilité de l'université :** Ceci est un système dont les exigences sont très tôt connues et qui sera utilisé en collaboration avec beaucoup d'autres systèmes tels que le système de gestion des subventions de recherche. Par conséquent, **une approche fondée sur la réutilisation (intégration et configuration) est susceptible d'être approprié** pour cela.

d. **Système Interactif de planification de voyage:** système avec une interface utilisateur complexe (plusieurs questions et réponses entre le système et l'utilisateur), mais qui doit être stable et fiable. **Une approche de développement incrémental est la plus appropriée** et avec le changement dans les exigences du système, l'expérience de l'utilisateur avec le système sera acquise.

3) Dans un processus fondé sur l'intégration et configuration, vous avez besoin de deux activités d'ingénierie des exigences parce qu'il est essentiel de s'adapter aux exigences du système en fonction des capacités systèmes/composants pour être réutilisés. Ces activités sont les suivantes:

   - Une activité initiale où vous comprenez le fonctionnement du système et de définir les besoins généraux de ce que le système doit faire. Ceux-ci devraient être exprimés avec suffisamment de détails que vous pouvez les utiliser comme une base pour décider si le système/composant satisfait certaines exigences et peut donc être réutilisé.
   - Une fois que les systèmes/composants sont sélectionnés, vous avez besoin de plus de l'ingénierie des exigences pour vérifier que les fonctionnalités du logiciel réutilisé répondent aux besoins du système et d'identifier les modifications et les ajouts requis.

4) Il existe une différence fondamentale entre les exigences de l'utilisateur et du système, ce qui signifie qu'ils doivent être considérés séparément.

a) Les exigences de l'utilisateur sont destinées à décrire les fonctions et les caractéristiques du système de point de vue utilisateur et il est essentiel que les utilisateurs comprennent ces exigences. Elles doivent être exprimées en langage naturel et peuvent ne pas être exprimées avec beaucoup de détails, pour permettre une certaine flexibilité d'implémentation. Les personnes impliquées dans le processus doivent pouvoir comprendre l'environnement de l'utilisateur et le domaine d'application.

b) Les exigences du système sont beaucoup plus détaillées que les exigences de l'utilisateur et sont destinées à être une spécification précise du système qui peut faire partie d'un contrat de système. Ils peuvent également être utilisés dans les situations où le développement est sous-traité (externalisé) et l'équipe de développement a besoin d'une spécification complète de ce qui devrait être développé. Les exigences du système sont développées après l'établissement des exigences des utilisateurs.

6) Les systèmes doivent être modifiés car ils sont installés dans un environnement. L'environnement s'adapte à eux et cette adaptation génère naturellement des exigences système nouvelles/différentes. En outre, l'environnement du système est dynamique et produit constamment des nouvelles exigences à la suite de modifications apportées au métier, les objectifs de l'entreprise et les politiques commerciales. Le système devrait être adapté pour tenir compte ces exigences ou il sera moins utile.

   Exemples d'activités de processus qui soutiennent le changement:

   a. **Enregistrement** de la justification des exigences de sorte que la raison pour laquelle une exigence est incluse est connue. Cela aide les changements futurs.

   b. **Traçabilité** des exigences qui montre les dépendances entre les exigences et entre les exigences et la conception/code du système.

c. **Modélisation** de la conception où le modèle de conception documente la structure du logiciel.

d. **Reconstruction** du code qui améliore la qualité du code et le rend plus susceptible à changer.

Chapitre III

Développement Agile des Logiciels

Objectifs

- Comprendre la raison d'être des méthodes agiles de développement logiciel, le manifeste agile et les différences entre le développement agile et piloté plan;
- connaître les pratiques clés dans extreme programming (XP) et comment ils se rapportent aux principes généraux des méthodes agiles;
- comprendre l'approche Scrum pour la gestion agile de projets;
- être conscient des issues et des problèmes de mise à l'échelle des méthodes agiles pour le développement de grands systèmes logiciels.

Themes couverts

- Méthodes agiles
- Techniques de développement agile
- Gestion de projet Agile
- Mise à l'échelle des méthodes agiles

Chapitre 3:

Développement Agile des Logiciels

1. Introduction

Développement rapide de logiciels

✧ Le développement et la livraison rapides sont maintenant souvent l'exigence la plus importante pour les systèmes logiciels
- Les entreprises fonctionnent selon une exigence de changement rapide et il est pratiquement impossible de produire un ensemble d'exigences logicielles stables
- Le logiciel doit évoluer rapidement pour refléter les besoins changeants de l'entreprise.

✧ Le développement axé sur le plan est essentiel pour certains types de systèmes mais ne répond pas à ces besoins commerciaux.

✧ Des méthodes de développement agiles ont émergé à la fin des années 1990, dont l'objectif était de réduire radicalement le délai de livraison pour les systèmes logiciels de travail

Développement Agile

✧ La spécification, la conception et la mise en œuvre du programme sont inter-reliées

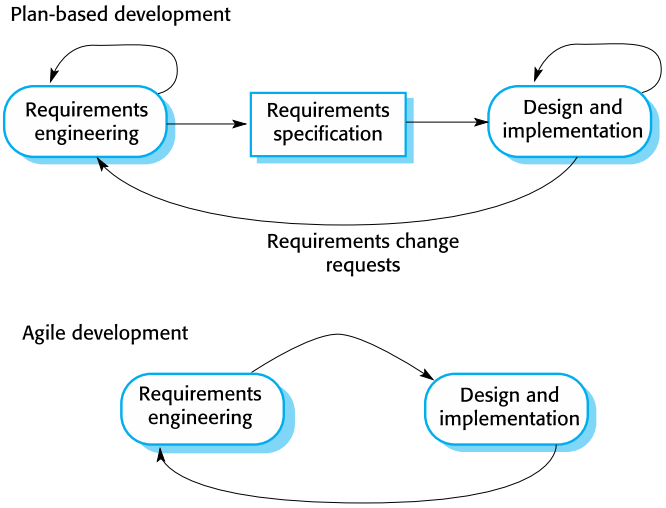
✧ Le système est développé en une série de versions ou d'incréments avec des parties prenantes (stakeholders) impliquées dans la spécification et l'évaluation de la version.

✧ Livraison fréquente de nouvelles versions pour évaluation

✧ Support d'outils étendu (par exemple des outils de test automatisés) utilisés pour supporter le développement.

✧ Documentation minimale – se concentre sur le code de travail

Développement agile et planifié



✧ Développement piloté planifié
- Une approche planifiée de l'ingénierie logicielle repose sur des étapes de développement distinctes avec des résultats qui devant être produits à chacune de ces étapes planifiées à l'avance.
- Pas nécessairement un modèle en cascade - un développement incrémental piloté par plan est possible
- L'itération se produit à l'intérieur des activités.

✧ Développement agile
- La spécification, la conception, la mise en œuvre et les tests sont interreliés et les résultats du processus de développement sont décidés par le biais d'un processus de négociation durant le processus de développement du logiciel.

2. Méthodes Agiles

Méthodes Agiles

✧ L'insatisfaction suscitée par les frais généraux liés aux méthodes de conception de logiciels des années 1980 et 1990 a conduit à la création de méthodes agiles. Ces méthodes:
- Concentrent sur le code plutôt que sur le design
- Sont basés sur une approche itérative du développement de logiciels
- Sont destinés à fournir des logiciels de travail rapidement et évoluer cette rapidité pour répondre aux exigences changeantes.

✧ L'objectif des méthodes agiles est de réduire les frais généraux dans le processus logiciel (par exemple en limitant la documentation) et de pouvoir répondre rapidement aux besoins changeants sans de trop travail.

Le Manifeste agile (http://www.agilemanifesto.org/)

✧ Réunion organisée par Kent Beck en Février 2001
- 17 personnalités, dont les créateurs de Crystal, Scrum, Adaptive Software Development, etc.

✧ Nous découvrons les meilleures façons pour développer un logiciel en le faisant et en aidant les autres à le faire. A travers ce travail, nous en sommes venus à valoriser:

- **Personnes et interaction** plutôt que processus et outils
- **Logiciel fonctionnel** plutôt que documentation complète
- **Collaboration avec le client** plutôt que négociation de contrat
- **Réagir au changement** plutôt que suivre un plan

✧ En fait, bien que les éléments de droite soient importants, nous pensons que les éléments de gauche le sont encore plus.

Les principes des méthodes agiles

Principle	Description
Implication du client (Customer involvement)	Les clients doivent être étroitement impliqués tout au long du processus de développement. Leur rôle est de fournir et de prioriser les nouvelles exigences du système et d'évaluer les itérations du système.
livraison incrémentale (Incremental delivery)	Le logiciel est développé par incréments avec le client en spécifiant les exigences à inclure dans chaque incrément.
Personnes pas processus (People not process)	Les compétences de l'équipe de développement doivent être reconnues et exploitées. Les membres de l'équipe devraient être laissés à développer leurs propres façons de travailler sans processus prescriptifs.
Embrasser le changement (Embrace change)	Attendez-vous les changements dans les exigences du système, et donc concevoir le système pour qu'il puisse tenir compte ces changements.
Maintenir la simplicité (Maintain simplicity)	Concentrez-vous sur la simplicité dans le logiciel en cours de développement et dans le processus de développement. Dans la mesure du possible, travaillez activement pour éliminer la complexité du système.

Applicabilité de la méthode Agile

✧ Développement de produits où une entreprise de logiciels développe un produit de petite ou moyenne taille à vendre.

- Presque tous les produits et applications logiciels sont maintenant développés en utilisant une approche agile

✧ Le développement de systèmes personnalisés au sein d'une organisation, où le client s'engage clairement à s'impliquer dans le processus de développement et où il existe peu de règles et réglementations externes qui affectent le logiciel.

Panorama des méthode Agiles

✧ Beaucoup de méthodes
- eXtreme Programming
- Dynamic Software Development Method
- Adaptive Software Development
- Crystal Clear
- SCRUM
- Feature Driven Development

✧ Choix de la méthode est en fonction de la taille du projet et de l'équipe.

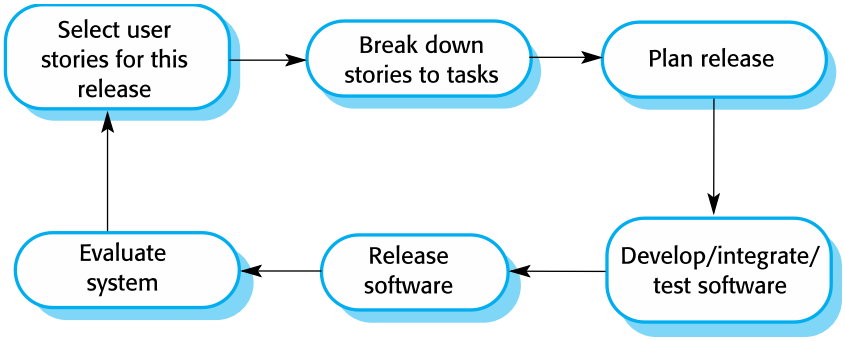
3. Techniques de développement agile

eXtreme Programming

✧ Une méthode agile très influente, développée à la fin des années 1990, qui a introduit une série de techniques de développement agiles.

✧ eXtreme Programming (XP) adopte une approche «extrême» du développement itératif.
- Des nouvelles versions peuvent être construites plusieurs fois par jour;
- Les incréments sont livrés aux clients toutes les 2 semaines;
- Tous les tests doivent être exécutés pour chaque version (build) et la version n'est acceptée que si les tests s'exécutent correctement.

Cycle de sortie (release) d' eXtreme Programming



Pratiques d' eXtreme Programming

Principe ou pratique	Description
Planification incrémentale (Incremental planning)	Les exigences sont enregistrées sur des cartes de « stories » et les « stories » à inclure dans une sortie (release) sont déterminées par le temps disponible et leur priorité relative. Les développeurs décomposent ces stories dans des «tâches» de développement. Voir les figures 3.5 et 3.6.
Petites sorties ou versions (Small releases)	L'ensemble minimal de fonctionnalités utiles qui fournit une valeur commerciale est développé en premier. Les versions du système sont fréquentes et ajoutent progressivement des fonctionnalités à la première version.
Conception Simple (Simple design)	Suffisamment de conception est réalisée pour répondre aux exigences actuelles et pas plus.
Développement piloté par test (test-first development ou test-driven development)	Un framework de test unitaire automatisé est utilisé pour écrire des tests pour une nouvelle pièce de fonctionnalité avant que cette fonctionnalité soit implémentée.
Remaniement de code (refactoring)	Tous les développeurs sont censés ajuster le code en permanence dès que des améliorations de code sont trouvées. Cela permet de garder le code simple et maintenable.
Programmation par paire (Pair programming)	Les développeurs travaillent par paires, vérifient le travail de l'autre et fournissent le support pour toujours faire un bon travail.
Propriété collective (Collective ownership)	Les paires de développeurs travaillent sur toutes les parties du système, et que tous les développeurs prennent la responsabilité de tout le code. Tout le monde peut changer n'importe quoi.
Intégration continue (Continuous integration)	Dès que le travail sur une tâche est terminé, il est intégré à l'ensemble du système. Après une telle intégration, tous les tests unitaires du système doivent passer.
Rythme durable (Sustainable pace)	Les grandes quantités d'heures supplémentaires ne sont pas considérées comme acceptables car l'effet net est souvent de réduire la qualité du code et la productivité à moyen terme
Client sur site (On-site customer)	Un représentant de l'utilisateur final du système (le client) devrait être disponible à plein temps pour l'utilisation de l'équipe XP. Dans un processus de programmation extrême, le client est un membre de l'équipe de développement et est chargé d'apporter les exigences du système à l'équipe pour la mise en œuvre.

XP et principes agiles

✧ Le développement incrémental est pris en charge par de petites versions fréquentes du système.

✧ L'implication du client implique un engagement du client à temps plein avec l'équipe.

✧ Les personnes et pas les processus à travers la programmation par paires, la propriété collective et un processus qui évite les longues heures de travail.

✧ Changement pris en charge par des versions régulières du système.

✧ Maintenir la simplicité grâce à un remaniement (refactoring) constant du code.

Pratiques influentes d'XP

✧ Extreme programming a un accent technique et n'est pas facile à intégrer avec la pratique de gestion dans la plupart des organisations.

✧ Par conséquent, alors que le développement agile utilise des pratiques de XP, la méthode telle que définie à l'origine n'est pas largement utilisée.

✧ Principales pratiques
- Scénarios (stories) d'utilisateurs pour la spécification
- Remaniement (Refactoring)
- Développement piloté par Test
- Programmation par paire (en binôme)

Scénarios (stories) d'utilisateurs pour les exigences

✧ Dans XP, un client ou un utilisateur fait partie de l'équipe XP et il est responsable de prendre des décisions sur les exigences.

✧ Les exigences de l'utilisateur sont exprimées sous la forme de scénarios (stories) utilisateur.

✧ Ceux-ci sont écrits sur des cartes et l'équipe de développement les décompose en tâches d'implémentation. Ces tâches sont la base des estimations de l'échéancier et des coûts.

✧ Le client choisit les stories à inclure dans la prochaine version en fonction de ses priorités et des estimations du calendrier.

Un scenario (story) de «prescription médicale»

Prescribing medication

The record of the patient must be open for input. Click on the medication field and select either 'current medication', 'new medication' or 'formulary'.

If you select 'current medication', you will be asked to check the dose; If you wish to change the dose, enter the new dose then confirm the prescription.

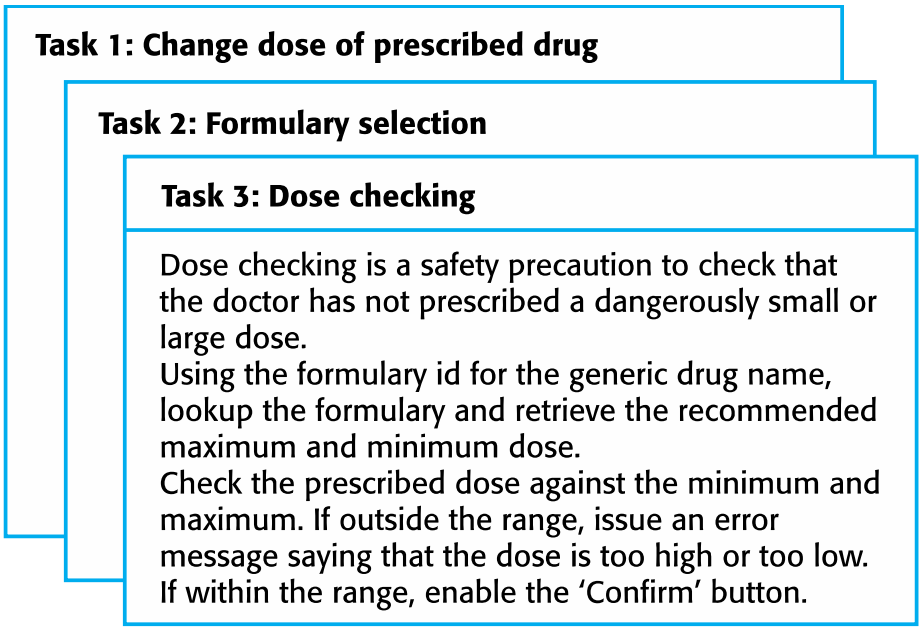
If you choose, 'new medication', the system assumes that you know which medication you wish to prescribe. Type the first few letters of the drug name. You will then see a list of possible drugs starting with these letters. Choose the required medication. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

If you choose 'formulary', you will be presented with a search box for the approved formulary. Search for the drug required then select it. You will then be asked to check that the medication you have selected is correct. Enter the dose then confirm the prescription.

In all cases, the system will check that the dose is within the approved range and will ask you to change it if it is outside the range of recommended doses.

After you have confirmed the prescription, it will be displayed for checking. Either click 'OK' or 'Change'. If you click 'OK', your prescription will be recorded on the audit database. If you click 'Change', you reenter the 'Prescribing medication' process.

Des exemples de cartes de tâches pour la prescription de médicaments



Remaniement (Refactoring)

✧ La sagesse conventionnelle en génie logiciel consiste à concevoir pour le changement. Cela vaut la peine de consacrer du temps et de l'énergie à anticiper les changements, car cela réduit les coûts plus tard dans le cycle de vie.

✧ XP, cependant, soutient que cela ne vaut pas la peine car les changements ne peuvent pas être anticipés de manière fiable.

✧ Au contraire, il propose une amélioration constante du code (refactoring) pour faciliter les changements quand ils doivent être implémentés.

Remaniement (Refactoring)

✧ L'équipe de programmation recherche les améliorations logicielles possibles et apporte ces améliorations même là où il n'y a pas de besoin immédiat.

✧ Cela améliore la compréhensibilité du logiciel et réduit ainsi le besoin de documentation.

✧ Les modifications sont plus faciles à faire car le code est bien structuré et clair.

✧ Cependant, certains changements nécessitent un remaniement (Refactoring) de l'architecture et cela coûte beaucoup plus cher.

Exemples de refactoring

✧ Réorganisation d'une hiérarchie de classes pour supprimer le code en double.

✧ Ranger et renommer les attributs et les méthodes pour les rendre plus faciles à comprendre.

✧ Le remplacement du code en ligne par des appels à des méthodes incluses dans une bibliothèque de programmes.

Développement piloté par les tests (Test-first development, Test-Driven development)

✧ Les tests sont essentiels pour XP et XP a développé une approche où le programme est testé après chaque changement.

✧ Caractéristiques de test XP:
- Développement piloté par les tests ou écrire des test en premier.
- Développement incrémental de tests à partir des scénarios.
- Implication des utilisateurs dans le développement et la validation des tests.
- Les tests automatisés sont utilisés pour exécuter tous les tests de composants à chaque fois qu'une nouvelle version est réalisée.

Développement piloté par les tests

✧ Écrire des tests avant le code clarifie les exigences à mettre en œuvre.

✧ Les tests sont écrits sous forme de programmes plutôt que de données afin de pouvoir être exécutés automatiquement. Le test inclut une vérification qu'il a correctement exécuté.
- Il s'appuie généralement sur un framework de test tel que Junit.

✧ Tous les tests précédents et nouveaux sont exécutés automatiquement lorsque de nouvelles fonctionnalités sont ajoutées, vérifiant ainsi que la nouvelle fonctionnalité n'a pas introduit d'erreurs.

Implication du client

✧ Le rôle du client dans le processus de test est d'aider à développer des tests d'acceptation pour les stories qui seront implémentées dans la prochaine version du système.

✧ Le client qui fait partie de l'équipe écrit des tests au fur et à mesure du développement. Tout nouveau code est donc validé pour s'assurer qu'il est ce que le client a besoin.

✧ Cependant, les personnes qui adoptent le rôle de client ont un temps limité et ne peuvent donc pas travailler à plein temps avec l'équipe de développement. Ils peuvent estimer que fournir les exigences était suffisant d'une contribution et peuvent donc être réticents à s'impliquer dans le processus de test.

Description du cas de test pour la vérification de la dose

Test 4: Dose checking
Input: 1. A number in mg representing a single dose of the drug. 2. A number representing the number of single doses per day. **Tests:** 1. Test for inputs where the single dose is correct but the frequency is too high. 2. Test for inputs where the single dose is too high and too low. 3. Test for inputs where the single dose * frequency is too high and too low. 4. Test for inputs where single dose * frequency is in the permitted range. **Output:** OK or error message indicating that the dose is outside the safe range.

Automatisation des tests

✧ L'automatisation des tests signifie que les tests sont écrits en tant que composants exécutables avant la mise en œuvre de la tâche

- Ces composants de test doivent être autonomes, simuler la soumission des entrées à tester et vérifier que le résultat satisfait aux spécifications de sortie. Un framework de test automatisé (par exemple Junit) est un système qui facilite l'écriture de tests exécutables et la soumission d'un ensemble de tests pour l'exécution.

✧ Comme les tests sont automatisés, il y a toujours un ensemble de tests qui peuvent être exécutés rapidement et facilement

- Chaque fois qu'une fonctionnalité est ajoutée au système, les tests peuvent être exécutés et les problèmes introduits par le nouveau code peuvent être détectés immédiatement.

Problèmes avec le développement piloté par les tests

✧ Les programmeurs préfèrent la programmation que faire tests et parfois ils prennent des raccourcis lors de l'écriture des tests. Par exemple, ils peuvent écrire des tests incomplets qui ne vérifient pas toutes les exceptions possibles qui peuvent se produire.

✧ Certains tests peuvent être très difficiles à écrire progressivement. Par exemple, dans une interface utilisateur complexe, il est souvent difficile d'écrire des tests unitaires pour le code qui implémente la «logique d'affichage» et le flux de travail (workflow) entre les écrans.

✧ Il est difficile de juger de l'exhaustivité d'un ensemble de tests. Bien que vous puissiez avoir beaucoup de tests système, votre ensemble de test peut ne pas fournir une couverture complète.

Programmation par paire

✧ La programmation par paire implique des programmeurs travaillant par paires, développant du code ensemble.

✧ Cela aide à développer une possession commune du code et propage les connaissances à travers

l'équipe.

✧ Il sert comme un processus d'examen informel, chaque ligne de code étant examinée par plus d'une personne.

✧ Il encourage le refactoring car toute l'équipe peut bénéficier de l'amélioration du code système.

Programmation par paire

✧ Dans la programmation par paires, les programmeurs s'assoient ensemble sur le même ordinateur pour développer le logiciel.

✧ Les paires sont créées dynamiquement afin que tous les membres de l'équipe travaillent ensemble pendant le processus de développement.

✧ Le partage des connaissances qui se produit lors de la programmation par paires est très important car il réduit les risques globaux pour un projet lorsque les membres quittent. l'équipe.

✧ La programmation par paire n'est pas nécessairement inefficace et il existe des preuves qui suggèrent qu'une paire travaillant ensemble est plus efficace que deux programmeurs travaillant séparément.

4. Gestion de Projet Agile

Gestion de projet Agile

✧ La responsabilité principale des gestionnaires de projet de logiciel est de gérer le projet afin que le logiciel est livré à temps et dans le budget prévu pour le projet.

✧ L'approche standard de la gestion de projet est axée sur le plan. Les gestionnaires établissent un plan pour le projet montrant ce qui devrait être fourni, quand il devrait être livré et qui travaillera sur le développement des livrables du projet.

✧ La gestion de projet agile nécessite une approche différente, adaptée au développement incrémental et aux pratiques utilisées dans les méthodes agiles.

Gestion de projet Agile

✧ Scrum est une méthode de développement agile orientée projet informatique dont les ressources sont régulièrement actualisées.

✧ La méthode Scrum tire son nom du monde du rugby, scrum = mêlée.

✧ Le principe de base étant d'être toujours prêt à réorienter le projet au fil de son avancement.

Scrum

✧ Scrum (www.scrum.org) est une méthode agile qui se concentre sur la gestion du développement itératif plutôt que sur des pratiques agiles spécifiques.

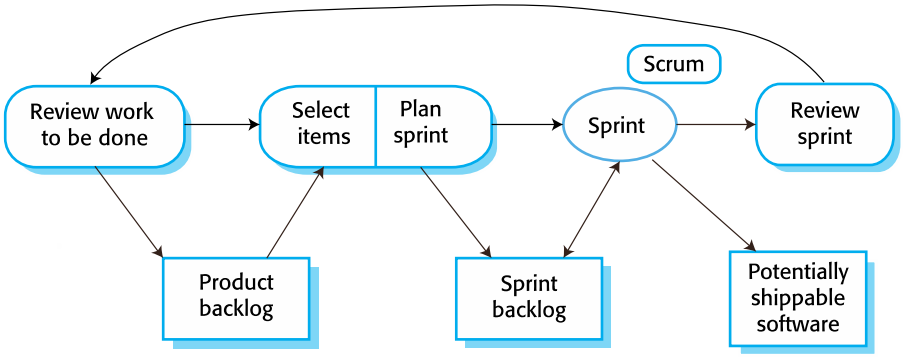
✧ Il y a trois phases dans Scrum:

- La phase initiale est une phase de planification générale dans laquelle vous établissez les objectifs généraux du projet et concevez l'architecture du logiciel.
- Ceci est suivi par une série de cycles de sprint, où chaque cycle développe un incrément du système.
- La phase de clôture du projet achève le projet, complète la documentation requise, comme les cadres d'aide et les manuels de l'utilisateur, et évalue les leçons tirées du projet.

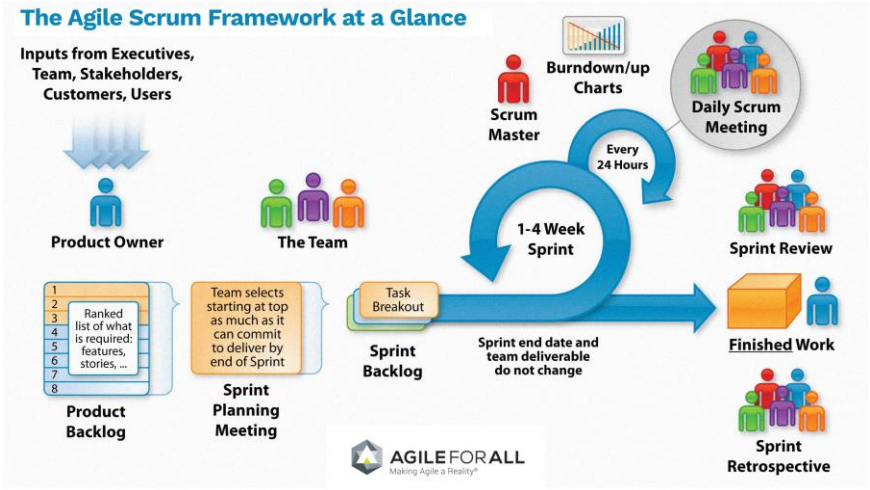
Terminologie Scrum

Terme Scrum	Définition
Équipe de développement (Development team)	Un groupe auto-organisateur de développeurs de logiciels, qui ne devrait pas être plus de 7 personnes. Ils sont responsables du développement du logiciel et d'autres documents essentiels de projet.
produit partiel (ou increment) potentiellement livrable ou utilisable (Potentially shippable product increment)	L'incrément logiciel fourni par un sprint. L'idée est que cela devrait être «potentiellement livrable», ce qui signifie qu'il est dans un état fini et qu'aucun autre travail, tel que le test, n'est nécessaire pour l'intégrer dans le produit final. En pratique, cela n'est pas toujours réalisable.
Carnet de produit (Product Backlog)	Ceci est une liste d'éléments à faire que l'équipe Scrum doit aborder. Il peuvent être des définitions de caractéristiques pour le logiciel, les exigences logicielles, les stories d'utilisateur ou la description des tâches supplémentaires nécessaires, telles que la définition de l'architecture ou la documentation de l'utilisateur.
Propriétaire du produit (Product owner)	Un individu (ou éventuellement un petit groupe) dont le travail consiste à identifier les caractéristiques ou les exigences du produit, à donner propriété pour le développement et à revoir continuellement le Product Backlog pour s'assurer que le projet continue de répondre aux besoins critiques de l'entreprise. Le Product Owner peut être un client mais peut également être un gestionnaire de produit dans une société de logiciels ou un autre représentant des parties prenantes.
Scrum	Une réunion quotidienne de l'équipe Scrum qui examine les progrès et priorise le travail à faire ce jour-là. Idéalement, cela devrait être une courte réunion en face à face qui inclut toute l'équipe.
Scrum Master	Le Scrum Master est responsable pour s'assurer que le processus Scrum est suivi et guide l'équipe dans l'utilisation efficace du Scrum. Il est responsable de l'interface avec le reste de la compagnie et de veiller à ce que l'équipe Scrum ne soit pas détournée par une interférence extérieure. Les développeurs Scrum sont catégoriques que le Scrum Master ne doit pas être considéré comme un gestionnaire de projet. D'autres, cependant, peuvent ne pas toujours trouver facile de voir la différence.
Sprint	Une itération de développement. Les sprints durent généralement de 2 à 4 semaines.
Vélocité (Velocity)	Une estimation de l'effort de Product Backlog qu'une équipe peut couvrir en un seul sprint. Comprendre la vitesse d'une équipe les aide à estimer ce qui peut être couvert dans un sprint et fournit une base pour mesurer l'amélioration des performances.

Cycle de sprint Scrum



Cycle de sprint Scrum détaillé



Le Cycle de sprint Scrum

✧ Les sprints sont de longueur fixe, normalement 2-4 semaines.

✧ Le point de départ de la planification est le product backlog, qui est la liste des travaux à effectuer sur le projet.

✧ La phase de sélection implique toute l'équipe du projet qui travaille avec le client pour sélectionner les caractéristiques et les fonctionnalités du product backlog à développer pendant le sprint.

The Scrum sprint cycle

✧ Une fois que ceux-ci sont convenus, l'équipe s'organise pour développer le logiciel.

✧ Au cours de cette étape, l'équipe est isolée du client et de l'organisation, toutes les communications étant acheminées par le «Scrum Master».

✧ Le rôle du Scrum Master est de protéger l'équipe de développement des distractions externes.

✧ À la fin du sprint, le travail effectué est revu et présenté aux parties prenantes. Le prochain cycle de sprint commence alors.

Travail d'équipe dans Scrum

✧ Le «Scrum Master» est un facilitateur qui organise des réunions quotidiennes, suit le Backlog du travail à faire, enregistre les décisions, mesure le progrès par rapport Backlog et communique avec les clients et la direction en dehors de l'équipe.

✧ Toute l'équipe participe à de courtes réunions quotidiennes (Scrums) où tous les membres de l'équipe partagent des informations, décrivent leurs progrès depuis la dernière réunion, les problèmes qui se sont posés et ce qui est prévu pour le lendemain.

- Cela signifie que tout le monde au sein de l'équipe sait ce qui se passe et, si des problèmes surviennent, peut réorganiser le travail à court terme pour y faire face.

Avantages de Scrum

✧ Le produit est décomposé en un ensemble de morceaux gérables et compréhensibles.

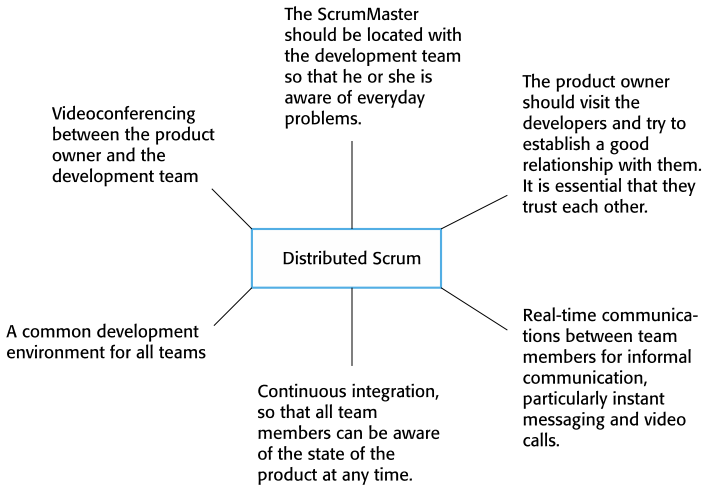
✧ Les exigences instables ne freinent pas le progrès.

✧ Toute l'équipe a la visibilité de tout et par conséquent la communication d'équipe est améliorée.

✧ Les clients voient la livraison des incréments à temps et obtiennent des commentaires sur le fonctionnement du produit.

✧ La confiance entre les clients et les développeurs est établie et une culture positive est créée dans laquelle tout le monde s'attend à ce que le projet réussisse.

Scrum distribué



5. Mise à l'échelle des méthodes agiles

Mise à l'échelle des méthodes agiles

✧ Les méthodes agiles ont été couronnées de succès pour des projets de petite et moyenne taille qui peuvent être développés par une petite équipe co-localisée.

✧ Il est parfois soutenu que le succès de ces méthodes provient d'une meilleure communication, ce qui est possible lorsque tout le monde travaille ensemble.

✧ La mise à l'échelle des méthodes agiles implique de les modifier pour faire face à des projets plus grands et plus longs où il y a plusieurs équipes de développement, travaillant peut-être dans différents localisations.

Extensibilité (Scaling out) et mise à l'échelle (scaling up)

✧ L' Extensibilité concerne l'utilisation de méthodes agiles pour développer de grands systèmes logiciels qui ne peuvent pas être développés par une petite équipe.

✧ La mise à l'échelle s'intéresse à la manière dont les méthodes agiles peuvent être introduites dans une grande entreprise avec de nombreuses années d'expérience en développement de logiciel.

✧ Lors de la mise à l'échelle des méthodes agiles, il est important de maintenir les fondamentaux agiles:
- Planification flexible, versions système fréquentes, intégration continue, développement piloté par les tests et bonnes communications d'équipe.

Problèmes pratiques avec les méthodes agiles

✧ L'informalité du développement agile est incompatible avec l'approche juridique de la définition des contrats couramment utilisée dans les grandes entreprises.

✧ Les méthodes agiles sont plus appropriées pour le développement de nouveaux logiciels plutôt que pour la maintenance des logiciels. Pourtant, la majorité des coûts de logiciels dans les grandes entreprises proviennent de la maintenance de leurs systèmes logiciels existants.

✧ Les méthodes agiles sont conçues pour de petites équipes co-localisées, mais le développement de logiciels implique maintenant des équipes réparties dans le monde entier.

Problèmes contractuels

✧ La plupart des contrats de logiciels pour les systèmes personnalisés sont basés sur une spécification qui définit ce qui doit être implémenté par le développeur du système pour le client du système.

✧ Cependant, cela empêche l'entrelacement entre les spécifications et le développement comme c'est la norme dans le développement agile.

✧ Un contrat qui paie pour le temps de développement plutôt que la fonctionnalité est requis.
- Cependant, cela est considéré comme un risque élevé par nombreux services juridiques parce que ce qui doit être livré ne peut pas être garanti.

Méthodes agiles et maintenance logicielle

✧ La plupart des entreprises dépensent plus pour la maintenance de logiciels existants que pour le développement de nouveaux logiciels. Donc, si les méthodes agiles doivent être couronnées de succès, elles doivent soutenir la maintenance ainsi que le développement original.

✧ Deux questions clés:

- Les systèmes développés en utilisant une approche agile sont-ils maintenables, étant donné l'importance accordée au processus de développement de la minimisation de la documentation officielle?
- Les méthodes agiles peuvent-elles être utilisées efficacement pour faire évoluer un système en réponse à des demandes de changement de clients?

✧ Des problèmes peuvent survenir si l'équipe de développement originale ne peut pas être maintenue.

Maintenance agile

✧ Les principaux problèmes sont les suivants:
- Manque de documentation du produit
- Garder les clients impliqués dans le processus de développement
- Maintenir la continuité de l'équipe de développement

✧ Le développement agile repose sur l'équipe de développement qui sait et comprend ce qui doit être fait.

✧ Pour les systèmes à longue durée de vie, c'est un vrai problème car les développeurs d'origine ne travailleront pas toujours sur le système.

Méthodes agiles et planifiées

✧ La plupart des projets comprennent des éléments de processus agiles et planifiés. Décider de l'équilibre dépend de:
- Est-il important d'avoir une spécification et une conception très détaillées avant de passer à la mise en œuvre? Si c'est le cas, vous devez probablement utiliser une approche axée sur le plan.
- Est-ce une stratégie de livraison incrémentale, où vous livrez le logiciel aux clients et obtenez un retour rapide de leur part, réaliste? Si c'est le cas, envisagez d'utiliser des méthodes agiles.
- Quelle est la taille du système en cours de développement? Les méthodes agiles sont plus efficaces lorsque le système peut être développé avec une petite équipe co-localisée capable de communiquer de manière informelle. Cela peut ne pas être possible pour les grands systèmes qui nécessitent des équipes de développement plus importantes, de sorte qu'une approche axée sur le plan peut devoir être utilisée.

Principes agiles et pratique organisationnelle

Principe	Pratique
Implication du client (Customer involvement)	Cela dépend d'un client désireux et capable de passer du temps avec l'équipe de développement et qui peut représenter toutes les parties prenantes du système. Souvent, les représentants des clients ont d'autres demandes sur leur temps et ne peuvent pas participer pleinement au développement du logiciel. Là où il y a des parties prenantes externes, telles que les régulateurs, il est difficile de représenter leur point de vue à l'équipe agile.
Embrasser le changement (Embrace change)	Prioriser les changements peut être extrêmement difficile, en particulier dans les systèmes pour lesquels il existe de nombreuses parties prenantes. Généralement, chaque partie prenante donne des priorités différentes aux différents changements.

livraison incrémentale (Incremental delivery)	Les itérations rapides et la planification à court terme du développement ne cadrent pas toujours avec les cycles de planification à long terme de la planification des activités et du marketing. Les responsables marketing peuvent avoir besoin de connaître les caractéristiques du produit plusieurs mois à l'avance pour préparer une campagne marketing efficace.
Principe	Pratique
Maintenir la simplicité (Maintain simplicity)	Sous la pression des calendriers de livraison, les membres de l'équipe peuvent ne pas avoir le temps de réaliser les simplifications souhaitables du système.
Personnes pas Processus (People not process)	Les membres individuels de l'équipe peuvent ne pas avoir de personnalités adéquates pour la participation intense typique des méthodes agiles et peuvent donc ne pas bien interagir avec les autres membres de l'équipe.

Issues du Système

✧ Quelle est la taille du système en cours de développement?
- Les méthodes agiles sont plus efficaces qu'une équipe co-localisée relativement petite qui peut communiquer de manière informelle.

✧ Quel type de système est en cours de développement?
- Les systèmes qui nécessitent beaucoup d'analyse avant la mise en œuvre nécessitent une conception assez détaillée pour effectuer cette analyse.

✧ Quelle est la durée de vie prévue du système?
- Les systèmes à longue durée de vie nécessitent une documentation pour communiquer les intentions des développeurs du système à l'équipe de support.

✧ Le système est-il soumis à une réglementation externe?
- Si un système est réglementé, vous devrez probablement produire une documentation détaillée dans le cadre de la procédure de sécurité du système.

Personnes et équipes

✧ Quelle est la qualité des concepteurs et des programmeurs dans l'équipe de développement?
- Il est parfois soutenu que les méthodes agiles nécessitent des niveaux de compétences plus élevés que les approches basées sur le plan dans lesquelles les programmeurs traduisent simplement une conception détaillée en code.

✧ Comment l'équipe de développement est-elle organisée?
- Des documents de conception peuvent être requis si l'équipe est distribuée.

✧ Quelles technologies de support sont disponibles?
- Le support IDE pour la visualisation et l'analyse de programme est essentiel si la documentation de conception n'est pas disponible.

Issues organisationnels

✧ Les organisations d'ingénierie traditionnelles ont une culture du développement basé sur le plan, car c'est la norme en ingénierie.

- ✧ Est-ce une pratique organisationnelle standard d'élaborer une spécification détaillée du système?
- ✧ Les représentants des clients seront-ils disponibles pour fournir des commentaires sur les incréments du système?
- ✧ Le développement agile informel peut-il s'intégrer dans la culture organisationnelle de la documentation détaillée?

Méthodes agiles pour les grands systèmes

- ✧ Les grands systèmes sont généralement des ensembles de systèmes distincts de communication, où des équipes distinctes développent chaque système. Souvent, ces équipes travaillent dans différents endroits, parfois dans des fuseaux horaires différents.
- ✧ Les grands systèmes sont des «brownfield systems», c'est-à-dire qu'ils intègrent et interagissent avec un certain nombre de systèmes existants. La plupart des exigences du système sont concernées par cette interaction et ne se prêtent donc pas vraiment à la flexibilité et au développement incrémental.
- ✧ Lorsque plusieurs systèmes sont intégrés pour créer un système, une partie importante du développement concerne la configuration du système plutôt que le développement du code d'origine.

Développement de grands systèmes

- ✧ Les grands systèmes et leurs processus de développement sont souvent limités par des règles et réglementations externes limitant leur développement.
- ✧ Les grands systèmes ont un long temps d'approvisionnement et de développement. Il est difficile de maintenir des équipes cohérentes qui connaissent le système au cours de cette période car, inévitablement, les personnes passent à d'autres emplois et projets.
- ✧ Les grands systèmes ont généralement un ensemble diversifié de parties prenantes. Il est pratiquement impossible d'impliquer toutes ces différentes parties prenantes dans le processus de développement.

Mise à l'échelle des grands systèmes

- ✧ Une approche complètement progressive de l'ingénierie des exigences est impossible.
- ✧ Il ne peut pas y avoir un seul propriétaire du produit ou représentant du client.
- ✧ Pour le développement de grands systèmes, il n'est pas possible de se concentrer uniquement sur le code du système.
- ✧ Les mécanismes de communication entre équipes doivent être conçus et utilisés.
- ✧ L'intégration continue est pratiquement impossible. Cependant, il est essentiel de maintenir des constructions système fréquentes et des versions régulières du système.

Multi-équipes Scrum

✧ Réplication de rôle
- Chaque équipe a un Product Owner pour son composant de travail et ScrumMaster.

✧ Architectes de produits
- Chaque équipe choisit un architecte produit et ces architectes collaborent pour concevoir et faire évoluer l'architecture globale du système.

✧ Aligner les sorties
- Les dates de sortie des produits de chaque équipe sont alignées afin qu'un système démontrable et complet soit produit.

✧ Scrum of Scrums
- Il y a un Scrum of Scrums quotidien où les représentants de chaque équipe se rencontrent pour discuter des progrès et planifier le travail à faire.

Méthodes agiles dans les organisations

✧ Les gestionnaires de projet qui n'ont pas l'expérience des méthodes agiles peuvent être réticents à accepter le risque d'une nouvelle approche.

✧ Les grandes organisations ont souvent des procédures et des normes de qualité que tous les projets doivent suivre et, en raison de leur nature bureaucratique, elles risquent d'être incompatibles avec les méthodes agiles.

✧ Les méthodes agiles semblent fonctionner mieux lorsque les membres de l'équipe ont un niveau de compétence relativement élevé. Cependant, au sein des grandes organisations, il y aura probablement un large éventail de compétences et de capacités.

✧ Il peut y avoir une résistance culturelle aux méthodes agiles, en particulier dans les organisations qui utilisent depuis longtemps des procédés d'ingénierie des systèmes conventionnels.

6. Points clés

✓ Les méthodes agiles sont des méthodes de développement incrémentielles qui se concentrent sur le développement rapide de logiciels, les versions fréquentes du logiciel, la réduction des frais généraux en minimisant la documentation et en produisant un code de haute qualité.

✓ Les pratiques de développement agiles incluent
- Stories d'utilisateurs pour la spécification du système
- Les versions fréquentes du logiciel,
- Amélioration continue du logiciel
- Développement piloté par les Tests
- Participation du client à l'équipe de développement.

✓ Scrum est une méthode agile qui fournit un cadre de gestion de projet.
- Il est centré autour d'un ensemble de sprints, qui sont des périodes fixes lors du développement d'un incrément système.

✓ De nombreuses méthodes de développement pratiques sont un mélange de développement planifié

et agile.

- ✓ Mettre à l'échelle des méthodes agiles pour les grands systèmes est difficile.
  - Les grands systèmes ont besoin d'une conception initiale et certains documents et pratiques organisationnelles peuvent entrer en conflit avec l'informalité des approches agiles.

7. Exercices

I) Quiz :

1) L'agilité n'est rien de plus que la capacité d'une équipe de projet à réagir rapidement au changement.

a) Vrai
b) Faux

2) Lequel des éléments suivants n'est pas nécessaire pour appliquer l'agilité à un processus logiciel?

a) Éliminer l'utilisation de la planification du projet et les tests
b) Seuls les produits de travail essentiels sont produits
c) Processus permet à l'équipe d'organiser les tâches
d) Utilise une stratégie de livraison de produits incrémentale

3) Comment créez-vous des processus agiles pour gérer l'imprévisibilité?

a) La collecte des exigences doit être menée très soigneusement
b) L'analyse des risques doit être effectuée avant la planification
c) Les incréments de logiciel doivent être livrés dans des périodes de temps courtes
d) Les processus logiciels doivent s'adapter aux changements de manière incrémentale
e) Les deux c et d

4) Dans les processus logiciels agiles, les priorités les plus élevées sont pour satisfaire le client à travers la livraison précoce et continue des versions utiles.

a) Vrai
b) Faux

5) Lesquels des traits suivants doivent exister parmi les membres d'une équipe logicielle agile?

a) Compétence
b) Capacité de prise de décision
c) Confiance mutuelle et respect
d) Tout ce qui précède

6) Dans le développement agile, il est plus important de créer des logiciels qui répondent aux besoins des clients d'aujourd'hui que de se soucier des fonctionnalités qui pourraient être nécessaires dans le futur.

a) Vrai
b) Faux

7) Quelles sont les quatre activités cadres (framework) trouvées dans le modèle de processus eXtreme Programming (XP)?

a) analyse, conception, codage, test
b) planification, analyse, conception, codage
c) planification, analyse, codage, test
d) planification, conception, codage, test

8) Tous les modèles de processus agiles se conforment plus ou moins aux principes énoncés dans le "Manifeste pour le développement des logiciels agiles".

a) Vrai
b) Faux

9) Quelles sont les trois activités cadres (Framework) pour le modèle de processus Adaptive Software Development (ASD)?

a) analyse, conception, codage
b) étude de faisabilité, itération du modèle fonctionnel, mise en œuvre
c) la collecte des exigences, la planification du cycle adaptatif, le développement itératif
d) spéculation, collaboration, apprentissage

10) Laquelle n'est pas l'une des questions clés auxquelles chaque membre de l'équipe doit répondre à chaque réunion Scrum quotidienne?

a) Qu'avez-vous fait depuis la dernière réunion?
b) Quels obstacles créez-vous?
c) Quelle est la cause du problème que vous rencontrez?
d) Que comptez-vous accomplir lors de la prochaine réunion d'équipe?

11) La méthode de développement de systèmes dynamiques (**DSDM** : The Dynamic Systems Development Method) suggère une philosophie basée sur le principe de Pareto (80% de l'application peut être livrée dans 20% du temps nécessaire pour construire l'application complète).

a) Vrai

b) Faux

12) Dans le développement piloté par les fonctionnalités (**FDD** : Feature-Driven Development), une fonctionnalité valorisée par le client est une fonction valorisée par le client qui peut être fournie en deux semaines ou moins.

a) Vrai
b) Faux

13) La modélisation agile (**AM** : Agile Modeling) fournit des conseils au praticien au cours de laquelle de ces tâches logicielles?

a) Analyse
b) Conception
c) Codage
d) Test
e) Les deux A et B

14) Le processus unifié agile (AUP) utilise les activités UP classiques (création, élaboration, construction, transition) pour aider l'équipe à visualiser le flux de processus global.

a) Vrai
b) Faux

15) Dans les modèles de processus agiles, le seul produit de travail livrable est le programme de travail.

a) Vrai
b) Faux

II) Questions de recherche:

1. Expliquez pourquoi la livraison rapide et le déploiement de nouveaux systèmes sont souvent plus importants pour les entreprises que la fonctionnalité détaillée de ces systèmes.
2. Expliquer comment les principes fondamentaux des méthodes agiles conduisent au développement et au déploiement accélérés de logiciels.
3. Extreme programming exprime les exigences des utilisateurs comme des stories, chaque story étant écrite sur une carte. Discutez des avantages et des inconvénients de cette approche de la description des exigences.
4. Quand recommanderiez-vous l'utilisation d'une méthode agile pour développer un système logiciel?
5. Expliquer pourquoi le développement piloté par le test aide le programmeur à développer une meilleure compréhension des exigences du système. Quelles sont les difficultés potentielles avec le développement piloté par le test?
6. Comparer et opposer l'approche Scrum à la gestion de projet avec les approches planifiées conventionnelles. Les comparaisons devraient être basées sur l'efficacité de chaque approche pour

planifier l'allocation des personnes aux projets, estimer le coût des projets, maintenir la cohésion de l'équipe et la gestion des changements dans l'adhésion à l'équipe de projet.

7. Suggérez quatre raisons pour lesquelles le taux de productivité des programmeurs travaillant en binôme pourrait être plus de la moitié de celui de deux programmeurs travaillant individuellement.
8. Pourquoi est-il nécessaire d'introduire des méthodes et de la documentation à partir d'approches basées sur des plans lors de la mise à l'échelle de méthodes agiles pour des projets plus importants développés par des équipes de développement distribuées.
9. Vous êtes un gestionnaire de logiciels dans une entreprise qui développe des logiciels de contrôle critiques pour les avions. Vous êtes responsable du développement d'un système d'aide à la conception de logiciels qui prend en charge la traduction des exigences logicielles en une spécification logicielle formelle. Commentez les avantages et les inconvénients des stratégies de développement suivantes:

    a) Recueillir les exigences d'un tel système auprès des ingénieurs logiciels et des parties prenantes externes (telles que l'autorité de certification réglementaire) et développer le système en utilisant une approche planifiée.

    b) Développez un prototype utilisant un langage de script, tel que Ruby ou Python, évaluez ce prototype avec des ingénieurs logiciels et d'autres parties prenantes, puis examinez les exigences du système. Redévelopper le système final en utilisant Java.

    c) Développer le système en Java en utilisant une approche agile avec un utilisateur impliqué dans l'équipe de développement.
10. Il a été suggéré que l'un des problèmes d'avoir un utilisateur étroitement impliqué dans une équipe de développement de logiciels est qu'ils «vont natifs»; c'est-à-dire qu'ils adoptent les perspectives de l'équipe de développement et perdent la vue des besoins de leurs collègues utilisateurs. Suggérez trois façons de contourner ce problème et discutez des avantages et des inconvénients de chaque approche.
11. Pour réduire les coûts et l'impact environnemental des déplacements quotidiens, votre entreprise décide de fermer un certain nombre de bureaux et d'aider le personnel à travailler à domicile. Cependant, la haute direction qui introduit la politique ne sait pas que le logiciel est développé en utilisant des méthodes agiles, qui reposent sur un travail d'équipes étroites et la programmation par paires. Discutez des difficultés que cette nouvelle politique pourrait causer et de la façon dont vous pourriez contourner ces problèmes

8. Solutions

I) Quiz :

1:b 2:a 3:e 4:a 5:d 6:a 7:d 8:a 9:d 10:c 11:a 12:b 13:e 14:a 15:b

II) Questions de recherche:

2. Les principes fondamentaux du développement agile sont les suivants:

a) **Personnes et interaction plutôt que processus et outils.** En prenant les avantages des compétences individuelles et en veillant à ce que l'équipe de développement sache ce que font les autres, les frais généraux de communication formelle et d'assurance de processus sont évités. Cela signifie que l'équipe peut se concentrer sur le développement de logiciels de travail.

b) **Logiciel fonctionnel plutôt que documentation complète.** Cela contribue à accélérer le développement parce que le temps n'est pas passé à se développer, vérifier et gérer la documentation. Le temps du programmeur est plutôt axé sur le développement et le test du code.

c) **Collaboration avec le client plutôt que négociation de contrat.** Plutôt que de passer du temps à développer, analyser et négocier les exigences à inclure dans un contrat système, les développeurs agiles soutiennent qu'il est plus efficace d'obtenir des commentaires des clients directement au cours du développement sur ce qui est requis. Cela permet de développer et de livrer des fonctionnalités utiles plus tôt que cela ne serait possible si des contrats étaient requis.

d) **Réagir au changement plutôt que suivre un plan.** Les développeurs agiles soutiennent (à juste titre) qu'être attentif aux changements est plus efficace que de suivre un processus basé sur un plan parce que le changement est inévitable quel que soit le processus utilisé. Il y a des frais généraux importants dans la modification des plans pour s'adapter au changement et l'inflexibilité d'un plan signifie que le travail peut être fait qui est ensuite mis au rebut.

3.

Avantages des stories:

1. Ils représentent des situations réelles qui se posent généralement afin que le système soutenir les opérations les plus courantes de l'utilisateur.
2. Il est facile pour les utilisateurs de comprendre et de critiquer les stories.
3. Ils représentent des incréments de fonctionnalité - la mise en œuvre d'un story délivre une certaine valeur pour l'utilisateur.

Inconvénients des stories

1. Ils sont susceptibles d'être incomplètes et leur nature informelle rend l'incomplétude difficile à détecter.
2. Ils se concentrent sur les exigences fonctionnelles plutôt que les non-fonctionnelles.
3. Représentation des exigences transversales du système telles que la performance et la fiabilité est impossible lorsque les stories sont utilisées.
4. La relation entre l'architecture du système et les stories d'utilisateurs est peu claire et donc, la conception architecturale est difficile.

6.

Planification de l'allocation des personnes aux projets

Scrum

Scrum gère l'allocation des personnes de manière informelle. Les membres de l'équipe font une offre pour les fonctionnalités du backlog de produit à mettre en œuvre s'ils pensent que leur expertise est appropriée. Alternativement, les tâches peuvent être attribuées par le maître Scrum.

Il n'y a pas de mécanisme formel dans Scrum pour la planification des membres du projet ayant une expertise très spécifique à affecter temporairement à une équipe. Ce besoin doit être identifié par le maître Scrum et il doit discuter de la manière dont l'expertise peut être mise à disposition.

Développement basé sur un plan

Un plan de projet est utilisé pour identifier les parties du système à livrer et celles-ci sont spécifiées dans le document d'exigences. L'expertise requise pour chaque partie peut alors être identifiée et l'affectation des personnes aux projets prévus sur cette base.

Estimation des coûts du projet

Scrum

Les coûts du projet sont estimés en fonction de la date de livraison requise pour le logiciel et les personnes travaillant dans l'équipe Scrum. La fonctionnalité du système est ajustée pour qu'un système de travail soit toujours livré pour l'estimation du coût original. Bien sûr, cela peut ne pas convenir au client et il doit s'impliquer dans le rééchelonnement de la livraison du système.

Développement basé sur un plan

Les coûts du projet sont basés sur une analyse des fonctionnalités spécifiées dans le document d'exigences ainsi que sur les exigences non fonctionnelles du système. Ils peuvent être ajustés pour refléter la taille de l'équipe et le calendrier de livraison. Il est normal que les coûts soient sous-estimés et que le projet final coûte beaucoup plus cher que prévu. Un coût moyen pour les membres de l'équipe est supposé.

Maintenir la cohésion de l'équipe

Scrum

Les membres de l'équipe se rencontrent quotidiennement en face à face ou par voie électronique. Des discussions informelles et des communications approfondies sont encouragées. Les membres de l'équipe négocient le travail à effectuer à partir le backlog du projet. Tout cela conduit à un sentiment partagé de propriété du produit et à une équipe cohésive (très soudée).

Développement basé sur un plan

La cohésion d'équipe est la responsabilité du chef de projet et il doit prendre des actions explicites pour l'encourager. L'approche générale repose sur des réunions formelles relativement peu fréquentes et qui ne conduisent pas au développement d'une équipe cohésive

Gestion des modifications dans l'appartenance à une équipe de projet

Scrum

C'est un sujet qui est rarement abordé dans Scrum mais qui est un problème fondamental parce que tant d'informations sont informelles et dépendent des gens qui se souviennent de ce qui a été convenu. Quand quelqu'un part, il peut être très difficile de mettre à jour un membre de l'équipe de remplacement, surtout si la documentation du projet est très limitée.

Développement basé sur un plan

Le plan de gestion du projet est basé sur l'expertise plutôt que sur les individus et les documents de projet devraient être disponibles. Par conséquent, si un membre de l'équipe quitte, alors un nouveau membre de l'équipe avec une expertise comparable peut lire ce qui a été fait et, après avoir compris cela, devrait pouvoir servir de remplaçant.

8.

1. La planification de projet est souvent essentielle lors du développement de logiciels avec des équipes plus importantes pour (a) s'assurer que les bonnes personnes sont disponibles quand elles sont nécessaires pour être impliquées dans le processus de développement et (b) s'assurer que les livraisons des différentes parties du système développé par différentes équipes sont alignés. Cela signifie que si la partie A dépend de la partie B, le calendrier devrait s'assurer que la partie B est développée avant la partie A.

2. L'analyse des besoins et la documentation sont importantes pour décider comment répartir le travail entre les équipes et s'assurer que chaque équipe a une certaine compréhension de ce que font les autres équipes.

3. La documentation de conception, en particulier les spécifications de l'interface, est importante pour que les équipes puissent se développer indépendamment sans avoir accès à un logiciel en cours de développement.

4. La gestion des risques peut être nécessaire pour s'assurer que toutes les équipes comprennent les risques encourus et peuvent organiser leur travail pour minimiser ces risques. La gestion des risques peut également être utile pour gérer les différents calendriers de livraison utilisés par différentes équipes.

10.

1. Impliquer plusieurs utilisateurs dans l'équipe de développement. Les avantages sont que vous obtenez plusieurs points de vue sur le problème, une meilleure couverture des tâches de l'utilisateur et donc des exigences et moins de chances d'avoir un utilisateur atypique (Sans type déterminé.). Les désavantages sont le coût, les difficultés à obtenir l'engagement de l'utilisateur et les éventuels conflits d'utilisateurs.

2. Changer l'utilisateur qui est impliqué dans l'équipe. Les avantages sont, encore une fois, des perspectives multiples. Les désavantages sont que chaque utilisateur prend du temps pour être productif et des exigences contradictoires possibles de différents utilisateurs.

3. Validez les suggestions de l'utilisateur avec les autres représentants des utilisateurs. Les avantages sont la vérification indépendante des suggestions; désavantage est que cela ralentit le processus de développement car il faut du temps pour faire les vérifications.

Chapitre IV

Ingénierie des Exigences

Objectifs

- comprendre les concepts de l'utilisateur et des exigences du système et pourquoi ces exigences devraient être écrites de différentes manières;
- comprendre les différences entre les exigences logicielles fonctionnelle et non fonctionnelle;
- comprendre les principales activités d'ingénierie des exigences de élicitation, Spécification (analyse) et la validation, et les relations entre ces activités;
- comprendre comment les exigences peuvent être organisées dans un document des exigences du logiciel;

Themes couverts

- Exigences fonctionnelles et non fonctionnelles
- Processus d'ingénierie des exigences
- Elicitation des exigences
- Spécification des exigences
- Validation des exigences
- Changement des exigences

Chapitre 4:

Ingénierie des Exigences

1. Introduction

L'ingénierie des exigences

✧ Le processus d'établissement des services que le client a besoin d'un système et les contraintes sous lesquelles il fonctionne et il est développé.

✧ Les exigences elles-mêmes sont les descriptions des services de système et les contraintes qui sont générées au cours du processus d'ingénierie des exigences.

Qu'est-ce qu'une exigence?

✧ Cela peut aller d'une déclaration abstraite de haut niveau d'un service ou d'une contrainte de système à une spécification fonctionnelle mathématique détaillée.

✧ Ceci est inévitable car les exigences peuvent remplir une double fonction

- Peuvent-être la base d'une offre pour un contrat - doivent donc être ouvertes à l'interprétation;
- Peuvent-être la base pour le contrat lui-même - doivent donc être défini en détail;
- Ces deux déclarations peuvent être appelés exigences.

L'abstraction des exigences (Davis, 1993)

✧"If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system." [Davis, A. M. (1993). Software Requirements: Objects, Functions and States. Englewood Cliffs, NJ:Prentice Hall]

Types des exigences

✧Les exigences de l'utilisateur

- Déclarations en langage naturel avec les diagrammes de services fournis par le système et ses contraintes opérationnelles. Écrit pour les clients.

✧Les exigences du système

- Un document structuré établissant des descriptions détaillées sur les fonctions, les services et les contraintes opérationnelles du système. Définit ce qui devrait être mis en œuvre et peut donc faire partie d'un contrat entre le client et le fournisseur (contracteur).

Les exigences de l'utilisateur et du système

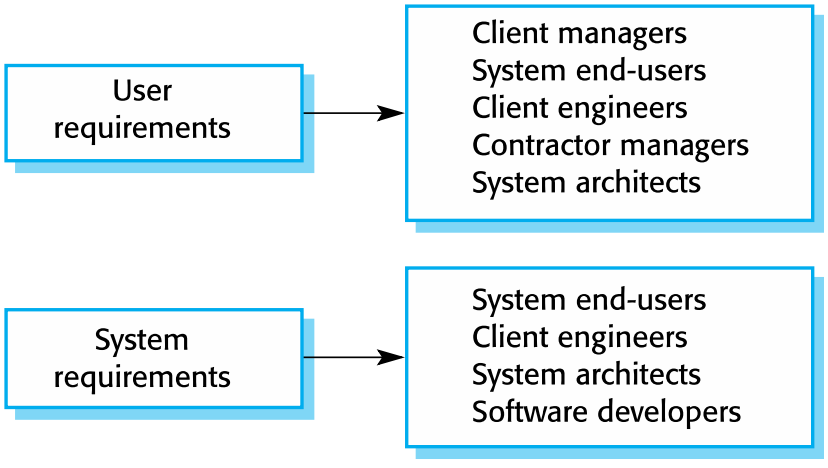
User requirements definition

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Les lecteurs des différents types de spécification des exigences



Parties prenantes (Stakeholders) du système

✧Toute personne ou organisation qui est touchée par le système d'une manière ou d'une autre et qui a un intérêt légitime

✧Types d'intervenants

- Les utilisateurs finaux
- Les gestionnaires de système
- Propriétaires de système
- Parties prenantes externes

Parties prenantes du système Mentcare

✧Les patients dont les informations sont enregistrées dans le système.

✧Les médecins qui sont responsables de l'évaluation et du traitement des patients.

✧Les infirmières qui coordonnent les consultations avec les médecins et administrent certains traitements.

✧Réceptionnistes médicaux qui gèrent les rendez-vous des patients.

✧Le personnel informatique responsable de l'installation et de la maintenance du système.

✧Un gestionnaire de l'éthique médicale qui doit s'assurer que le système respecte les lignes directrices éthiques actuelles en matière de soins aux patients.

✧Les gestionnaires de soins de santé qui obtiennent des informations de gestion du système.

✧Le personnel des dossiers médicaux est responsable de veiller à ce que les informations du système puissent être maintenues et préservées, et à ce que les procédures de tenue (sauvegarder) de dossiers soient correctement mises en œuvre.

Les méthodes agiles et les exigences

✧De nombreuses méthodes agiles font valoir que la production d'exigences détaillées du système est une perte de temps car les exigences changent si rapidement.

✧Le document d'exigences est donc toujours périmé.

✧Les méthodes agiles utilisent généralement l'ingénierie des exigences incrémentales et peuvent exprimer des exigences en tant que «user stories» (voir le chapitre 3).

✧Ceci est pratique pour les systèmes de métier (d'entreprise), mais problématique pour les systèmes qui nécessitent une analyse avant livraison (par exemple, des systèmes critiques) ou des systèmes développés par plusieurs équipes.

2. Exigences fonctionnelles et non fonctionnelles

Les exigences fonctionnelles et non fonctionnelles

✧Les exigences fonctionnelles

- Les services que le système devrait fournir, comment le système devrait réagir à des entrées particulières et comment le système devrait se comporter dans des situations particulières.
- Peut dire ce que le système ne devrait pas faire.

✧Exigences non-fonctionnelles

- Les contraintes sur les services ou les fonctions offertes par le système telles que des contraintes de temps, les contraintes sur le processus de développement, normes, etc.
- Elles s'appliquent souvent au système dans son ensemble plutôt qu'à des fonctionnalités ou services individuels.

✧Exigences du domaine

- Contraintes sur le système à partir du domaine d'opération.

Les exigences fonctionnelles

✧Décrire les fonctionnalités et les services du système.

✧Dépend du type de logiciel, des utilisateurs attendus et du type de système où le logiciel est utilisé.

✧Les exigences des utilisateurs fonctionnelles peuvent être des déclarations de haut niveau sur ce que le système devrait faire.

✧Les exigences fonctionnelles du système devraient décrire les services du système en détail.

Système Mentcare: exigences fonctionnelles

✧Un utilisateur doit être en mesure de rechercher les listes de rendez-vous pour toutes les cliniques.

✧Le système doit générer chaque jour, pour chaque clinique, une liste de patients qui doivent assister à des rendez-vous ce jour-là.

✧Chaque membre du personnel utilisant le système doit être identifié de manière unique par son numéro d'employé de 8 chiffres.

L'imprécision des exigences

✧Les problèmes surviennent lorsque les exigences ne sont pas précisément indiquées.

✧Exigences ambiguës peuvent être interprétées de différentes manières par les développeurs et les utilisateurs.

✧Considérons le terme «rechercher» dans l'exigence 1

- Intention de l'utilisateur : recherche d'un nom du patient pour tous les rendez-vous dans toutes les cliniques;
- Interprétation du développeur : la recherche d'un nom du patient dans une clinique individuel. L'utilisateur choisit la clinique puis fait la recherche.

Complétude et cohérence des exigences

✧En principe, les exigences devraient être à la fois complètes et cohérentes.

✧Complete

- Elles devraient inclure les descriptions de toutes les exigences nécessaires.

✧Cohérente

- Il ne devrait pas y avoir aucun conflit ou des contradictions dans les descriptions des exigences du système.

✧Dans la pratique, en raison de la complexité du système et de l'environnement, il est impossible de produire un document des exigences complet et cohérent.

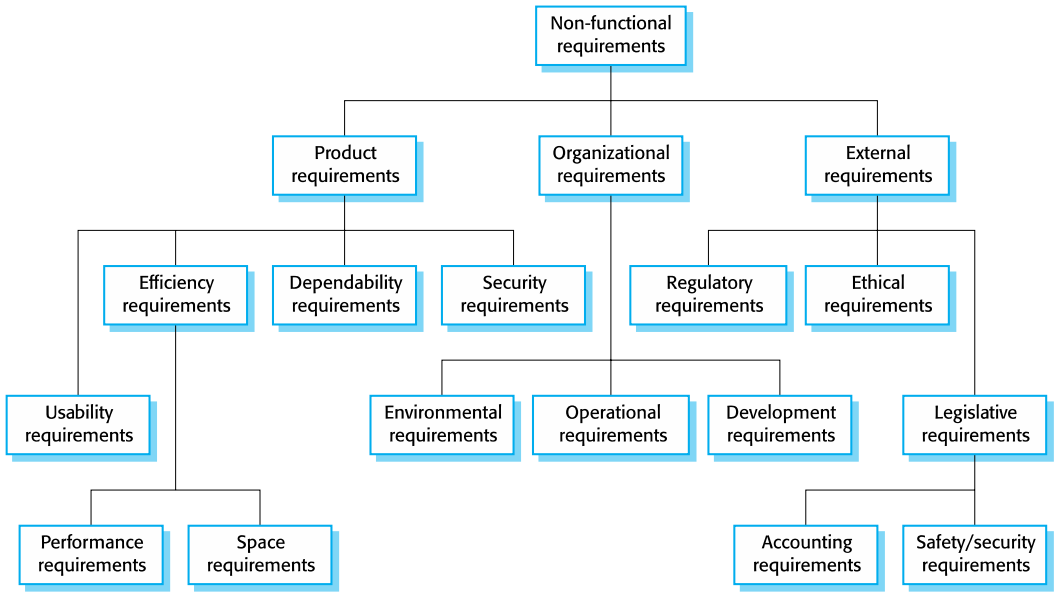
Les exigences non fonctionnelles

✧Celles-ci définissent les propriétés et les contraintes système, par exemple la fiabilité, temps de réponse et les besoins de stockage. Les contraintes sont sur les capacités des dispositifs d'entrées/sortie, les représentations du système, etc.

✧Exigences du processus peuvent aussi être spécifiées en mandatant un IDE particulier, langage de programmation ou une méthode de développement.

✧Exigences non fonctionnelles peuvent être plus critiques que les exigences fonctionnelles. Si celles-ci ne sont pas remplies, le système peut être inutile.

Types d'exigences non fonctionnelles



La mise en œuvre des exigences non fonctionnelles

✧Les exigences non fonctionnelles peuvent affecter l'architecture globale d'un système plutôt que les composants individuels.

- Par exemple, pour s'assurer que les exigences de performance sont remplies, vous pouvez organiser le système afin de minimiser les communications entre les composants.

✧ Une exigence non-fonctionnelle unique, comme une exigence de sécurité, peut générer un certain nombre d'exigences fonctionnelles connexes qui définissent les services du système qui sont requises.

- Il peut également générer des exigences qui restreignent les exigences existantes.

Classifications des exigences non-fonctionnelles

✧Exigences de produit

- Exigences qui précisent que le produit livré doit se comporter d'une manière particulière par exemple, la vitesse d'exécution, la fiabilité, etc.

✧Exigences organisationnelles

- Exigences qui sont une conséquence des politiques et procédures organisationnelles par exemple, les standards des processus utilisés, les exigences de mise en œuvre, etc.

✧Exigences externes

- Exigences qui découlent de facteurs qui sont externes au système et ses exigences par exemple, les exigences d'interopérabilité, les exigences législatives, etc.

Exemples d'exigences non fonctionnelles dans le système Mentcare

Product requirement

The Mentcare system shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement

Users of the Mentcare system shall authenticate themselves using their health authority identity card.

External requirement

The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Objectifs et exigences

✧Un problème commun aux exigences non fonctionnelles est que les utilisateurs ou les clients proposent souvent ces exigences sous forme d'objectifs généraux, tels que la facilité d'utilisation, la capacité du système à récupérer après une défaillance ou la réponse rapide de l'utilisateur.

✧Les objectifs énoncent de bonnes intentions des utilisateurs du système, mais posent des problèmes aux développeurs de systèmes car ils laissent une marge d'interprétation et de litige une fois le système livré.

✧Les exigences non fonctionnelles peuvent être très difficile pour les énoncer avec précision, et les exigences imprécises peuvent être difficiles à vérifier.

✧Objectif(but)

- Une intention générale de l'utilisateur tel que la facilité d'utilisation.

✧Exigence non-fonctionnelle vérifiable

- Une déclaration en utilisant certaine mesure qui peut être objectivement testé.

Exemple: Exigences d'utilisabilité

✧Le système devrait être facile à utiliser par le personnel médical et devrait être organisée de telle sorte que les erreurs de l'utilisateur sont réduits au minimum. (Objectif)

✧Le personnel médical doit pouvoir utiliser toutes les fonctions du système après quatre heures de formation. Après cette formation, le nombre moyen d'erreurs commises par les utilisateurs expérimentés ne doit pas dépasser deux par heure d'utilisation du système. (Exigence non-fonctionnelle vérifiable)

Métriques pour spécifier les exigences non fonctionnelles

Propriété	Mesure
Vitesse (Speed)	Processed transactions/second User/event response time Screen refresh time
Taille (Size)	Mbytes Number of ROM chips
Facilité d'utilisation (Ease of use)	Training time Number of help frames
Fiabilité (Reliability)	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustesse (Robustness)	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portabilité (Portability)	Percentage of target dependent statements Number of target systems

3. Processus d'ingénierie des exigences

Processus d'ingénierie des exigences

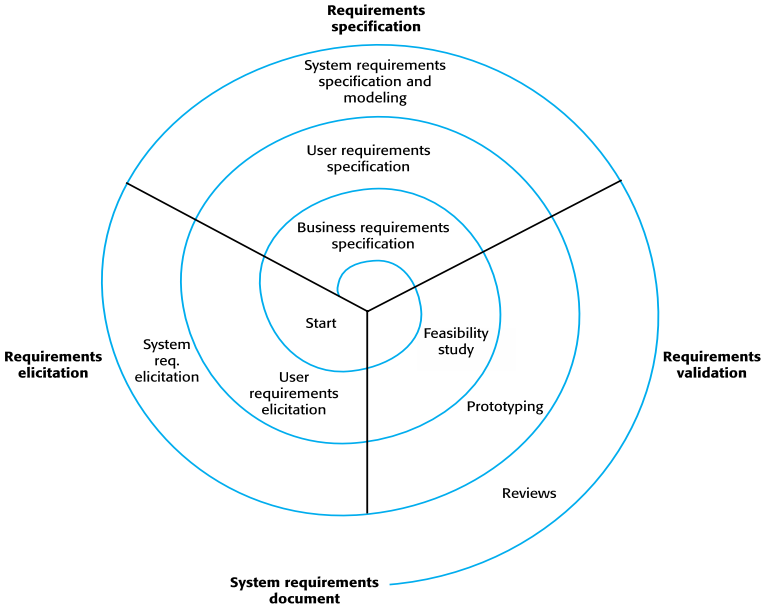
✧Les processus utilisés pour RE (Requirements engineering) varient largement en fonction du domaine d'application, les personnes impliquées et l'organisation d'élaboration des exigences.

✧Cependant, il y a un certain nombre d'activités génériques communes à tous les processus

- Elicitation (Extraction) des exigences;
- Analyse (specification) des exigences;
- Validation des exigences;
- Gestion des exigences.

✧Dans la pratique, RE est une opération itérative dans lequel ces activités sont entrelacés.

Une vue spirale du processus d'ingénierie des exigences



4. Elicitation des exigences

Elicitation et analyse des exigences

✧Implique le personnel technique travaillant avec les clients pour se renseigner sur le domaine d'application, les services que le système devrait fournir et les contraintes opérationnelles du système.

✧Peut impliquer les utilisateurs finaux, les gestionnaires, les ingénieurs impliqués dans l'entretien, les experts du domaine, les syndicats, etc. Ils sont appelés parties prenantes (stakeholders).

✧Les ingénieurs logiciel travaillent avec un éventail de parties prenantes du système pour connaître le domaine d'application, les services que le système devrait fournir, la performance du système requise, les contraintes matérielles, d'autres systèmes, etc.

✧Les étapes comprennent:

- Découverte des exigences,
- Classification et organisation des exigences,
- Priorisation et négociation des exigences,
- Spécification des exigences.

Problèmes de l'extraction et de l'analyse des exigences

✧Les intervenants ne savent pas ce qu'ils veulent vraiment.

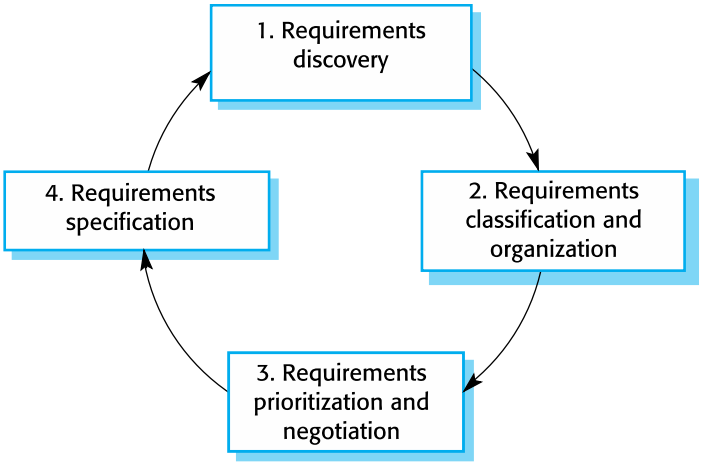
✧Les parties prenantes expriment des exigences dans leurs propres termes.

✧Les différentes parties prenantes peuvent avoir des exigences contradictoires.

✧Les facteurs organisationnels et politiques peuvent influer sur les exigences du système.

✧Les exigences changent pendant le processus d'analyse. De nouveaux parties prenantes peuvent émerger et l'environnement de métier peut être changé.

Le processus d'elicitation et d'analyse des exigences



Les activités du processus

✧Découverte des exigences
- Interagir avec les parties prenantes pour découvrir leurs besoins. Les exigences du domaine sont également découvertes à ce stade.

✧Classification et organisation des exigences
- Grouper et organiser les exigences en groupes cohérents.

✧Prioritisation et négociation
- Donner la priorité à des exigences et résoudre les conflits des exigences.

✧Spécification des exigences
- Les exigences sont documentées et entrées dans le prochain tour de la spirale.

Découverte des exigences

✧Le processus de collecte d'informations sur les systèmes requis et existants et la distillation des exigences de l'utilisateur et du système à partir de ces informations.

✧Interaction avec les parties prenantes du système de gestionnaires aux régulateurs externes.

✧Systèmes ont normalement un éventail de parties prenantes.

Interview

✧Interviews formels ou informels avec les parties prenantes font partie de la plupart des processus de RE.

✧Types d'interview
- Interview fermé basé sur la liste prédéterminée de questions

- Interview ouvert où diverses questions sont explorées avec les parties prenantes.

✧L'interview efficace

- Soyez ouvert d'esprit, évitez les idées préconçues sur les exigences et soyez prêt à écouter les parties prenantes.
- Invitez l'interviewé à lancer des discussions en utilisant une question de tremplin, une proposition d'exigences ou en travaillant ensemble sur un système prototype.

L'interview dans la pratique

✧Normalement, un mélange d'interview fermé et ouvert.

✧Les interviews sont bons pour obtenir une compréhension globale de ce que les parties prenantes font et comment ils peuvent interagir avec le système.

✧Les intervieweurs doivent être ouverts d'esprit sans idées préconçues sur ce que le système devrait faire

✧Vous devez encourager l'utilisation pour parler du système en suggérant des exigences plutôt que de simplement leur demander ce qu'ils veulent.

Problèmes avec les interviews

✧Les spécialistes de l'application peuvent utiliser un langage pour décrire leur travail qui n'est pas facile à comprendre pour l'ingénieur des exigences.

✧Les interviews ne sont pas bonnes pour comprendre les exigences de domaine

- Les ingénieurs d'exigences ne peuvent pas comprendre la terminologie d'un domaine spécifique;
- Certaines connaissances du domaine sont si familières que les gens trouvent difficile d'expliquer ou pensent que cela ne vaut pas la peine de s'exprimer.

Ethnographie

✧Un scientifique social passe un temps considérable en observant et en analysant comment les gens fonctionnent réellement.

✧Les gens n'ont pas pour expliquer ou prononcer leur travail.

✧Les facteurs sociaux et organisationnels d'importance peuvent être observées.

✧Les études ethnographiques ont montré que le travail est généralement plus riche et plus complexe que d'être suggéré par des simples modèles de systèmes.

Portée de l'ethnographie

✧Exigences qui sont dérivées de la façon de travail des gens plutôt que la façon suggérée par un processus.

✧Exigences qui sont dérivées de la coopération et de la sensibilisation aux activités d'autres personnes.

- Prise de conscience de ce que font les autres conduit à des changements dans la façon dont

nous faisons les choses.

✧Ethnographie est efficace pour comprendre les processus existants, mais ne peut pas identifier les nouvelles fonctionnalités qui devraient être ajoutés à un système.

Stories et scénarios

✧Les scénarios et les stories d'utilisateurs sont des exemples concrets (real-life examples) de la manière dont un système peut être utilisé.

✧Les stories et les scénarios sont une description de la façon dont un système peut être utilisé pour une tâche particulière.

✧Parce qu'ils sont basés sur une situation pratique, les parties prenantes peuvent être rattachées à eux et peuvent commenter leur situation par rapport à l'histoire (story).

Partage de photos en classe (iLearn)

✧Jack is a primary school teacher in Ullapool (a village in northern Scotland). He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development and economic impact of fishing. As part of this, pupils are asked to gather and share reminiscences from relatives, use newspaper archives and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history resources site) to access newspaper archives and photographs. However, Jack also needs a photo sharing site as he wants pupils to take and comment on each others' photos and to upload scans of old photographs that they may have in their families.

Jack sends an email to a primary school teachers group, which he is a member of to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he uses KidsTakePics, a photo sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account. He uses the iLearn setup service to add KidsTakePics to the services seen by the pupils in his class so that when they log in, they can immediately use the system to upload photos from their mobile devices and class computers.

Scénarios

✧Une forme structurée d'histoire (story) d'utilisateur

✧Ils devraient comprendre

- Une description de la situation de départ;
- Une description du flux normal des événements;
- Une description de ce qui peut mal tourner;
- Informations sur d'autres activités concurrentes
- Une description de l'état lorsque le scénario se termine.

Téléchargement de photos (iLearn)

✧Initial assumption: A user or a group of users have one or more digital photographs to be uploaded to the picture sharing site. These are saved on either a tablet or laptop computer. They have successfully logged on to KidsTakePics.

✧Normal: The user chooses upload photos and they are prompted to select the photos to be uploaded on their computer and to select the project name under which the photos will be stored. They should also be given the option of inputting keywords that should be associated with each uploaded photo. Uploaded photos are named by creating a conjunction of the user name with the filename of the photo on the local computer.

✧On completion of the upload, the system automatically sends an email to the project moderator asking them to check new content and generates an on-screen message to the user that this has been done.

Téléchargement de photos (iLearn)

✧What can go wrong:

✧No moderator is associated with the selected project. An email is automatically generated to the school administrator asking them to nominate a project moderator. Users should be informed that there could be a delay in making their photos visible.

✧Photos with the same name have already been uploaded by the same user. The user should be asked if they wish to re-upload the photos with the same name, rename the photos or cancel the upload. If they chose to re-upload the photos, the originals are overwritten. If they chose to rename the photos, a new name is automatically generated by adding a number to the existing file name.

✧Other activities: The moderator may be logged on to the system and may approve photos as they are uploaded.

✧System state on completion: User is logged on. The selected photos have been uploaded and assigned a status 'awaiting moderation'. Photos are visible to the moderator and to the user who uploaded them.

5. Spécification des exigences

Spécification des exigences

✧Le processus d'écriture des exigences de l'utilisateur et du système dans un document d'exigences.

✧Les exigences des utilisateurs doivent être compréhensibles par les utilisateurs finaux et les clients qui n'ont pas de formation technique.

✧Les exigences du système sont des exigences plus détaillées et peuvent inclure des informations plus techniques.

✧Les exigences peuvent faire partie d'un contrat pour le développement du système

- Il est donc important que ceux-ci soient aussi complets que possible.

Façons d'écriture d'une spécification des exigences du système

Notation	Description
Langage naturel (Natural language)	Les exigences sont écrites en utilisant des phrases numérotées en langage naturel. Chaque phrase doit exprimer une exigence.
Langage naturel structuré (Structured natural language)	Les exigences sont écrits en langage naturel sur un formulaire standard ou un modèle. Chaque champ fournit des informations sur un aspect de l'exigence.
Langages de description de conception (Design description languages)	Cette approche utilise un langage comme un langage de programmation, mais avec plus de fonctionnalités abstraites pour préciser les exigences en définissant un modèle opérationnel du système. Cette approche est maintenant rarement utilisé mais il peut être utile pour les spécifications de l'interface.
Notations graphiques (Graphical notations)	Les modèles graphiques, complétés par des annotations de texte, sont utilisés pour définir les exigences fonctionnelles pour le système; cas d'utilisation d'UML, des diagrammes de séquence sont couramment utilisés.
Spécifications mathématiques (Mathematical specifications)	Ces notations sont basées sur des concepts mathématiques tels que les machines à états finis ou des ensembles. Bien que ces spécifications non ambigués peuvent réduire l'ambiguïté dans un document d'exigences, la plupart des clients ne comprends pas une spécification formelle. Ils ne peuvent pas vérifier qu'elle représente ce qu'ils veulent et sont réticents à l'accepter comme un contrat de système

Exigences et conception

✧En principe, les exigences devraient indiquer ce que le système devrait faire et la conception devrait décrire comment il le fait.

✧En pratique, les exigences et la conception sont indissociables

- Une architecture de système peut être conçue pour structurer les exigences;
- Le système peut interagir avec d'autres systèmes qui génèrent des exigences de conception;
- L'utilisation d'une architecture spécifique pour satisfaire des exigences non fonctionnelles peut être une exigence de domaine.
- Cela peut être la conséquence d'une exigence réglementaire.

Langage naturel de spécification

✧Les exigences sont écrites comme des phrases en langage naturel complétées par des diagrammes et des tableaux.

✧Utilisé pour l'écriture des exigences parce que c'est expressive, intuitif et universel. Cela signifie que les exigences peuvent être comprises par les utilisateurs et clients.

Lignes directrices (guidelines) pour l'écriture des exigences

✧Inventer un format standard et l'utiliser pour toutes les exigences.

✧Utiliser un langage d'une manière cohérente. Utilisation nécessaire pour les exigences obligatoires, souhaitable pour les exigences desirables.

✧Utilisez le texte en surbrillance pour identifier des éléments clés de l'exigence.

✧Éviter l'utilisation du jargon informatique.

✧Inclure une explication (logique) des raisons pour lesquelles une exigence est nécessaire.

Problèmes avec le langage naturel

✧Manque de clarté:
- La précision est difficile, ce qui rend le document difficile à lire.

✧Confusion des exigences
- Les exigences fonctionnelles et non-fonctionnelles ont tendance à être mélangées.

✧Amalgamation (fusion) des exigences
- Plusieurs différentes exigences peuvent être exprimées ensemble.

Exemples des exigences pour le système logiciel de pompe à insuline

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. (Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. (A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)

Spécifications structurées

✧Approche d'écriture des exigences où la liberté du rédacteur des exigences est limitée et les exigences sont écrites de manière standard.

✧Cela fonctionne bien pour certains types d'exigences, par ex. exigences pour le système de contrôle embarqué, mais elle est parfois trop rigide pour l'écriture des exigences du système métier (entreprise).

Spécifications basées sur un formulaire (Form-based)

✧Définition de la fonction ou de l'entité.

✧Description des entrées et d'où elles viennent.

✧Description des sorties et où elles vont.

✧Informations sur les informations nécessaires au calcul et aux autres entités utilisées.

✧Description de l'action à entreprendre

✧Pré et post conditions (si approprié).

✧Les effets secondaires (s'il y en a) de la fonction.

Une spécification structurée d'une exigence pour une pompe à insuline

Insulin Pump/Control Software/SRS/3.3.2

Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Spécification tabulaire

✧Utilisé pour compléter le langage naturel.

✧Particulièrement utile lorsque vous devez définir un certain nombre des alternatives d'action.

✧Par exemple, les systèmes de pompe à insuline fait ses calculs sur le taux de changement du niveau de sucre dans le sang et la spécification tabulaire explique comment calculer les besoins en insuline pour différents scénarios.

Spécification tabulaire de calcul pour une pompe à insuline

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0

Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 – r1) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

Cas d'utilisation

✧Les cas d'utilisation sont une sorte de scénario inclus dans le langage UML.

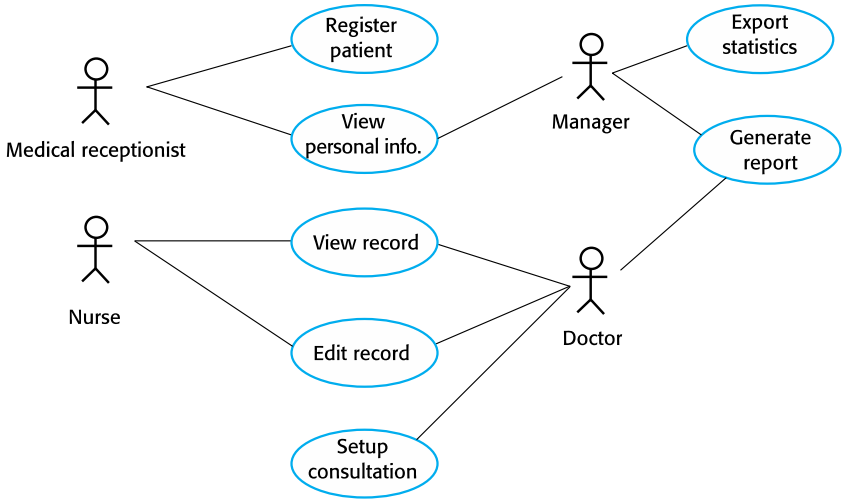
✧Les cas d'utilisation identifient les acteurs d'une interaction et décrivent l'interaction elle-même.

✧Un ensemble de cas d'utilisation doit décrire toutes les interactions possibles avec le système.

✧Modèle graphique de haut niveau complété par une description tabulaire plus détaillée (voir chapitre 5).

✧Les diagrammes de séquence UML peuvent être utilisés pour ajouter des détails aux cas d'utilisation en montrant la séquence du traitement des événements dans le système.

Les cas d'utilisation pour le système Mentcare



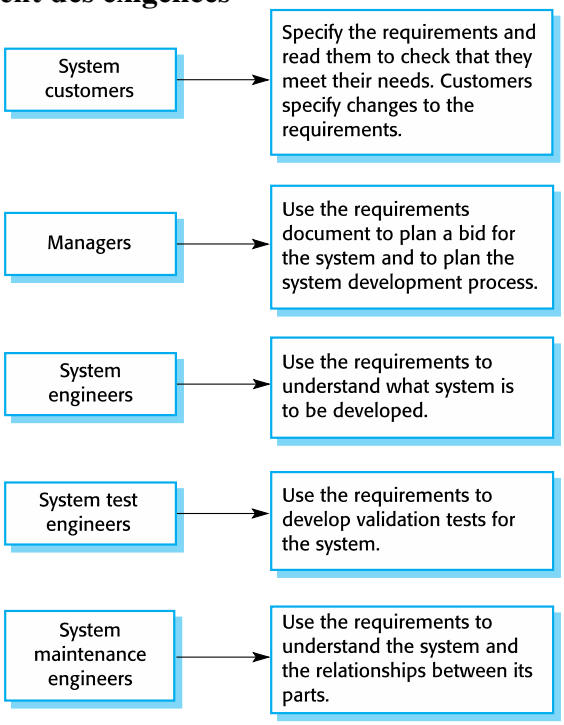
Le document des exigences du logiciel

✧Le document des exigences du logiciel (Software Requirements Spécifications - SRS) est la déclaration officielle de ce qui est exigé des développeurs de systèmes.

✧Devrait inclure à la fois une définition des exigences des utilisateurs et une spécification des exigences du système.

✧Il n'est pas un document de conception. Autant que possible, il devrait préciser ce que le système doit faire (QUOI) plutôt que comment il doit le faire (COMMENT).

Les utilisateurs d'un document des exigences



La variabilité dans le document des exigences

✧Les informations dans le document des exigences dépendent de type du système et de l'approche de développement utilisée.

✧Les systèmes développés par l'incrémentation seront, généralement, moins détaillés dans le document des exigences.

✧Les normes relatives aux documents d'exigences ont été conçues par ex. Norme IEEE (ex. IEEE/ANSI 830-1998). Ceux-ci sont principalement applicables aux exigences pour les grands projets d'ingénierie des systèmes.

La structure d'un document des exigences

Chapter	Description
Préface (Preface)	Cela devrait définir le lectorat prévu du document et décrire son histoire de version, y compris une justification pour la création d'une nouvelle version et un résumé des modifications apportées dans chaque version.
Introduction (Introduction)	Cela devrait décrire le besoin du système. Il devrait décrire brièvement les fonctions du système et expliquer comment il fonctionnera avec d'autres systèmes. Il doit également décrire comment le système s'intègre dans l'ensemble des objectifs commerciaux ou stratégiques de l'organisation qui commande le logiciel.

Glossaire (Glossary)	Cela devrait définir les termes techniques utilisés dans le document. Vous ne devriez pas faire des hypothèses sur l'expérience ou l'expertise du lecteur.
Définition des exigences de l'utilisateur (User requirements definition)	Ici, vous décrivez les services fournis pour l'utilisateur. Les exigences non fonctionnelles du système devraient également être décrites dans cette section. Cette description peut utiliser un langage naturel, diagrammes, ou d'autres notations qui sont compréhensibles pour les clients. Les normes de produits et de processus qui doivent être suivies doivent être précisés.
Architecture du système (System architecture)	Ce chapitre devrait présenter une vue d'ensemble de haut niveau de l'architecture du système prévue, montrant la répartition des fonctions entre les modules du système. Les composants architecturaux réutilisés doivent être mis en évidence.
spécification des exigences système (System requirements specification)	Cela devrait décrire les exigences fonctionnelles et non fonctionnelles plus en détail. Si nécessaire, d'autres détails peuvent également être ajoutés aux exigences non fonctionnelles. Les interfaces avec d'autres systèmes peuvent être définies.
Modèles du système (System models)	Cela peut inclure des modèles de système graphique montrant les relations entre les composants du système et le système et son environnement. Des exemples de modèles possibles sont les modèles d'objets, les modèles de flux de données ou les modèles de données sémantiques..
l'évolution du système (System evolution)	Cela devrait décrire les hypothèses fondamentales sur lesquelles repose le système, ainsi que les changements anticipés dus à l'évolution du matériel, à l'évolution des besoins des utilisateurs, etc. Cette section est utile pour les concepteurs de systèmes car elle peut les aider à éviter les décisions de conception qui limiteraient les changements futurs probables du système.
Annexes (Appendices)	Ceux-ci devraient fournir des informations détaillées et spécifiques liées à l'application en cours d'élaboration; par exemple, les descriptions de matériel et de base de données. Les exigences matérielles définissent les configurations minimales et optimales pour le système. Les exigences de base de données définissent l'organisation logique des données utilisées par le système et les relations entre les données.
Index (Index)	Plusieurs index du document peuvent être inclus. En plus d'un index alphabétique normal, il peut y avoir un index de diagrammes, un index de fonctions, et ainsi de suite.

6. Validation des exigences

✧Concernée par démontrer que les exigences définissent le système que le client veut vraiment.

✧Coûts de l'erreur sur les exigences sont élevées et donc la validation est très importante

- Résolution d'une erreur des exigences après la livraison peut coûter jusqu'à 100 fois le coût de la fixation d'une erreur de mise en œuvre.

Vérification des exigences

✧**Validité**. Est-ce que le système fournit les fonctions qui supportent bien les exigences du client?

✧**Cohérence**. Y at-il des conflits d'exigence?

✧**Complétude**. Sont toutes les fonctions demandées par le client incluses?

✧**Réalisme**. Peuvent les exigences à implémenter offrir un budget disponible et technologie

✧**Vérifiabilité**. Peuvent les exigences être vérifiés?

Techniques de validation des exigences

✧Revue des Exigences
- Analyse manuelle systématique des exigences.

✧Prototypage
- Utilisation d'un modèle exécutable du système pour vérifier les exigences.

✧Génération de cas de test
- Le développement de tests pour les exigences afin de vérifier la testabilité.

Revues (revisions) des exigences

✧Des examens réguliers devraient avoir lieu pendant la formulation de la définition des exigences.

✧Les staff du client et du contracteur devrait participer aux examens.

✧Les examens peuvent être formels (avec des documents complétés) ou informels. Une bonne communication entre les développeurs, les clients et les utilisateurs peut résoudre les problèmes à un stade précoce.

Les vérifications de la révisions

✧Vérifiabilité
- Est-ce que l'exigence est concretement vérifiable?

✧Compréhensibilité
- Est-ce que l'exigence est bien comprise?

✧Traçabilité
- Est-ce que l'origine de l'exigence est clairement indiqué?

✧Adaptabilité
- Peut l'exigence être modifié sans un impact important sur d'autres exigences?

7. Changement des exigences

Les environnements métier et technique du système changent toujours après l'installation.
- Nouveau matériel peut être introduit, il peut être nécessaire d'interfacer le système avec

d'autres systèmes, les priorités de métier peuvent changer (avec changements conséquents dans le système de support requis), et une nouvelle législation et les règlements peuvent être introduites que le système doit nécessairement respecter.

Les gens qui payent pour un système et les utilisateurs de ce système sont rarement les mêmes personnes.

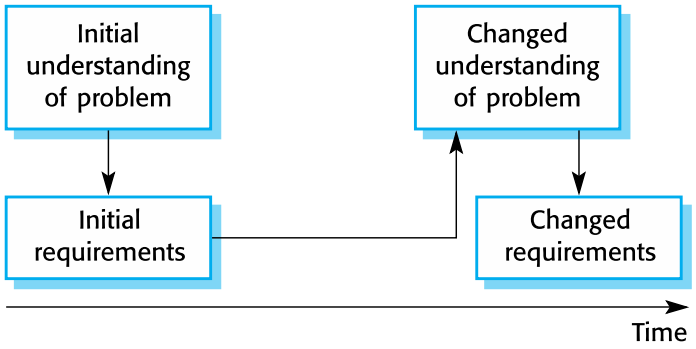
✧Les clients du système imposent des exigences en raison de contraintes organisationnelles et budgétaires. Ceux-ci peuvent entrer en conflit avec les exigences de l'utilisateur final et, après la délivrance, de nouvelles fonctionnalités peuvent être ajoutées pour répondre à l'utilisateur si le système veut atteindre ses objectifs.

Changement des exigences

✧Les grands systèmes ont généralement une communauté d'utilisateurs diversifiée, de nombreux utilisateurs ayant des exigences et des priorités différentes qui peuvent être conflictuelles ou contradictoires.

- Les exigences finales du système sont inévitablement un compromis entre eux et, avec l'expérience, on découvre souvent que l'équilibre du soutien apporté aux différents utilisateurs doit être modifié.

L'évolution des exigences



Gestion des exigences

✧La gestion des exigences est le processus de gestion des changement des exigences en cours pendant le processus d'ingénierie des exigences et le développement du système.

✧De nouvelles exigences apparaissent lors du développement du système et après sa mise en service

✧Vous devez garder une trace des besoins individuels et maintenir des liens entre les besoins dépendants de sorte que vous pouvez évaluer l'impact des changements des exigences. Vous devez établir un processus formel pour faire des propositions de changement et les relier aux exigences du système.

La planification de la gestion des exigences

✧Établit le niveau de détail de gestion des exigences qui est nécessaire.

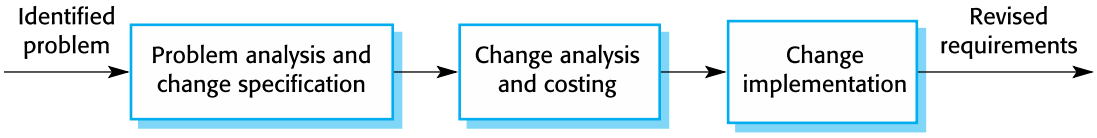
✧Les décisions de gestion des exigences:

- L'identification des exigences : Chaque exigence doit être identifié de manière unique afin qu'il puisse être recoupées avec d'autres exigences.
- Un processus de gestion du changement : C'est l'ensemble des activités qui permettent d'évaluer l'impact et le coût des changements.
- Politiques de traçabilité : Ces politiques définissent les relations entre les exigences, et entre les exigences et la conception du système qui doit être enregistrée.
- Support d'outil : Les outils qui peuvent être utilisés varient des systèmes de gestion des exigences spécialisées à des feuilles de calcul et des systèmes de base de données simples.

La gestion de changement des exigences

✧Décider si un changement des exigences devrait être acceptée

- Analyse du problème et la spécification du changement
  - Lors de cette étape, le problème ou la proposition de changement est analysé pour vérifier qu'il est valide. Cette analyse est renvoyée au demandeur du changement qui peut répondre avec une proposition des exigences de changement plus spécifiques, ou décider de retirer la demande.
- L'analyse du changement et les coûts
  - L'effet de la modification proposée est évaluée à l'aide des informations de traçabilité et la connaissance générale des exigences du système. Une fois cette analyse terminée, une décision est prise ou non de procéder avec les changements des exigences.
- Changer la mise en œuvre
  - Le document sur les exigences et, si nécessaire, la conception du système et la mise en œuvre, sont modifiés. Idéalement, le document doit être organisée de telle sorte que des changements peuvent être facilement mises en œuvre.



8. Points clés

- ✓ Les exigences pour un système logiciel énoncent ce que le système doit faire et définir des contraintes sur son fonctionnement et sa mise en œuvre.
- ✓ Les exigences fonctionnelles sont des déclarations des services que le système doit fournir ou des descriptions de la façon dont certains calculs doivent être effectués.
- ✓ Les exigences non fonctionnelles limitent souvent le système en cours de développement et le processus de développement utilisé.
- ✓ Ils sont souvent liés à des propriétés émergentes du système et donc s'appliquent au système dans son ensemble.
- ✓ Points clés
- ✓ Le processus d'ingénierie des exigences est un processus itératif y compris les exigences extraction, analyse, spécification et validation.
- ✓ L'élicitation et l'analyse des exigences est un processus itératif qui peut être représenté comme une

spirale d'activités – découverte des exigences, classification et l'organisation des exigences, négociation et documentation des exigences.

- ✓ Vous pouvez utiliser une gamme de techniques pour l'élicitation des exigences, y compris des interviews et l'ethnographie. Des stories d'utilisateurs et des scénarios peuvent être utilisés pour faciliter les discussions.
- ✓ La spécification des exigences est le processus de documentation formelle des besoins de l'utilisateur et du système et la création d'un document sur les exigences logicielles.
- ✓ Le document sur les exigences de logiciels est un exposé conjoint des exigences du système. Elle doit être organisée de telle sorte que les clients du système et les développeurs de logiciels peuvent utiliser.
- ✓ La validation des exigences est le processus de vérifier les conditions de validité, la cohérence, l'exhaustivité, le réalisme et la vérifiabilité.
- ✓ Les changements commerciaux, techniques et organisationnelles conduisent inévitablement à des changements aux exigences pour un système logiciel. La gestion des exigences est le processus de gestion et de contrôle de ces changements.

9. Exercices

I) Quiz :

Cocher la bonne réponse :

1. L'ingénierie des exigences est un processus générique qui ne varie pas d'un projet logiciel à l'autre.

   a. Vrai
   b. Faux

2. Trois choses qui rendent l'élicitation des exigences difficile sont les problèmes de :

   a. budgétisation
   b. portée du système
   c. compréhension
   d. volatilité
   e. b, c et d

3. Une partie prenante est une personne qui achètera le système logiciel réalisé en cours de développement.

   a. Vrai
   b. Faux

4. Il est relativement commun pour différents clients de proposer des exigences contradictoires, chacun faisant valoir que sa version est la bonne.

   a. Vrai
   b. Faux

II) Questions de recherche:

1. Découvrez les ambiguïtés ou les omissions dans l'énoncé des besoins pour partie d'un système d'émission de tickets ou billets (*ticket-issuing system*) qui suit:

« Un système de tickets automatisé vend des billets de train. Les utilisateurs sélectionnent leur destination en introduisant une carte de crédit et un numéro d'identification personnel. Le billet de train est émis et leur compte de carte de crédit est débité. Lorsque l'utilisateur appuie sur le bouton « **Start** », un écran de menu des destinations potentielles est activé, avec un message à l'utilisateur de sélectionner une destination. Une fois la destination est sélectionnée, les utilisateurs sont invités à entrer leur carte de crédit. Sa validité est vérifiée et l'utilisateur est alors invité à entrer un identifiant personnel. Lorsque la transaction de crédit est validée, le ticket est émis. »

2. Ecrire un ensemble d'exigences non-fonctionnelles pour le système d'émission de tickets, indiquant sa fiabilité et son temps de réponse prévus.
3. En utilisant vos connaissances sur la façon de l'utilisation des Guichet Automatique Bancaire ou GAB (en anglais, ATM : Automated Teller Machine) :
   - Identifier les principaux cas d'utilisation qui pourrait servir comme une base pour comprendre les exigences d'un système de GAB
   - Elaborer le cas d'utilisation « Retirer de l'argent » selon la forme structurée suivante :

Cas d'utilisation : .....

Acteurs: .....

Entrées: .....

Sorties: .....

Fonctionnement normal: .....

.....

.....

.....

Exception:

.....

.....

III) Modélisation des exigences avec UML : Diagramme de cas d'utilisation

1) Répondre par Vrai ou Faux aux énoncés suivants:
   a. Les développeurs et les clients créent des cas d'utilisation pour aider l'équipe du logiciel à comprendre comment les différentes catégories d'utilisateurs finaux utiliseront les fonctions.
   b. Les acteurs du cas d'utilisation sont toujours des personnes, jamais des appareils système.
   c. Un acteur est obligatoirement une personne physique.
   d. Tous les cas d'utilisation ont une relation de communication directe avec un acteur.
   e. Deux cas d'utilisation peuvent être reliés par une relation de généralisation/spécialisation.
   f. Deux acteurs peuvent être reliés par une relation de généralisation/spécialisation.
2) Quelle est la différence entre les relations d'extension (relation stéréotypée « *extend* ») et d'inclusion (relation stéréotypée « *include* ») dans un diagramme de cas d'utilisation?

- Donnez un exemple de chaque type de relation.
- Qu'est-ce qu'un «point d'extension» (anglais : extension point) et une condition d'extension?

3) Un système simplifié de GAB offre les services suivants :

a) Distribution d'argent à tout porteur d'une carte (p.ex., carte de crédit ou carte bancaire), via un lecteur de cartes et un distributeur de billets.
b) Consultation du solde de compte, dépôt en liquide et dépôt de chèques pour les clients de la banque porteurs d'une carte bancaire.

N'oubliez pas non plus que :
c) Toutes les transactions sont sécurisées.
d) Il est parfois nécessaire de recharger le distributeur, etc.

À partir de ces quatre phrases, essayer progressivement :

- Identifier les acteurs,
- Identifier les cas d'utilisation,
- Organiser et structurer les cas d'utilisation en déterminant les relations entre eux afin de construire un diagramme de cas d'utilisation le plus complet possible.

10.Solutions

I) Quiz :

1:a 2:e 3:b 4:a

II) Questions de recherche:

1) Plusieurs ambiguïtés et omissions, à savoir:

a) Un client peut acheter plusieurs billets pour la même destination ensemble ou doit il acheter un à la fois?

b) Les clients peuvent annuler une demande si une erreur a été commise?

c) Comment le système devrait répondre si une carte invalide est entrée?

d) Qu'est-ce qui se passe si les clients essaient de mettre leur carte avant de choisir une destination (comme ils le feraient dans les machines DAB (Distributeurs Automatiques de Billets))?

e) Doit l'utilisateur presser le bouton de démarrage à nouveau s'il veut acheter un autre billet vers une autre destination?

f) Le système devrait seulement permettre la vente des billets entre la station où la machine est située et des destinations directes ou doit-il inclure toutes les destinations possibles?

2) Exigences non fonctionnelles possibles pour système de délivrance de billets comprennent:

a) Entre 06:00 et 23:00 dans une même journée, l'ensemble des temps d'arrêt du système ne doit pas dépasser 5 minutes.

b) Entre 06:00 et 23:00 dans une même journée, le temps de reprise après un échec du système ne doit pas dépasser 2 minutes.

c) Entre 23:00 et 06:00 dans une même journée, l'ensemble des temps d'arrêt du système devrait ne dépasser pas 20 minutes.

Tout cela sont des exigences de disponibilité - Notez que celles-ci varient en fonction du temps de jour. Les échecs lorsque la plupart des gens circulent sont moins acceptables que les échecs quand il y a peu de clients.

d) Après que le client appuie sur un bouton de la machine, l'écran devrait être mis à jour en moins de 0,5 secondes.

e) Le temps d'émission de billet après la validation de la carte de crédit ne devrait pas dépasser 10 secondes.

f) Lors de la validation des cartes de crédit, l'affichage devrait fournir un message d'état pour les clients indiquant que l'activité est en cours.

(Cela indique au client que le temps potentiellement de l'activité de la validation est toujours en cours et que le système n'a pas tout simplement échoué).

g) Le taux d'échec maximum acceptable pour les demandes d'émission de billets est de 1/10,000 (1 echec dans 10.000 demandes) (ROCOF : Rate Of oCcurrence Of Failures ou bien Taux d'occurrence de pannes).

Notez que c'est vraiment ROCOF. Je n'ai pas spécifié le nombre acceptable de tickets incorrects car cela dépend si le système inclut ou non des fonctions de trace permettant aux demandes des clients d'être consignées. Si c'est le cas, un taux d'échec relativement élevé est acceptable car les clients peuvent se plaindre et obtenir des remboursements. Sinon, seul un taux d'échec très faible est acceptable.

Évidemment, ces exigences sont arbitraires et il y a beaucoup d'autres réponses possibles. Vous devez simplement examiner leur crédibilité.

3) Il existe une variété de différents types de GAB si, évidemment, il n'y a pas un ensemble définitif de cas d'utilisation qui pourraient être produites. Toutefois, les cas d'utilisation couvrant les principales fonctions sont : **retirer de l'argent, l'affichage du solde, impression déclaration, modifier le code PIN et dépôt de cash.** La description de cas devrait décrire les acteurs impliqués, les entrées et sorties, le fonctionnement normal et exceptions.

- **Retirer de l'argent**:

Acteurs: Client, Système bancaire.

Entrées: Carte du client, PIN (Personal Identification Number ou numéro d'identification personnel), Détails du compte Bancaire.

Sorties: Carte du client, reçu, Détails du compte Bancaire du client.

Fonctionnement normal: le client entre sa carte dans la machine. Il/elle entre le code PIN en utilisant le clavier. S'il est correct, un menu d'options s'affiche. L'option Retirer cash est sélectionnée. Le client est invité à saisir le montant demandé. S'il y a suffisamment de fonds dans son compte, l'argent est délivré, un reçu est imprimé et le solde du compte est mis à jour. Avant la délivrance de l'argent, la carte est
retournée au client qui est incité par la machine pour prendre leur carte.

Exception:

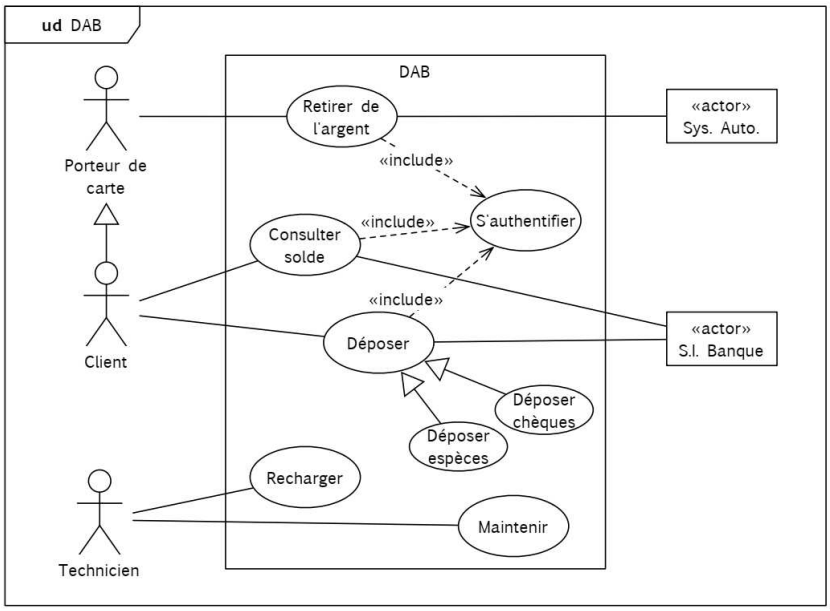
- **Carte non valide** : Carte est conservée par la machine; Client est conseillé de demander conseils.
- **PIN incorrect** : Le client est demandé à recomposer la clé PIN. S'il est incorrect après trois (3)
  tentatives, la carte est conservée par la machine et le client est invité à demander conseils.
- **Solde insuffisant-Transaction terminée**. Carte retournée au client avec un message

III) Modélisation des exigences avec UML : Diagramme de cas d'utilisation

1)
   a) Vrai
   b) Faux
   c) Faux, un acteur peut être une machine, une entreprise, un système logiciel ou autre
   d) Faux, on peut avoir des cas d'utilisation «internes» qui ne sont pas directement liés à des acteurs externes.
   e) Vrai
   f) Vrai

2)
   La relation d'extension est utilisée pour décrire un cas survenant exceptionnellement, ou de manière conditionnelle. On isole ainsi un cas optionnel du cas principal survenant systématiquement. Si l'occurrence de l'extension est conditionnelle, la condition est décrite sur la relation. La relation va du cas d'extension au cas principal. Un point d'extension est une zone dans le cas principal, dans lequel est décrite la circonstance dans laquelle l'extension est utilisée.
   La relation d'inclusion est utilisée pour décrire un cas qui est commun à plusieurs cas principaux. Ainsi, un cas d'inclusion peut (et même doit) être partagé par plusieurs cas principaux. Le cas d'inclusion décrit une tâche de bas niveau utilisée par une tâche de plus haut niveau. La relation va des cas principaux vers le cas d'inclusion.

3)
   Vous pouvez inspirer une solution assez complète à partir le diagramme ci-dessous :


ud DAB
DAB
Retirer de l'argent
«actor» Sys. Auto.
«include»
S'authentifier
Consulter solde
«include»
«include»
Déposer
«actor» S.I. Banque
Déposer chèques
Déposer espèces
Recharger
Maintenir
Porteur de carte
Client
Technicien

Chapitre V

Modélisation du Système

Objectifs

- comprendre comment les modèles graphiques peuvent être utilisés pour représenter les systèmes logiciels;
- comprendre pourquoi différents types de modèles sont requis et les perspectives fondamentales de modélisation du système de contexte, interaction, structure et comportement;
- ont été introduits dans certains types de diagrammes UML (Unified Modeling Language) et comment ces diagrammes peuvent être utilisés dans la modélisation du système;
- être conscient des idées sous-jacentes à l'ingénierie dirigée par les modèles, où un système est généré automatiquement à partir des modèles structurels et comportementaux.

Themes couverts

- Modèles de contexte
- Modèles d'interaction
- Modèles structurels
- Modèles comportementaux
- Ingénierie dirigée par les modèles

Chapitre 5:

Modélisation du Système

1. Introduction

Modélisation du Système

✧La modélisation de système est le processus de développement de modèles abstraits d'un système, chaque modèle présente une vue ou une perspective différente de ce système.

✧La modélisation du système consiste maintenant à représenter un système en utilisant une sorte de notation graphique, qui est actuellement presque toujours basée sur des notations du langage UML (Unified Modeling Language).

✧La modélisation du système aide l'analyste à comprendre la fonctionnalité du système, et les modèles sont utilisés pour communiquer avec les clients.

Modèles des systèmes existants et planifiés

✧Les modèles du système existant sont utilisés lors de l'ingénierie des exigences. Ils aident à clarifier ce que le système existant fait et peuvent être utilisés comme base pour discuter de ses forces et de ses faiblesses. Ceux-ci conduisent alors à des exigences pour le nouveau système.

✧Les modèles du nouveau système sont utilisés lors de l'ingénierie des exigences pour aider à expliquer les exigences proposées aux autres parties prenantes du système. Les ingénieurs utilisent ces modèles pour discuter des propositions de conception et documenter le système pour la l'implémentation.

✧Dans un processus d'ingénierie piloté par les modèles, il est possible de générer une implémentation de système complète ou partielle à partir du modèle de système.

Perspectives du système

✧Une perspective externe, où vous modélisez le contexte ou l'environnement du système.

✧Une perspective d'interaction, où vous modélisez les interactions entre un système et son

environnement, ou entre les composants d'un système.

✧Une perspective structurelle, où vous modélisez l'organisation d'un système ou la structure des données traitées par le système.

✧Une perspective comportementale, où vous modélisez le comportement dynamique du système et comment il réagit aux événements.

Types de diagrammes UML

✧Diagrammes d'activité, qui montrent les activités impliquées dans un processus ou dans le traitement des données.

✧Diagrammes de cas d'utilisation, qui montrent les interactions entre un système et son environnement.

✧Diagrammes de séquence, qui montrent les interactions entre les acteurs et le système et entre les composants du système.

✧Diagrammes de classes, qui montrent les classes d'objets dans le système et les associations entre ces classes.

✧Diagrammes d'état, qui montrent comment le système réagit aux événements internes et externes.

Utilisation des modèles graphiques

✧Pour faciliter la discussion sur un système existant ou proposé
- Les modèles incomplets et incorrects sont acceptables car leur rôle est de soutenir la discussion.

✧Pour documenter un système existant
- Les modèles doivent représenter fidèlement le système, mais ne doivent pas nécessairement être complets.

✧En tant que description détaillée du système pouvant être utilisée pour générer une implémentation du système
- Les modèles doivent être à la fois corrects et complets.

2. Modèles de Contexte

Modèles de Contexte

✧Les modèles de contexte sont utilisés pour illustrer le contexte opérationnel d'un système
- Ils montrent ce qui se trouve en dehors des limites du système.

✧Les préoccupations sociales et organisationnelles peuvent influencer la décision sur l'emplacement des frontières (limites) du système.

✧Les modèles architecturaux montrent le système et sa relation avec d'autres systèmes.

Frontières (limites) du système

✧Les limites du système sont établies pour définir ce qui est à l'intérieur et ce qui est à l'extérieur du système.

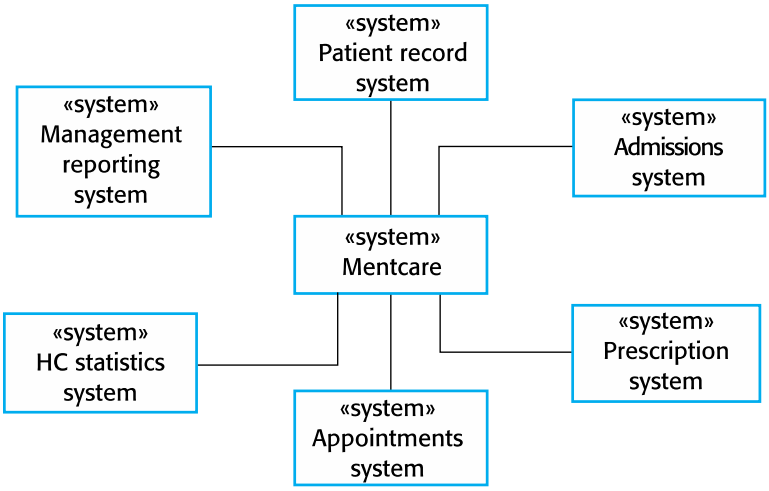
✧Ils montrent d'autres systèmes qui sont utilisés ou dépendent du système en cours de développement.

✧La position de la limite du système a un effet important sur les exigences du système.

✧Définir une limite de système est un jugement politique

✧Il peut y avoir des pressions pour développer des limites de système qui augmentent/diminuent l'influence ou la charge de travail des différentes parties d'une organisation.

Le contexte du système Mentcare



Perspective de processus

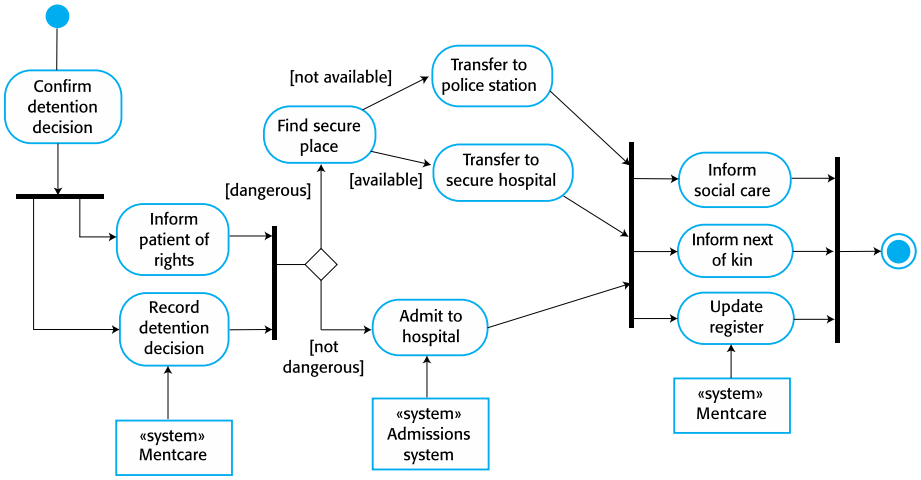
✧Les modèles de contexte montrent simplement les autres systèmes dans l'environnement, pas comment le système développé est utilisé dans cet environnement.

✧Les modèles de contexte simples sont utilisés avec d'autres modèles, tels que les modèles de processus métier. Ils décrivent des processus humains et automatisés dans lesquels des systèmes logiciels particuliers sont utilisés.

✧Les modèles de processus révèlent comment le système en cours de développement est utilisé dans des processus métier plus larges.

✧Les diagrammes d'activité UML peuvent être utilisés pour définir des modèles de processus métier.

Modèle de processus de détention involontaire



3. Modèles d'Interaction

Modèle d'Interaction

✧La modélisation de l'interaction de l'utilisateur est importante car elle permet d'identifier les besoins des utilisateurs.

✧La modélisation de l'interaction système-système met en évidence les problèmes de communication qui peuvent survenir.

✧La modélisation de l'interaction des composants nous aide à comprendre si une structure de système proposée est susceptible de fournir les performances et la fiabilité requises du système.

✧Des diagrammes de cas d'utilisation et des diagrammes de séquence peuvent être utilisés pour la modélisation d'interaction.

Modélisation de cas d'utilisation

✧Les cas d'utilisation ont été développés à l'origine pour prendre en charge l'élicitation des exigences et sont désormais intégrés dans le langage UML.

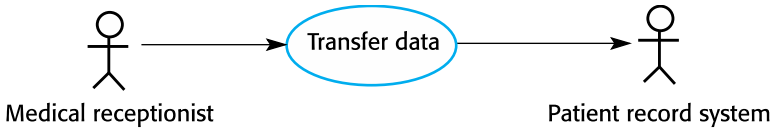
✧Chaque cas d'utilisation représente une tâche discrète impliquant une interaction externe avec un système.

✧Les acteurs dans un cas d'utilisation peuvent être des personnes ou d'autres systèmes.

✧Représenté schématiquement pour donner un aperçu du cas d'utilisation et sous une forme textuelle plus détaillée.

Cas d'utilisation « transférer des données »

✧Un cas d'utilisation dans le système Mentcare



Description tabulaire du cas d'utilisation 'Transférer des données'

Mentcare: Transfer data	
Actors	Medical receptionist, patient records system (PRS)
Description	A receptionist may transfer data from the Mentcare system to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Data	Patient's personal information, treatment summary
Stimulus	User command issued by medical receptionist
Response	Confirmation that PRS has been updated
Comments	The receptionist must have appropriate security permissions to access the patient information and the PRS.

Cas d'utilisation dans le système Mentcare impliquant le rôle de «Medical Receptionist»

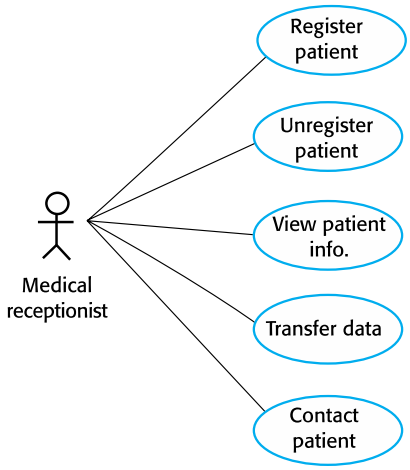


Diagramme de Séquence

✧Les diagrammes de séquence font partie de l'UML et sont utilisés pour modéliser les interactions entre les acteurs et les objets dans un système.

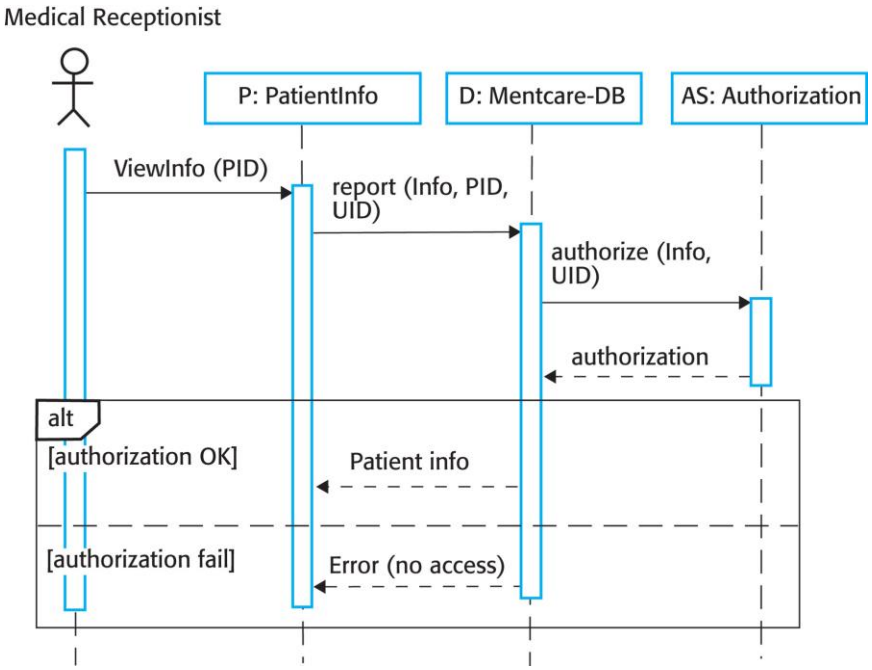
✧Un diagramme de séquence montre la séquence des interactions qui ont lieu au cours d'un cas

d'utilisation ou d'une instance de cas d'utilisation particulier.

✧Les objets et acteurs impliqués sont listés en haut du diagramme, avec une ligne pointillée tirée verticalement à partir de ceux-ci.

✧Les interactions entre les objets sont indiquées par des flèches annotées.

Diagramme de séquence pour Afficher les informations du patient «View patient information»

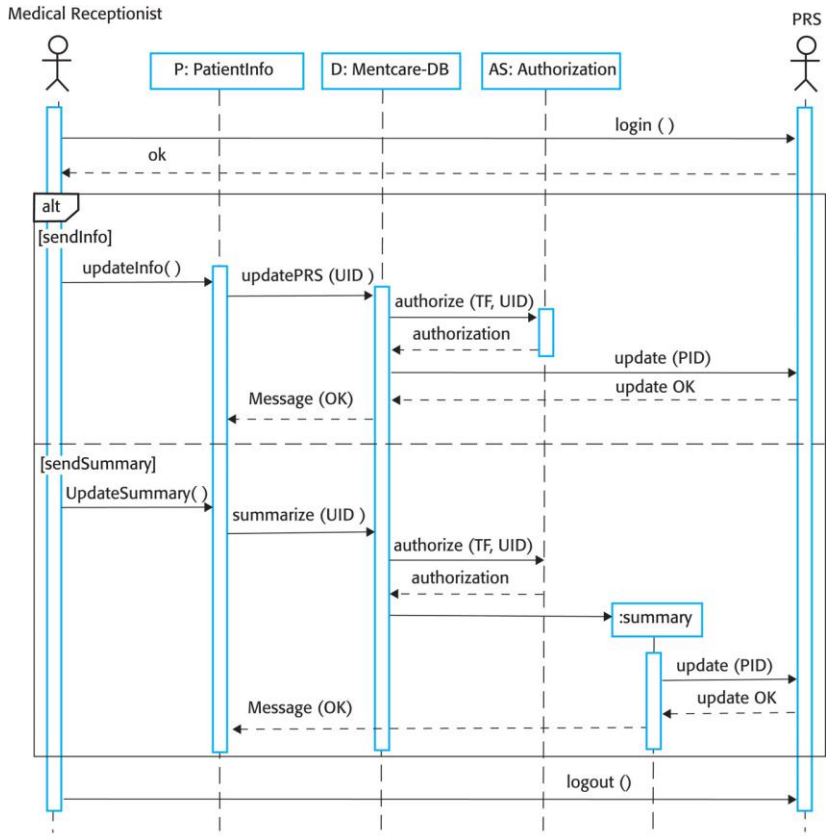


✧Ce diagramme modélise les interactions impliquées dans le cas d'utilisation «View patient information», où une réceptionniste médicale peut voir certaines informations sur le patient.

Lecture du Diagramme de séquence «View patient information»

1. La réceptionniste médicale déclenche la méthode ViewInfo dans une instance P de la classe d'objets PatientInfo, en fournissant l'identifiant du patient, PID. P est un objet d'interface utilisateur, qui est affiché comme un formulaire d'informations sur le patient.
2. L'instance P appelle la base de données pour retourner l'information requise, fournissant l'identifiant de la réceptionniste pour permettre la vérification de sécurité (à ce stade, nous ne nous soucions pas d'où vient UID).
3. La base de données vérifie avec un système d'autorisation que l'utilisateur est autorisé pour cette action.
4. Si autorisé, les informations du patient sont renvoyées et un formulaire sur l'écran de l'utilisateur est rempli. Si l'autorisation échoue, un message d'erreur est renvoyé.

Diagramme de séquence pour transférer les données «Transfer Data»



✧Deux caractéristiques supplémentaires:

- la communication directe entre les acteurs du système
- la création d'objets dans le cadre d'une séquence d'opérations.

Lecture du Diagramme de séquence «Transfer Data»

1. La réceptionniste se connecte au PRS.
2. Il y a deux options disponibles. Ceux-ci permettent le transfert direct des informations actualisées (mises à jour) de patient au PRS et le transfert de résumé (summary) des données de santé du Mentcare-DB vers le PRS.
3. Dans chaque cas, les autorisations de la réceptionniste sont vérifiées en utilisant le système d'autorisation.
4. Les informations personnelles peuvent être transférées directement de l'objet d'interface utilisateur vers le PRS. Alternativement, un enregistrement de résumé (summary) peut être créé à partir de la base de données et cet enregistrement est ensuite transféré.
5. À la fin du transfert, le PRS émet un message d'état et l'utilisateur se déconnecte.

4. Modèles Structurels

✧Les modèles structurels de logiciel affichent l'organisation d'un système en fonction des composants qui composent ce système et de leurs relations.

✧Les modèles structurels peuvent être des modèles statiques, qui montrent la structure de la conception du système, ou des modèles dynamiques, qui montrent l'organisation du système lors de son exécution.

✧L'organisation dynamique d'un système sous la forme d'un ensemble de threads en interaction peut être très différente d'un modèle statique des composants du système.

✧Vous créez des modèles structurels d'un système lorsque vous discutez et concevez l'architecture du système.

Diagramme de Classes

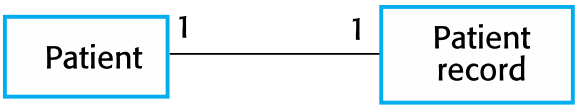
✧Les diagrammes de classes sont utilisés lors du développement d'un modèle de système orienté objet pour montrer les classes dans un système et les associations entre ces classes.

✧Une classe d'objets peut être considérée comme une définition générale d'un type d'objet système.

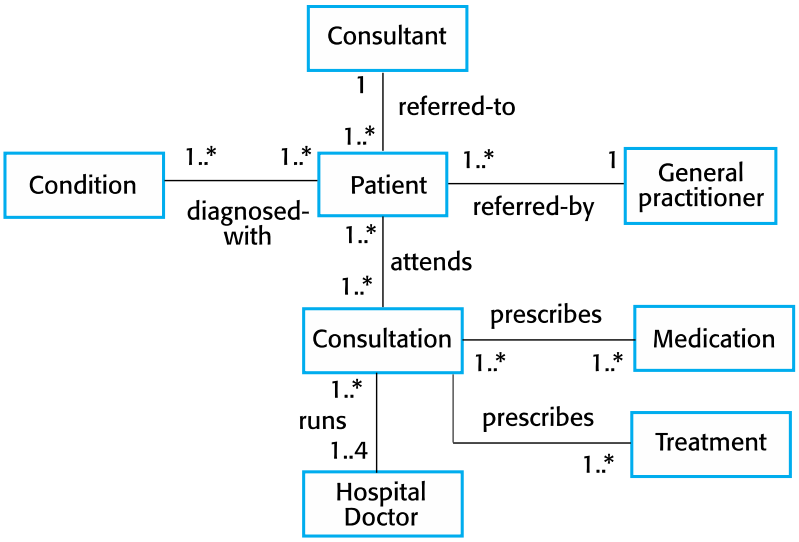
✧Une association est un lien entre les classes qui indique qu'il existe une relation entre ces classes.

✧Lorsque vous développez des modèles au cours des premières étapes du processus de génie logiciel, les objets représentent quelque chose dans le monde réel, comme un patient, une prescription, un médecin, etc.

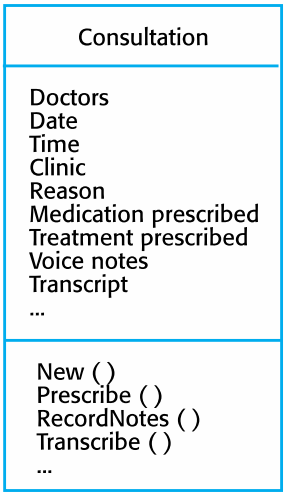
Classes UML et association



Classes et associations dans le système Mentcare



La classe « Consultation »



✧On suppose que les médecins enregistrent des notes vocales qui sont transcrites plus tard pour enregistrer les détails de la consultation.

✧Pour prescrire un médicament, le médecin doit utiliser l'opération Prescrire pour générer une ordonnance électronique.

Généralisation

✧La généralisation est une technique quotidienne que nous utilisons pour gérer la complexité.

✧Plutôt que d'apprendre les caractéristiques détaillées de chaque entité que nous expérimentons, nous plaçons ces entités dans des classes plus générales (animaux, voitures, maisons, etc.) et apprenons les caractéristiques de ces classes.

✧Cela nous permet d'inférer que différents membres de ces classes ont des caractéristiques communes, par ex. les écureuils et les rats sont des rongeurs.

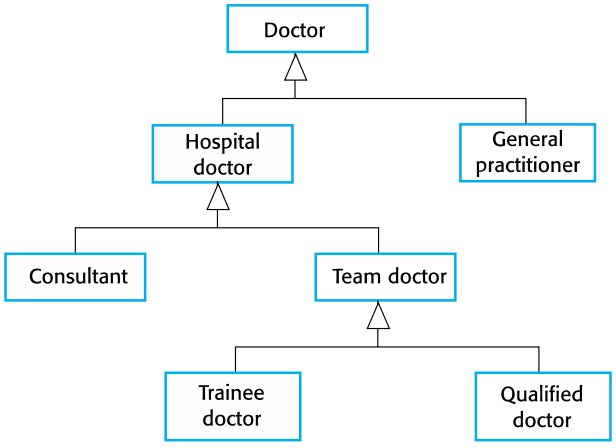
✧Dans les systèmes de modélisation, il est souvent utile d'examiner les classes d'un système pour voir s'il existe une possibilité de généralisation. Si des modifications sont proposées, vous n'avez pas besoin de regarder toutes les classes du système pour voir si elles sont affectées par le changement.

✧Dans les langages orientés objet, tels que Java, la généralisation est implémentée en utilisant les mécanismes d'héritage de classe intégrés dans le langage.

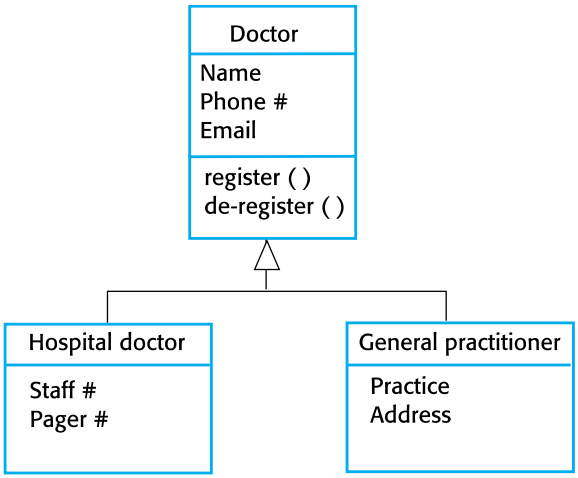
✧Dans une généralisation, les attributs et les opérations associés aux classes de niveau supérieur sont également associés aux classes de niveau inférieur.

✧ Les classes de niveau inférieur sont des sous-classes qui héritent des attributs et des opérations de leurs superclasses. Ces classes de niveau inférieur ajoutent ensuite des attributs et des opérations plus spécifiques.

Une hiérarchie de généralisation



Une hiérarchie de généralisation avec plus de détails

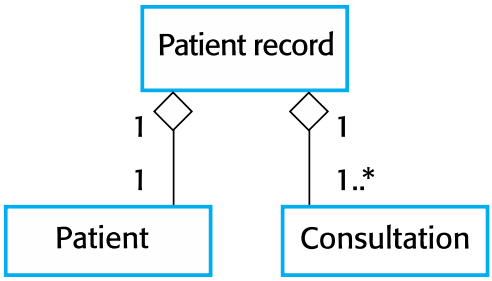


Modèles d'agrégation de classes d'objets

✧Un modèle d'agrégation montre comment les classes qui sont des collections sont composées d'autres classes.

✧Les modèles d'agrégation sont similaires à la relation « partie de (part-of) » dans les modèles de données sémantiques.

L' association d'aggregation



5. Modèles comportementaux

✧Les modèles comportementaux sont des modèles du comportement dynamique d'un système en cours d'exécution. Ils montrent ce qui se passe ou ce qui est supposé se produire lorsqu'un système répond à un stimulus de son environnement.

✧Vous pouvez penser à ces stimuli comme étant de deux types:

- **Données**: Certaines données arrivent qui doivent être traitées par le système.
- **Evénements**: Certains événements déclenchent le traitement du système. Les événements peuvent contenir des données associées, bien que ce ne soit pas toujours le cas.

Modélisation pilotée par les données « Data-driven modeling »

✧De nombreux systèmes d'entreprise (Business systems) sont des systèmes de traitement de données qui sont principalement pilotés par des données. Ils sont contrôlés par l'entrée de données dans le

système, avec relativement peu de traitement des événements externes.

✧Par exemple: un système de facturation téléphonique acceptera des informations sur les appels passés par un client, calculera les coûts de ces appels et générera une facture à envoyer à ce client.

✧Les modèles pilotés par les données montrent la séquence des actions impliquées dans le traitement des données d'entrée et la génération d'une sortie associée.

✧Ils sont particulièrement utiles lors de l'analyse des exigences car ils peuvent être utilisés pour montrer le traitement de bout en bout dans un système.

Modélisation pilotée par les données « Data-driven modeling »

✧Plusieurs diagrammes sont utilisés pour la modélisation de traitement des données:

- Diagramme de Flux de Données (Data Flow Diagram – DFD) (DeMarco, 1978)
- Diagramme d'activité
- Diagramme de Séquence
- Le langage UML ne prend pas en charge les DFD tels qu'ils ont été initialement proposés et utilisés pour la modélisation du traitement des données. La raison est que les DFD se concentrent sur les fonctions du système et ne reconnaissent pas les objets système.
- Cependant, UML 2.0 UML a introduit les diagrammes d'activité similaires aux DFD.

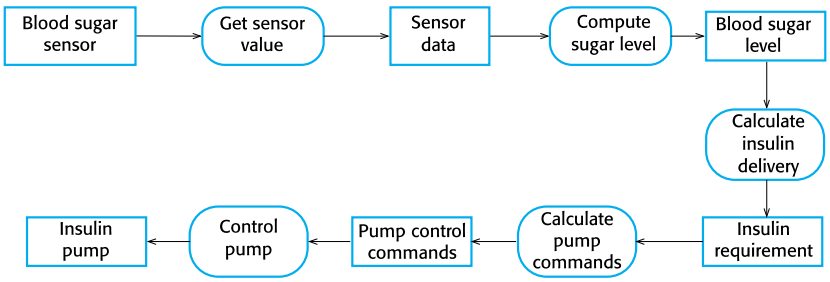
Modélisation pilotée par les données « Data-driven modeling »

✧Un autre moyen de montrer la séquence de traitement dans un système consiste à utiliser des diagrammes de séquence UML.

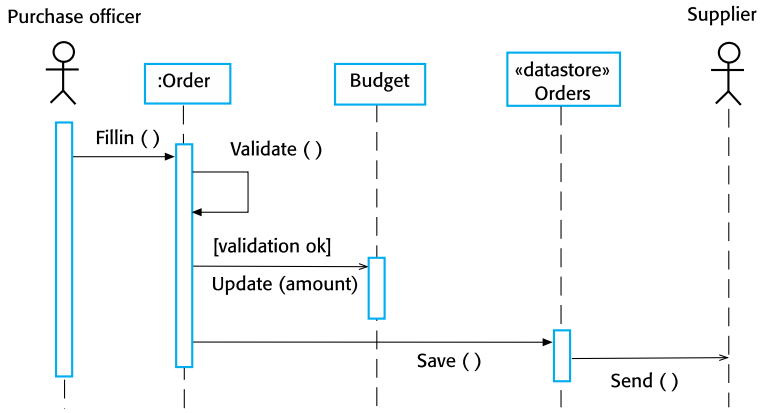
✧Vous avez vu comment ils peuvent être utilisés pour modéliser l'interaction mais, si vous les dessinez de sorte que les messages ne soient envoyés que de gauche à droite, ils montrent le traitement séquentiel des données dans le système.

✧Les modèles de séquence mettent en évidence des objets dans un système, tandis que les diagrammes de flux de données (DFD) mettent en évidence les fonctions.

Un modèle d'activité de fonctionnement de la pompe à insuline



Un diagramme de séquence pour le traitement de commandes « Order processing »



Modélisation pilotée par les événements « Event-driven modeling »

✧Les systèmes en temps réel[2] sont souvent pilotés par les événements, avec un minimum de traitement des données.

✧Par exemple, un système de commutation de téléphone fixe répond à des événements tels que «combiné décroché» en générant une tonalité.

✧La modélisation pilotée par les événements montre comment un système répond aux événements externes et internes.

✧Elle est basée sur l'hypothèse qu'un système a un nombre fini d'états et que les événements (stimuli) peuvent provoquer une transition d'un état à un autre.

Modèles de machines à états (state machine)

✧Ceux-ci modélisent le comportement du système en réponse à des événements externes et internes.

✧Ils montrent que les réponses du système aux stimuli sont souvent utilisées pour modéliser les systèmes en temps réel.

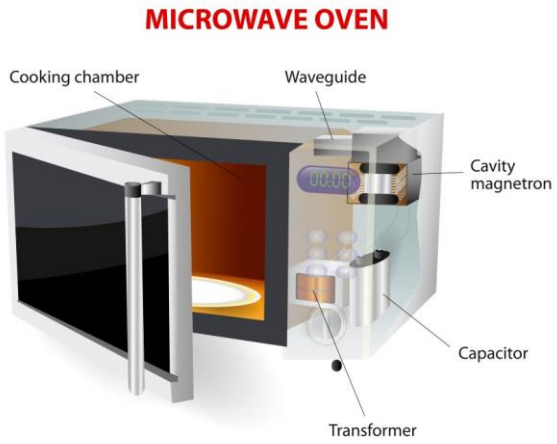
✧Les modèles de machine à états montrent les états du système comme des nœuds et les événements comme des arcs entre ces nœuds. Lorsqu'un événement se produit, le système passe d'un état à un autre.

✧« Statecharts »[3] sont une partie intégrante de l'UML et sont utilisés pour représenter les modèles de machines d'état.

[2] Un **système temps réel** est un système capable de contrôler (ou piloter) un procédé physique à une vitesse adaptée à l'évolution du procédé contrôlé. le système en temps réel ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés (soumis à des contraintes temporelles, pas forcément rapide).

[3] Appelé aussi: Diagramme états-transitions ou diagramme d'états

Exemple: Four à micro-ondes (Microwave oven)



Description de Fonctionnement du "Microwave oven"

✧Nous utilisons un exemple de logiciel de contrôle pour un four à micro-ondes très simple pour illustrer la modélisation événementielle.

✧Les vrais fours à micro-ondes sont en réalité beaucoup plus complexes que ce système, mais le système simplifié est plus facile à comprendre.

✧Ce micro-ondes simple a un commutateur pour sélectionner la pleine ou la moitié de puissance, un clavier numérique pour entrer le temps de cuisson, un bouton marche/arrêt, et un affichage alphanumérique.

✧On suppose que la séquence d'actions dans l'utilisation du micro-ondes est:

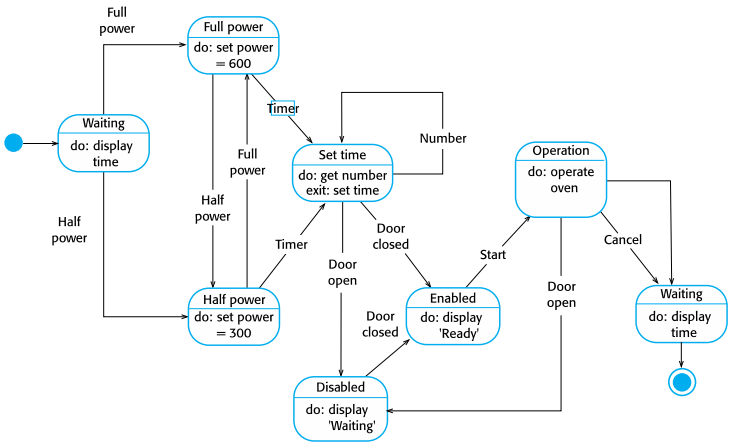
1. Sélectionnez le niveau de puissance (demi-puissance ou pleine puissance).
2. Entrez le temps de cuisson en utilisant un clavier numérique.
3. Appuyez sur Démarrer et les aliments sont cuits pour l'instant donné.

✧Pour des raisons de sécurité (safety), le four ne doit pas fonctionner lorsque la porte est ouverte et, à la fin de la cuisson, un signal sonore retentit (produit du beaucoup de bruit). Le four a un affichage alphanumérique très simple qui est utilisé pour afficher diverses alertes et messages d'avertissement.

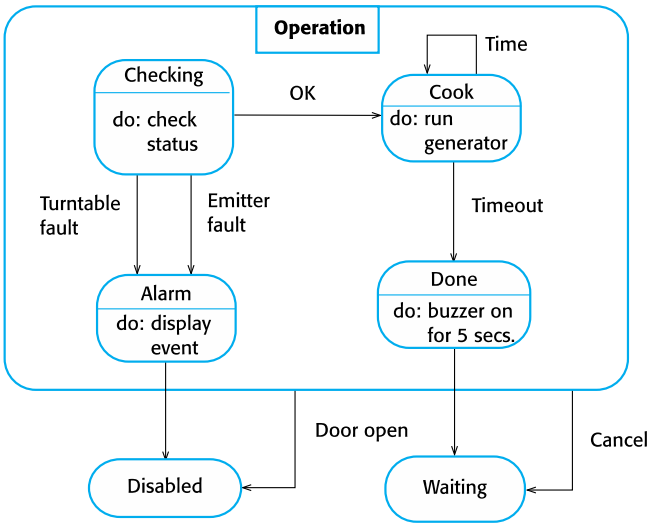
✧Dans les diagrammes d'état UML, les rectangles arrondis représentent les états du système. Ils peuvent inclure une brève description (après «faire ou Do») des actions entreprises dans cet état.

✧Les flèches étiquetées représentent des stimuli qui forcent une transition d'un état à un autre. Vous pouvez indiquer les états de début et de fin en utilisant des cercles pleins, comme dans les diagrammes d'activité.

Diagramme d'état d'un four à micro-ondes



Operation du four à micro-ondes « Microwave oven operation »



✧Le problème avec la modélisation basée sur l'état est que le nombre d'états possibles augmente rapidement. Pour les grands modèles de système, vous devez donc masquer les détails dans le modèles par la notion de « superstate »

✧Operation est un super état « Superstate » qui peut être étendu en plusieurs etats séparés « substates »

Etats et stimuli pour le four à micro-ondes

Etat (State)	Description
Waiting	Le four attend l'entrée. L'affichage indique l'heure actuelle.
Half power	La puissance du four est réglée sur 300 watts. L'écran affiche 'Half power'.
Full power	La puissance du four est réglée sur 600 watts. L'affichage indique 'Full

	power'.
Set time	Le temps de cuisson est réglé sur la valeur d'entrée de l'utilisateur. L'affichage indique le temps de cuisson sélectionné et est mis à jour à mesure que temps est réglée.
Disabled	L'opération du four est désactivée pour la sécurité. La lumière du four intérieur est allumée. L'affichage indique 'Not ready'.
Enabled	L'opération du four est activée. L'éclairage du four intérieur est éteint. L'écran affiche 'Ready to cook'.
Operation	Four en fonctionnement. La lumière du four intérieur est allumée. L'affichage montre le compte à rebours de la minuterie. À la fin de la cuisson, le buzzer (avertisseur sonore) retentit pendant cinq secondes. La lumière du four est allumée. L'écran affiche 'Cooking complete' tandis que le buzzer sonne.

Etats et stimuli pour le four à micro-ondes (b)

Stimulus	Description
Half power	L'utilisateur a appuyé sur le bouton «half-power» .
Full power	L'utilisateur a appuyé sur le bouton «full-power».
Timer	L'utilisateur a appuyé sur les boutons de Timer
Number	L'utilisateur a appuyé sur le clavier numérique
Door open	L'interrupteur de la porte du four n'est pas fermé.
Door closed	L'interrupteur de la porte du four est fermé.
Start	L'utilisateur a appuyé sur le bouton «Start».
Cancel	L'utilisateur a appuyé sur le bouton «Cancel» .

Activité[4] :

1) Donner un exemple d'un system d'entreprise (de gestion, en anglais: Business System) et modéliser son comportement par une modélisation pilotée par les données.

2) Donner un exemple d'un system temps réel (en anglais: Real-time system) et modéliser son comportement par une modélisation pilotée par les événements.

[4] Remarque: pour les deux modélisations utiliser un outil UML tel que (Astah : http://astah.net/editions/community).

6. Ingénierie Dirigée par les Modèles[5]

✧L'ingénierie dirigée par les modèles (IDM) est une approche du développement de logiciels où les modèles plutôt que les programmes sont les principaux résultats du processus de développement.

✧Les programmes qui s'exécutent sur une plate-forme matérielle/logicielle sont ensuite générés automatiquement à partir des modèles.

✧Les partisans de MDE soutiennent que cela augmente le niveau d'abstraction dans l'ingénierie logicielle, de sorte que les ingénieurs n'ont plus à se soucier des détails du langage de programmation ou des spécificités des plates-formes d'exécution.

Utilisation de l'IDM

✧L'ingénierie dirigée par les modèles est encore à un stade précoce de développement, et il n'est pas clair si cela aura ou non un effet significatif sur la pratique de l'ingénierie logicielle.

✧Avantages

- Permet aux systèmes d'être considérés à des niveaux plus élevés d'abstraction
- Générer du code automatiquement signifie qu'il est moins coûteux d'adapter les systèmes à des nouvelles plateformes.

✧Les inconvénients

- Des modèles pour l'abstraction et pas nécessairement pour la mise en œuvre.
- Les économies générées par la génération de code peuvent être compensées par les coûts de développement de traducteurs (trasformation) pour de nouvelles plates-formes.

Architecture pilotée par les modèles[6]

✧L'architecture pilotée par les modèles (MDA) a été le précurseur d'une ingénierie plus générale basée sur des modèles.

✧MDA est une approche centrée sur les modèles pour la conception et la mise en œuvre de logiciels qui utilise un sous-ensemble de modèles UML pour décrire un système.

✧Des modèles à différents niveaux d'abstraction sont créés. A partir d'un modèle indépendant de haut niveau, il est possible, en principe, de générer un programme de travail sans intervention manuelle.

Types de Modèles

✧Un modèle indépendant du calcul (computation independent model - CIM)

- Ceux-ci modélisent les abstractions de domaine importantes utilisées dans un système. Les CIM sont parfois appelés modèles de domaine.

✧Un modèle indépendant de la plateforme (platform independent model - PIM)

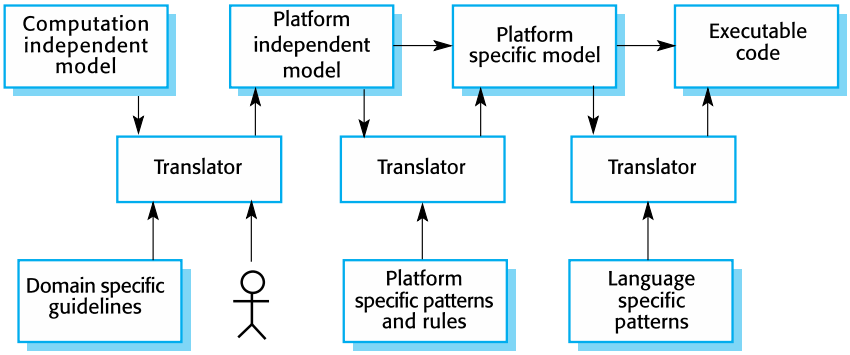
- Ceux-ci modélisent le fonctionnement du système sans référence à sa mise en œuvre. Le PIM est généralement décrit en utilisant des modèles UML qui montrent la structure statique du système et comment il réagit aux événements externes et internes.

[5] « Model-Driven Engineering - MDE»
[6] « Model driven architecture – MDA »

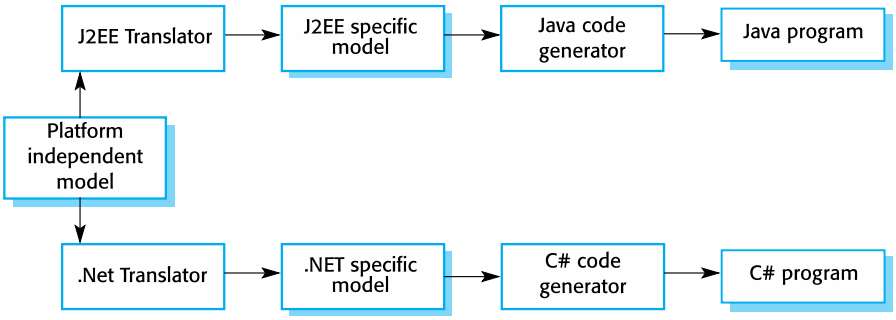
✧Modèles spécifiques à la plate-forme (Platform specific models - PSM)

- Ce sont des transformations du PIM au un PSM séparé pour chaque plate-forme d'application. En principe, il peut y avoir des couches de PSM, chaque couche ajoutant des détails spécifiques à la plate-forme.

Transformations MDA



Modèles multiples spécifiques à la plate-forme



Méthodes agiles et MDA

✧Les développeurs de MDA affirment qu'il est destiné à soutenir une approche itérative du développement et peuvent donc être utilisés dans des méthodes agiles.

✧La notion de modélisation initiale poussée contredit les idées fondamentales du manifeste agile et je soupçonne que peu de développeurs agiles se sentent à l'aise avec l'ingénierie dirigée par les modèles.

✧Si les transformations peuvent être complètement automatisées et qu'un programme complet peut être généré à partir d'un PIM, alors, en principe, le MDA pourrait être utilisé dans un processus de développement agile car aucun codage séparé ne serait requis.

Adoption de MDA

✧Une série de facteurs a limité l'adoption de MDE/MDA

✧Un support d'outil spécialisé est requis pour convertir des modèles d'un niveau à un autre.

✧La disponibilité des outils est limitée et les organisations peuvent nécessiter une adaptation et une

personnalisation de l'outil à leur environnement.

✧Pour les systèmes à longue durée de vie (long-lifetime systems) développés à l'aide de MDA, les entreprises hésitent à développer leurs propres outils ou à compter sur de petites entreprises qui pourraient faire faillite...

✧Les modèles sont un bon moyen pour faciliter les discussions sur la conception d'un logiciel. Cependant, les abstractions qui sont utiles pour les discussions peuvent ne pas être les bonnes abstractions pour la mise en œuvre.

✧Pour la plupart des systèmes complexes, la mise en œuvre n'est pas le problème majeur - l'ingénierie des exigences, la sécurité et la fiabilité, l'intégration avec les systèmes existants et les tests sont tous plus importants.

✧Les arguments pour l'indépendance de la plate-forme ne sont valables que pour les systèmes de grande taille et à longue durée de vie. Pour les produits logiciels et les systèmes d'information, les économies découlant de l'utilisation de MDA risquent d'être compensées par les coûts de son introduction et de son outillage.

✧L'adoption généralisée de méthodes agiles au cours de la même période d'évolution de MDA a détourné l'attention des approches dirigée par les modèles.

7. Points clés

- ✓ Un modèle est une vue abstraite d'un système qui ignore les détails du système. Des modèles de systèmes complémentaires peuvent être développés pour montrer le contexte, les interactions, la structure et le comportement du système.
- ✓ Les modèles de contexte montrent comment un système en cours de modélisation est positionné dans un environnement avec d'autres systèmes et processus.
- ✓ Les diagrammes de cas d'utilisation et les diagrammes de séquence sont utilisés pour décrire les interactions entre les utilisateurs et les systèmes dans le système en cours de conception. Les cas d'utilisation décrivent les interactions entre un système et des acteurs externes; diagrammes de séquence ajoutent plus d'informations à ceux-ci en montrant des interactions entre les objets du système.
- ✓ Les modèles structurels montrent l'organisation et l'architecture d'un système. Les diagrammes de classes sont utilisés pour définir la structure statique des classes dans un système et leurs associations.
- ✓ Les modèles comportementaux sont utilisés pour décrire le comportement dynamique d'un système d'exécution. Ce comportement peut être modélisé du point de vue des données traitées par le système ou des événements qui stimulent les réponses d'un système.
- ✓ Les diagrammes d'activités peuvent être utilisés pour modéliser le traitement des données, où chaque activité représente une étape du processus.
- ✓ Les diagrammes d'état sont utilisés pour modéliser le comportement d'un système en réponse à des événements internes ou externes.
- ✓ L'ingénierie dirigée par les modèles est une approche du développement de logiciels dans laquelle un système est représenté sous la forme d'un ensemble de modèles qui peuvent être automatiquement transformés en code exécutable.

8. Exercices

I) Quiz :

Choisir la bonne réponse :

1. Quel modèle dans la modélisation du système décrit le comportement dynamique d'un système?

   a) Modèle de contexte
   b) Modèle comportemental
   c) Modèle de données
   d) Modèle d'objet

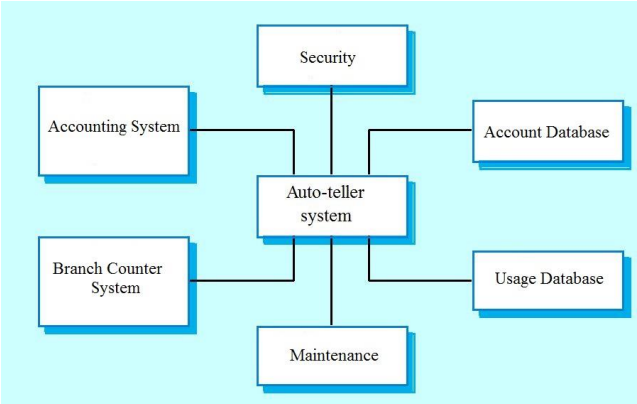
2. Quel modèle dans la modélisation du système décrit la nature statique du système?

   a) Modèle comportemental
   b) Modèle de contexte
   c) Modèle de données
   d) Modèle structurel

3. Quelle perspective dans la modélisation du système montre le système ou l'architecture de données.

   a) Perspective structurelle
   b) Perspective comportementale
   c) Perspective externe

4. Quel modèle de système est représenté par les opérations ATM présentées ci-dessous:



5. Les diagrammes d'activités sont utilisés pour modéliser le traitement des données

   a) Vrai
   b) Faux

6. L'ingénierie dirigée par les modèles n'est qu'un concept théorique. Il ne peut pas être converti en un code de travail/exécutable.

   a) Vrai
   b) Faux

7. The UML supports event-based modeling using _____ diagrams.

   a) Deployment
   b) Collaboration
   c) State chart

8. Lequel des diagrammes suivants n'est pas supporté par UML pour la modélisation pilotée par les données?

   a) Activité
   b) Diagramme de flux de données (DFD)
   c) Machine à états (Etats-Transitions)
   d) Composant

9. _____ allows us to infer that different members of classes have some common characteristics.

   a) Realization
   b) Aggregation
   c) Generalization
   d) dependency

10. On crée des modèles comportementaux d'un système lorsque nous discutons et concevons l'architecture du système.

    a) Vrai
    b) Faux

11. _____ & _____ diagrams of UML represent Interaction modeling

    a) Use Case, Sequence
    b) Class, Object
    c) Activity, State Chart

12. Quel niveau du modèle entité-association ou entité-relation (ERD : Entity Relationship Diagram) modélise toutes les entités et relations?
    a) Niveau 1
    b) Niveau 2
    c) Niveau 3

13. _____ classes are used to create the interface that the user sees and interacts with as the software is used.
    a) Controller
    b) Entity
    c) Boundary
    d) Business

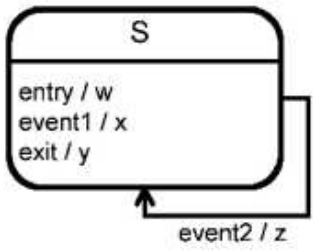
II) Questions de recherche:

1. Expliquez pourquoi il est important de modéliser le contexte d'un système en cours de développement. Donnez deux exemples d'erreurs possibles qui pourraient survenir si les ingénieurs logiciels ne comprennent pas le contexte du système.
2. Comment pourriez-vous utiliser un modèle d'un système qui existe déjà? Expliquez pourquoi il n'est pas toujours nécessaire qu'un tel modèle de système soit complet et correct. Serait-ce la même chose si vous développiez un modèle d'un nouveau système?
3. Vous avez été invité à développer un système qui aidera à planifier des événements et des fêtes de grande envergure tels que les mariages, les fêtes de fin d'études, les anniversaires, etc. En utilisant un diagramme d'activités, modélisez le contexte de processus d'un tel système. planifier une fête (réserver un lieu, organiser des invitations, etc.) et les éléments du système qui peuvent être utilisés à chaque étape.
4. Pour le système Mentcare, proposer un ensemble de cas d'utilisation qui illustre les interactions entre un médecin, qui voit les patients et prescrit des médicaments et des traitements, et le Mentcare.
5. Développer un diagramme de séquence montrant les interactions impliquées quand un étudiant s'inscrit à un cours dans une université. Les cours peuvent avoir une inscription limitée, de sorte que le processus d'inscription doit inclure des vérifications que les places sont disponibles. Supposons que l'étudiant accède à un catalogue de cours électronique pour s'informer sur les cours disponibles.
6. Regardez attentivement comment les messages et les boîtes aux lettres sont représentés dans le système de messagerie que vous utilisez. Modélisez les classes d'objets susceptibles d'être utilisées dans l'implémentation du système pour représenter une boîte aux lettres (Mailbox) et un message e-mail (Mail message).
7. Sur la base de votre expérience avec un guichet automatique bancaire (GAB), dessinez un diagramme d'activité qui modélise le traitement des données lorsqu'un client retire de l'argent de la machine.
8. Dessinez un diagramme de séquence pour le même système. Expliquez pourquoi vous pouvez développer des diagrammes d'activité et de séquence lors de la modélisation du comportement d'un système.
9. Dessiner des diagrammes d'état du logiciel de contrôle pour:

   - Une machine à laver automatique qui a différents programmes pour différents types de vêtements.
   - Le logiciel pour un lecteur de DVD.
   - Un système de répondeur téléphonique qui enregistre les messages entrants et affiche le nombre de messages acceptés sur une LED. Le système doit permettre au client du téléphone de se connecter à partir de n'importe quel endroit, de taper une séquence de numéros (identifiés comme des tonalités) et de lire les messages enregistrés.

10. Définir les effets d'entrée (entry) et de sortie (exit) pour les diagrammes d'états ?

11. Étant donné le diagramme d'états de la Figure suivante. Donnez la séquence d'activités (actions) exécutées quand on se trouve dans l'état S et :
    - event1 se produit
    - event2 se produit



12. Vous êtes un responsable de l'ingénierie logicielle et votre équipe propose que l'ingénierie dirigée par un modèle soit utilisée pour développer un nouveau système. Quels facteurs devez-vous prendre en compte lorsque vous décidez d'introduire ou non cette nouvelle approche dans le développement de logiciels?

9. Solutions

I) Quiz :

1:b (Data-Driven or Event-Driven Modeling)
2:d (structures statiques des classes pour un système et leurs associations)
3:a
4: Modèle de contexte : est utilisé pour illustrer le contexte opérationnel d'un système. Il montre ce qui se trouve en dehors des limites du système.
5:a
6: b, l'ingénierie dirigée par les modèles (IDM) est une approche du développement de logiciels dans laquelle un système est représenté sous la forme d'un ensemble de modèles qui peuvent être automatiquement transformés en code exécutable.
7: c, State diagrams show system states and events that cause transitions from one state to another.
8: b, Les DFD se concentrent sur les fonctions du système et ne reconnaissent pas les objets système.
9: c, Generalization is an everyday technique that we use to manage complexity. This means that common information will be maintained in one place only.
10: b, Les modèles structurels de logiciel exposent l'organisation d'un système en fonction des composants qui composent ce système et de leurs relations.
11: a, Use case modeling is mostly used to model interactions between a system and external actors. Sequence diagrams are used to model interactions between system components, although external agents may also be included.
12: b, Niveau 1 modélise tous les objets de données (entités) et leurs «connexions» les uns aux autres tandis que Niveau 3 modélise toutes les entités, les relations et les attributs qui fournissent une profondeur supplémentaire.
13: c

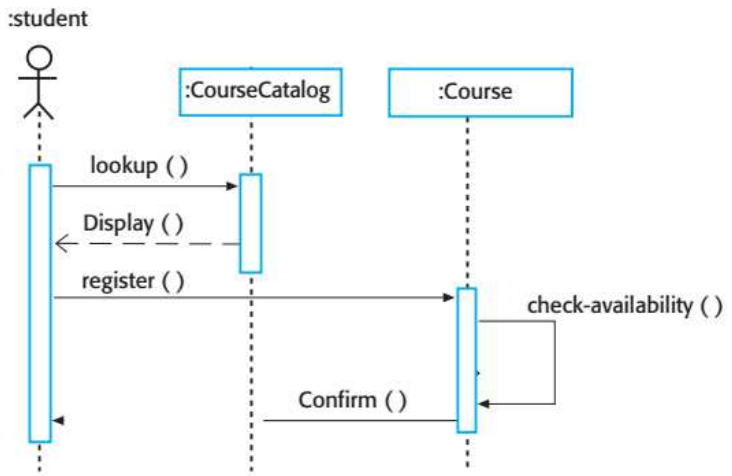
II) Questions de recherche:

2)
Vous pouvez créer et utiliser un modèle d'un système existant pour les raisons suivantes:

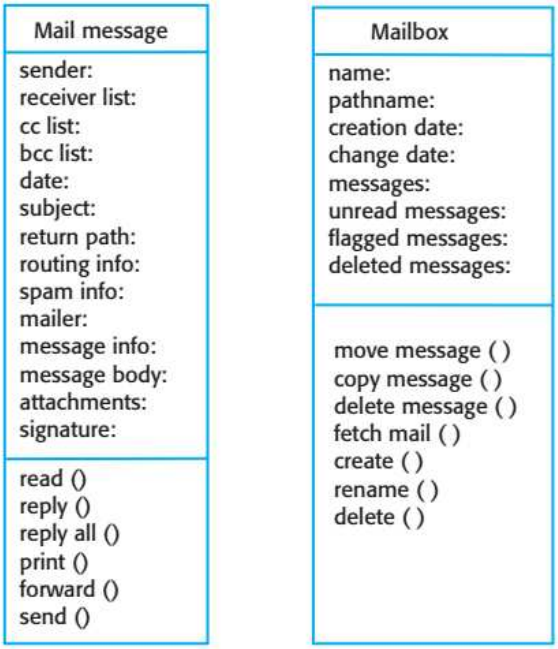
1. Comprendre et documenter l'architecture et le fonctionnement du système existant.
2. Servir de centre de discussion sur les changements possibles à ce système.
3. Informer la ré-implémentation du système.

Vous n'avez pas besoin d'un modèle complet sauf si l'intention est de documenter complètement le fonctionnement du système existant. Le but du modèle dans tels cas est généralement de vous aider à travailler sur des parties du système, de sorte que seules celles-ci doivent être modélisées.
De plus, si le modèle est utilisé comme un sujet de discussion, il est peu probable intéressé par les détails et peut donc ignorer les parties du système dans le modèle.
Cela est vrai, en général, pour les modèles de nouveaux systèmes, à moins qu'une approche basée sur les modèles de développement ait lieu dont un modèle complet est requis. Les autres circonstances dans lesquelles vous pourriez avoir besoin d'un modèle complet sont quand il y a une exigence contractuelle pour qu'un tel modèle soit produit dans le cadre de la documentation du système.
5)

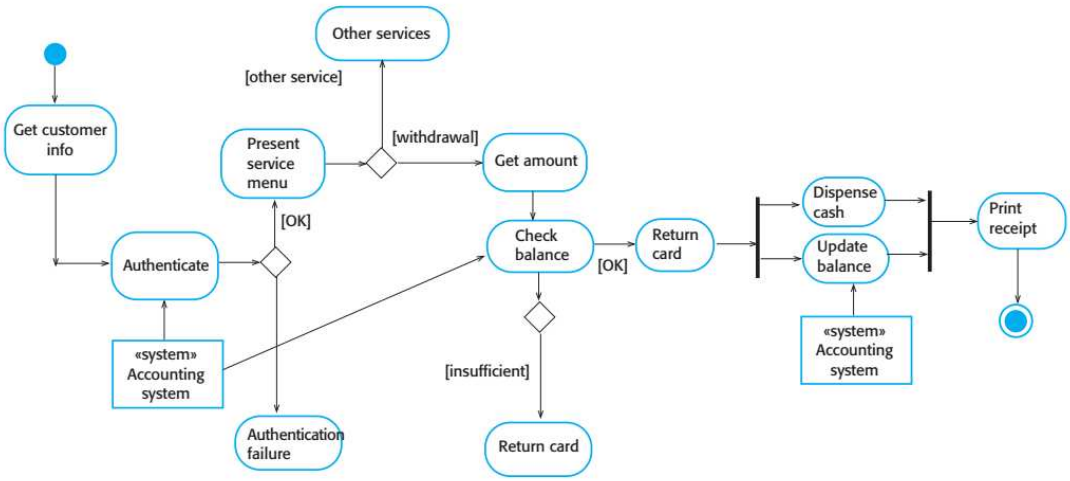
Un diagramme relativement simple est tout ce qui est nécessaire ici. Il est préférable de ne pas être trop pointilleux à propos des choses comme les styles de flèches UML que presque personne ne peut se rappeler les différences entre eux.



6)



7) Notez que la solution proposée n'a pas développé les activités représentant d'autres services ou l'authentification échouée.



10) Effets d'entrée (ou de sortie)‑Entry (Exit)

Un effet d'entrée (introduit par le mot-clé « `entry` » à l'intérieur du symbole d'un état) représente une action ou une activité qui est exécutée chaque fois que l'on entre dans cet état.

Cela permet de
factoriser un même effet qui sera déclenché par toutes les transitions qui entrent dans l'état.
L'effet de sortie (introduit par le mot-clé « `exit` ») est l'effet symétrique en sortie de l'état.

11)

1. x
2. y; z; w

12)

Les facteurs dont vous devez tenir compte au moment de prendre cette décision comprennent:

1. L'expertise de l'équipe dans l'utilisation de UML et MDA. (L'expertise est-elle déjà disponible ou une formation poussée sera-t-elle nécessaire?)
2. Les coûts et la fonctionnalité des outils disponibles pour soutenir MDA. (Les outils sont-ils disponibles à la maison ou devront-ils être achetés? Sont-ils assez bons pour le type de logiciel développé?
3. La durée de vie probable du logiciel que vous développez. (MDA est le plus approprié pour les systèmes à longue durée de vie)
4. Exigences pour la haute performance ou le débit (MDA repose sur la génération de code qui crée un code qui peut être moins efficace que le code écrit à la main)
5. Les avantages à long terme de l'utilisation de MDA (y a-t-il des économies réelles de cette approche)?
6. L'enthousiasme des développeurs de logiciels. (Tous les membres de l'équipe sont-ils engagés envers cette nouvelle approche?)

Chapitre VI

Conception Architecturale

Objectifs

- comprendre pourquoi la conception architecturale du logiciel est importante;
- comprendre les décisions qui doivent être prises au sujet de l'architecture du système pendant le processus de conception architecturale;
- ont été introduits à l'idée de modèles architecturaux, des façons bien éprouvées d'organisation des architectures de système, qui peuvent être réutilisés dans la conception de systèmes;
- connaître les modèles architecturaux souvent utilisés dans différents types de systèmes d'application, y compris les systèmes de traitement des transactions et les systèmes de traitement des langages.

Themes couverts

- Décisions de conception architecturale
- Vues architecturales
- Patrons architecturaux
- Architectures d'application

Chapitre 6:

Conception Architecturale

1. Introduction

Conception architecturale[7]

✧La conception architecturale vise à comprendre comment un système logiciel devrait être organisé et à concevoir la structure globale de ce système.

✧La conception architecturale est le lien essentiel entre la conception et l'ingénierie des exigences, car elle identifie les principaux composants structurels dans un système et les relations entre eux.

✧La sortie du processus de conception architecturale est un modèle architectural qui décrit comment le système est organisé comme un ensemble de composants communicants.

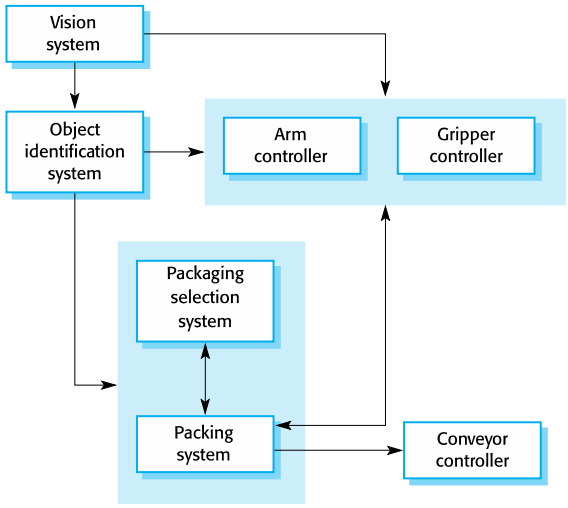
Agilité et architecture

✧Il est généralement admis qu'une première étape des processus agiles consiste à concevoir une architecture globale des systèmes.

✧Le refactoring de l'architecture du système est généralement coûteux car il affecte tant de composants dans le système

[7] Appelée aussi: conception générale, conception globale, conception préliminaire (preliminary design ou architectural design)

L'architecture d'un système de contrôle de robot d'emballage



Abstraction architecturale

✧L'architecture dans le petit est concernée par l'architecture des programmes individuels. À ce niveau, nous sommes préoccupés par la façon dont un programme individuel est décomposé en composants.

✧L'architecture dans le grand concerne l'architecture de systèmes d'entreprise complexes qui incluent d'autres systèmes, programmes et composants de programme. Ces systèmes d'entreprise sont répartis sur différents ordinateurs, qui peuvent être détenus et gérés par des sociétés différentes.

Avantages d'une architecture explicite

✧Communication avec les parties prenantes

- L'architecture peut être utilisée comme un centre de discussion par les parties prenantes du système.

✧L'analyse du système

- Signifie d'analyser s'il est possible que le système peut répondre à ses exigences non fonctionnelles.

✧Réutilisation à grande échelle

- L'architecture peut être réutilisable sur une gamme de systèmes
- Les architectures de lignes de produits (Product-line architectures) peuvent être développées.

Représentations architecturales

✧Des schémas fonctionnels simples et informels (montrant les entités et les relations) sont la méthode la plus fréquemment utilisée pour documenter les architectures logicielles.

✧Mais ceux-ci ont été critiqués parce qu'ils manquent de sémantique, ne montrent pas les types de relations entre les entités ni les propriétés visibles des entités dans l'architecture.

✧Dépend de l'utilisation de modèles architecturaux. Les exigences relatives à la sémantique du modèle

dépendent de la façon dont les modèles sont utilisés.

Diagrammes de boîtes/lignes

✧Très abstrait - ils ne montrent pas la nature des relations entre composants ni les propriétés visibles à l'extérieur des sous-systèmes.

✧Cependant, utile pour la communication avec les parties prenantes et pour la planification du projet.

Utilisation de modèles architecturaux

✧Pour faciliter la discussion sur la conception du système

- Une vue architecturale de haut niveau d'un système est utile pour la communication avec les parties prenantes du système et la planification du projet, car elle n'est pas encombrée de détails. Les parties prenantes peuvent s'y rapporter et comprendre une vision abstraite du système. Ils peuvent ensuite discuter le système dans son ensemble sans être déroutés par les détails.

✧Comme un moyen de documenter une architecture qui a été conçue

- Le but ici est de produire un modèle de système complet qui montre les différents composants d'un système, leurs interfaces et leurs connexions.

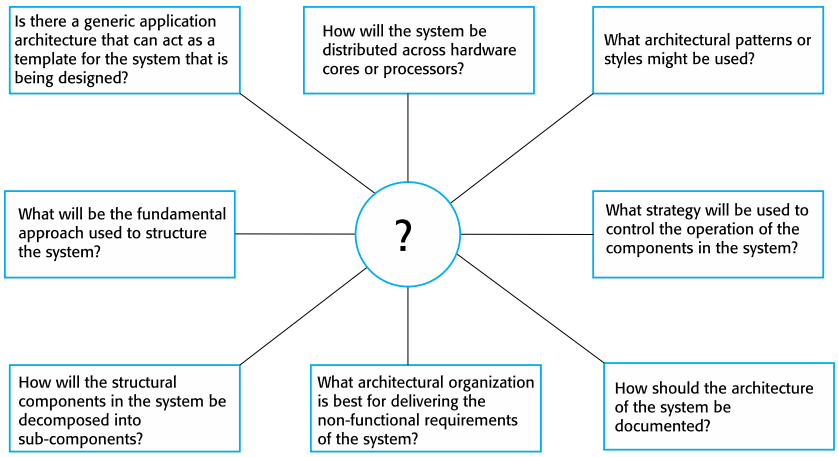
2. Décisions de Conception Architecturale

Décisions de conception architecturale

✧La conception architecturale est un processus créatif, de sorte que le processus diffère selon le type de système développé.

✧Cependant, un certain nombre de décisions communes couvrent tous les processus de conception et ces décisions affectent les caractéristiques non fonctionnelles du système.

Décisions de conception architecturale



Réutilisation de l'architecture

✧Les systèmes du même domaine ont souvent des architectures similaires qui reflètent les concepts de domaine.

✧Les applications de lignes de produits (application product lines) sont construites autour d'une architecture de base avec des variantes qui répondent aux exigences particulières des clients.

✧L'architecture d'un système peut être conçue autour d'un ou plusieurs patrons (styles) architecturaux.

- Ceux-ci capturent l'essence d'une architecture et peuvent être instanciés de différentes manières.

Le choix de style de l'architecture dépend des exigences non-fonctionnelles du système

✧**Performance**: Si elle est une exigence critique

- l'architecture doit être conçue pour localiser les opérations critiques dans un petit nombre de composants, et minimisez les communications. Utilisez des composants relativement importants plutôt que de petits composants à grain fin.

✧**Sécurité (security)**

- Utilisez une architecture en couches avec des atouts (les biens) essentiels dans les couches internes.

✧**Sureté (savety)**

- Localisez les fonctions essentielles à la sureté dans un seul composant ou un petit nombre de composants.

✧**Disponibilité**

- Inclure des composants et des mécanismes redondants pour la tolérance aux pannes.

✧**Maintenabilité**

- Utilisez des composants à grain fin remplaçables.

3. Vues architecturales

Vues architecturales

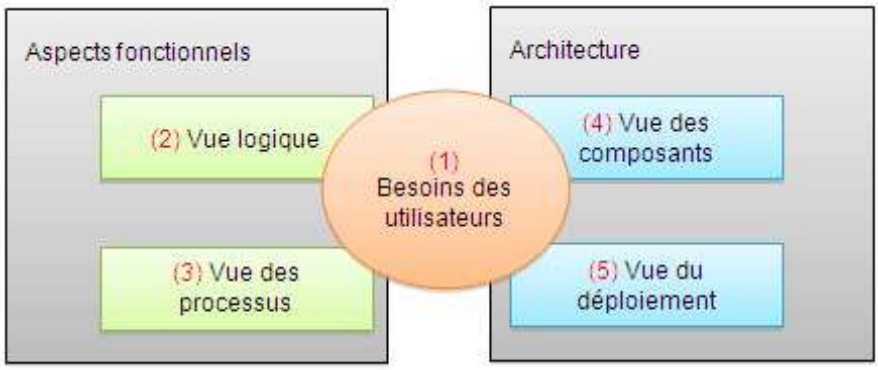
✧Quels points de vue ou perspectives sont utiles lors de la conception et de la documentation de l'architecture d'un système?

✧Quelles notations devraient être utilisées pour décrire les modèles architecturaux?

✧Chaque modèle architectural ne montre qu'une vue ou une perspective du système.

- Il peut montrer comment un système est décomposé en modules, comment les processus d'exécution interagissent ou les différentes manières dont les composants du système sont répartis sur un réseau. Pour la conception et la documentation, vous devez généralement présenter plusieurs vues de l'architecture logicielle.

Les 4 + 1 vues de l'architecture du système



✧Krutchen (1995), dans son modèle des 4 + 1 vues de l'architecture logicielle, suggère qu'il devrait exister quatre vues architecturales fondamentales, qui sont liées à l'aide de scénarios d'utilisation (modèle +1). Les points de vue qu'il suggère sont les suivants:

1. Une vue logique, qui montre les abstractions clés dans le système en tant qu'objets ou classes d'objets. Il devrait être possible de lier les exigences système aux entités de cette vue logique.
2. Une vue des processus qui montre comment, au moment de l'exécution, le système est composé des processus en interaction. Cette vue est utile pour évaluer les caractéristiques non fonctionnelles du système telles que la performance et la disponibilité.
3. Une vue de développement (composants), qui montre comment le logiciel est décomposé pour le développement, c'est-à-dire qu'il montre la décomposition du logiciel en composants mis en œuvre par un seul développeur ou une seule équipe de développement. Cette vue est utile pour les gestionnaires de logiciels et les programmeurs.
4. Une vue physique (déploiement) qui montre le matériel du système et la manière dont les composants logiciels sont répartis entre les processeurs du système. Cette vue est utile pour les ingénieurs système qui planifient un déploiement de système.

Représentation des vues architecturales

✧Certaines personnes soutiennent que le langage UML (Unified Modeling Language) est une notation appropriée pour décrire et documenter les architectures de systèmes.

✧Sommerville n'est pas d'accord avec cela car il ne pense pas que l'UML inclut des abstractions appropriées pour une description de système de haut niveau.

✧Des langages de description architecturaux (ADLs) ont été développés mais ne sont pas largement utilisés

4. Patrons architecturaux

Patrons (modèles ou styles) architecturaux

✧Les patrons[8] sont un moyen de représenter, partager et réutiliser les connaissances.

✧Un patron architectural est une description stylisée d'une bonne pratique de conception, qui a été

[8] object-oriented design patterns (Gamma et al., 1995)

expérimentée et testée dans différents environnements.

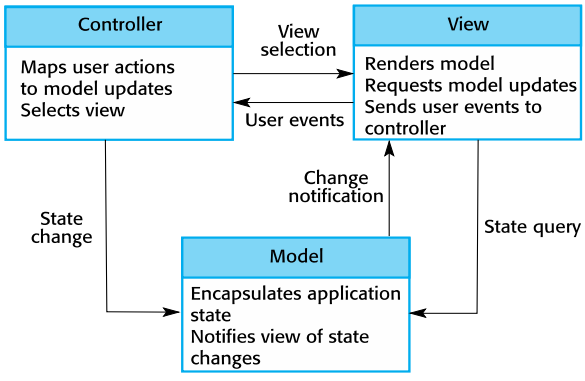
✧Les patrons devraient inclure des informations sur quand ils sont et quand ils ne sont pas utiles.

✧Les patrons peuvent être représentés en utilisant des descriptions tabulaires et graphiques.

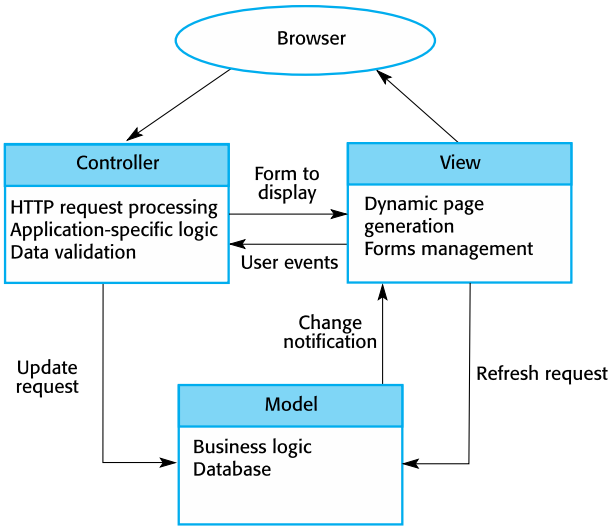
Exemple de patron : Le patron Model-View-Controller (MVC)

Nom	MVC (Model-View-Controller)
Description	Sépare la présentation et l'interaction des données du système. Le système est structuré en trois composants logiques qui interagissent les uns avec les autres. Le composant « Model » gère les données système et les opérations associées sur ces données. Le composant « View » définit et gère la manière dont les données sont présentées à l'utilisateur. Le composant « Controller » gère l'interaction de l'utilisateur (par exemple, les touches, les clics de souris, etc.) et transmet ces interactions au « View » et « Model ». Voir la Figure 6.3.
Exemple	La Figure 6.4 montre l'architecture d'un système d'application basé sur le Web (web-based application system) organisé en utilisant le patron MVC.
Quand utilisé	Utilisé lorsqu'il existe plusieurs façons de visualiser et d'interagir avec les données. Également utilisé lorsque les exigences futures pour l'interaction et la présentation des données sont inconnues.
Avantages	Permet aux données de changer indépendamment de leur représentation et vice versa. Prend en charge la présentation des mêmes données de différentes manières avec les modifications apportées dans chacune des représentations affichées.
Désavantages	Peut impliquer un code supplémentaire et une complexité de code lorsque le modèle de données et les interactions sont simples.

L'organisation du Model-View-Controller



Architecture d'application Web utilisant le modèle MVC



1) Architecture en couches « layered architecture »

✧Utilisé pour modéliser l'interfaçage des sous-systèmes.

✧Organise le système en un ensemble de couches (ou de machines abstraites) fournissant chacune un ensemble de services.

✧Prend en charge le développement incrémentiel de sous-systèmes dans différentes couches. Quand une interface de couche change, seule la couche adjacente est affectée.

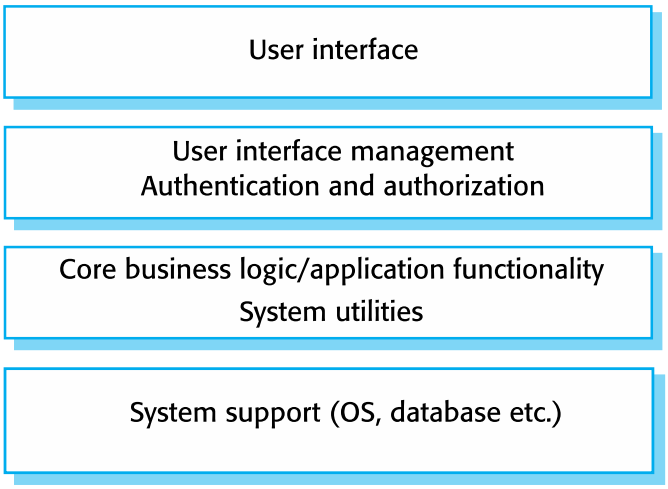
✧Pour des implémentations multiplateformes d'un système d'application, seules les couches internes dépendant de la machine doivent être ré-implémentées pour prendre en compte les fonctionnalités d'un système d'exploitation ou d'une base de données différents.

Le patron (modèle) d'architecture en couches

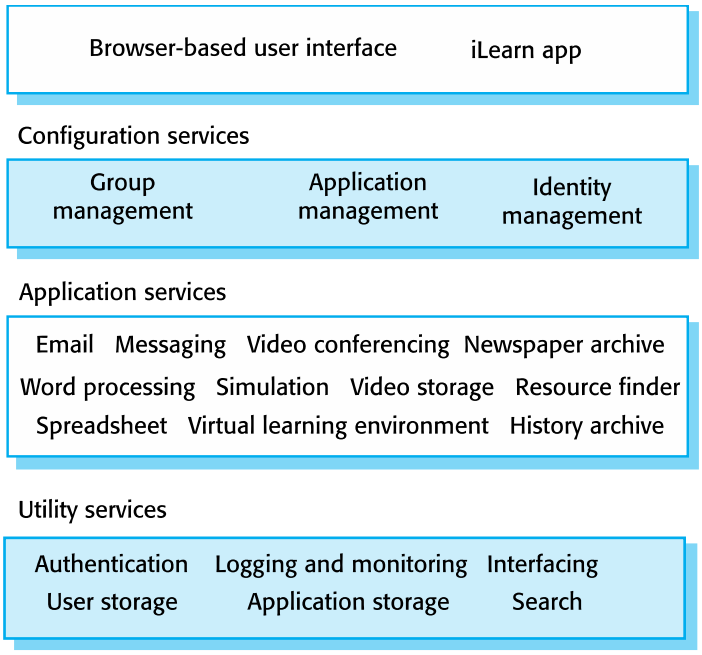
Nom	Architecture en couches
Description	Organise le système en couches avec les fonctionnalités reliées et associées à chaque couche. Une couche fournit des services à la couche supérieure afin que les couches de niveau inférieur représentent les services de base susceptibles d'être utilisés dans tout le système. Voir la Figure 6.6.
Exemple	Un modèle en couches d'un système de partage de droits d'auteur des documents tenus dans différentes bibliothèques, comme le montre la Figure 6.7.
Quand utilisé	Utilisé lors de la construction de nouvelles installations au-dessus des systèmes existants; lorsque le développement est réparti entre plusieurs équipes, chaque équipe étant responsable d'une couche de fonctionnalité; quand il y a une exigence de sécurité à plusieurs niveaux.
Avantages	Permet le remplacement de couches entières tant que l'interface est maintenue. Des installations redondantes (par exemple, l'authentification) peuvent être

	prévues dans chaque couche pour augmenter la fiabilité du système.
Désavantages	En pratique, fournir une séparation nette entre les couches est souvent difficile et une couche de haut niveau peut avoir à interagir directement avec des couches de niveau inférieur plutôt que par la couche immédiatement en dessous. Les performances peuvent être un problème en raison de plusieurs niveaux d'interprétation d'une demande de service lors du traitement de chaque couche.

Une architecture générique en couches



L'architecture du système iLearn



2) Architecture de référentiel[9] ou « Repository architecture»[10]

✧Les sous-systèmes doivent échanger des données. Cela peut être fait de deux façons:

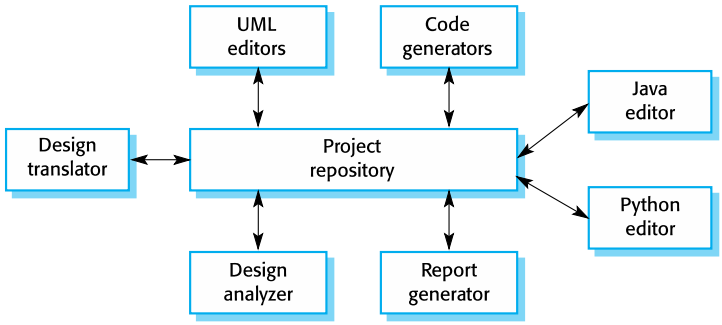
- Les données partagées sont conservées dans une base de données centrale ou un référentiel et peuvent être consultées par tous les sous-systèmes;
- Chaque sous-système maintient sa propre base de données et transmet les données explicitement à d'autres sous-systèmes.

✧Lorsque de grandes quantités de données doivent être partagées, le modèle référentiel de partage est le plus souvent utilisé car il s'agit d'un mécanisme efficace de partage de données.

Le patron du Référentiel « Repository pattern »

Nom	Référentiel "Repository"
Description	Toutes les données d'un système sont gérées dans un référentiel central accessible à tous les composants du système. Les composants n'interagissent pas directement, uniquement via le référentiel.
Exemple	La Figure 6.9 est un exemple d'un IDE où les composants utilisent un référentiel d'informations de conception de système. Chaque outil logiciel génère des informations qui peuvent ensuite être utilisées par d'autres outils.
Quand utilisé	Vous devriez utiliser ce patron lorsque vous avez un système dans lequel de grands volumes d'informations sont générés et doivent être stockés pendant longtemps. Vous pouvez également l'utiliser dans des systèmes pilotés par les données (data-driven systems) où l'inclusion de données dans le référentiel déclenche une action ou un outil.
Avantages	Les composants peuvent être indépendants: ils n'ont pas besoin de connaître l'existence d'autres composants. Les modifications apportées par un composant peuvent être propagées à tous les composants. Toutes les données peuvent être gérées de manière cohérente car elles se font toutes en un seul endroit.
Désavantages	Le référentiel étant un point de défaillance unique, et donc, les problèmes dans le référentiel affectent l'ensemble du système. Peut être inefficace dans l'organisation de toutes les communications à travers le référentiel. La distribution du référentiel sur plusieurs ordinateurs peut s'avérer difficile.

Une architecture de référentiel pour un IDE



[9] **dépôt , entrepôt**

[10] **also called, Data-Centered Architecture, ou architecture centrée sur les données.**

3) Architecture client-serveur « Client-Server Architecture »

✧Modèle de système distribué qui montre comment les données et le traitement sont répartis sur une gamme de composants.

- Peut-être implémenté sur un seul ordinateur.

✧Ensemble de serveurs autonomes qui fournissent des services spécifiques tels que l'impression, la gestion de données, etc.

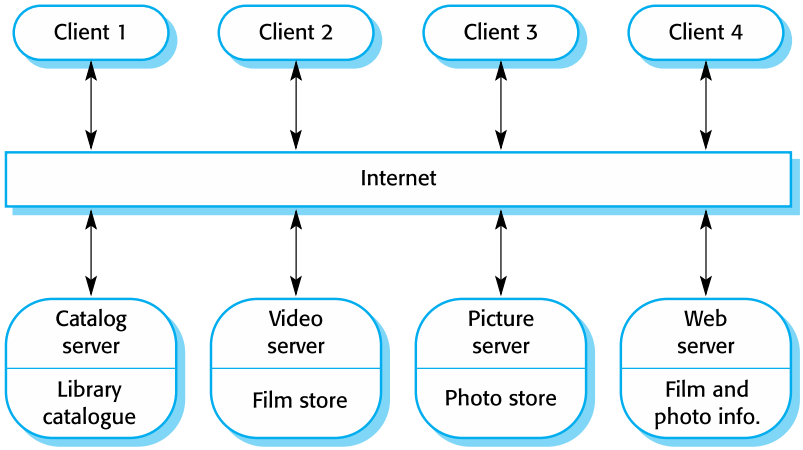
✧Ensemble de clients qui font appel à ces services.

✧Réseau qui permet aux clients d'accéder aux serveurs.

Le patron Client-serveur

Nom	Client-serveur
Description	Dans une architecture client-serveur, la fonctionnalité du système est organisée en services, chaque service étant fourni par un serveur distinct. Les clients sont des utilisateurs de ces services et des serveurs d'accès pour les utiliser.
Exemple	La Figure 6.11 est un exemple de bibliothèque de films et de vidéos/DVD organisée en tant que système client-serveur.
Quand utilisé	Utilisé lorsque les données d'une base de données partagée doivent être accessibles à partir d'une plage d'emplacements. Parce que les serveurs peuvent être répliqués, peuvent également être utilisés lorsque la charge sur un système est variable.
Avantages	Le principal avantage de ce modèle est que les serveurs peuvent être répartis sur un réseau. Une fonctionnalité générale (par exemple, un service d'impression) peut être disponible pour tous les clients et n'a pas besoin d'être implémentée par tous les services.
Désavantages	Chaque service est un point de défaillance unique susceptible de subir des attaques par déni de service ou une défaillance du serveur. Les performances peuvent être imprévisibles car elles dépendent du réseau et du système. Peut-être des problèmes de gestion si les serveurs appartiennent à différentes organisations.

Une architecture client-serveur pour une cinémathèque (bibliothèque de films)



4) L'architecture de tuyau et filtre « Pipe and filter architecture »[11]

✧Les transformations fonctionnelles traitent leurs entrées pour produire des sorties.

✧Peut-être référé comme modèle de tuyau et de filtre (comme dans le shell UNIX).

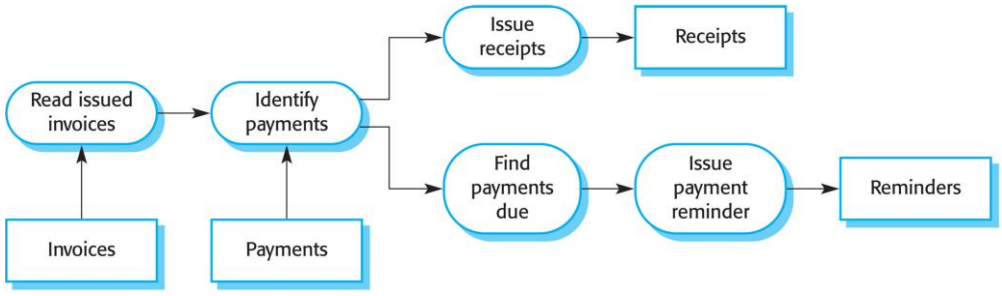
✧Les variantes de cette approche sont très courantes. Lorsque les transformations sont séquentielles, il s'agit d'un modèle séquentiel par lots qui est largement utilisé dans les systèmes de traitement de données.

✧Pas vraiment adapté pour les systèmes interactifs.

Le patron de tuyau et filtre

Nom	Tuyau et filtre “Pipe and filter”
Description	Le traitement des données dans un système est organisé de sorte que chaque composant de traitement (filtre) soit discret et exécute un type de transformation de données. Les données circulent (comme dans un tuyau) d'un composant à l'autre pour le traitement.
Exemple	La Figure 6.13 est un exemple de système de tuyau et de filtre utilisé pour le traitement des factures.
Quand utilisé	Généralement utilisé dans les applications de traitement de données (à la fois par lots et par transaction) où les entrées sont traitées dans des étapes distinctes pour générer des sorties connexes.
Avantages	Facile à comprendre et prend en charge la réutilisation de la transformation. Le style de workflow correspond à la structure de nombreux processus métier. L'évolution en ajoutant des transformations est simple. Peut-être implémenté en tant que système séquentiel ou concurrent.
Désavantages	Le format de transfert de données doit être convenu entre les transformations communicantes. Chaque transformation doit analyser son entrée et séparer sa sortie de la forme convenue. Cela augmente le surdébit du système et peut signifier qu'il est impossible de réutiliser des transformations fonctionnelles qui utilisent des structures de données incompatibles.

Un exemple de l'architecture de tuyau et de filtre utilisée dans un système de traitement de factures



[11] (also called, Data-Flow Architecture ou architecture de flux de données)

5. Architectures d'application

✧Les systèmes d'application sont conçus pour répondre à un besoin organisationnel.

✧Comme les entreprises ont beaucoup en commun, leurs systèmes d'application ont également tendance à avoir une architecture commune qui reflète les exigences de l'application.

✧Une architecture d'application générique est une architecture pour un type de système logiciel qui peut être configuré et adapté pour créer un système répondant à des exigences spécifiques.

Utilisation des architectures d'application

✧Comme un point de départ pour la conception architecturale.

✧Comme une liste de vérification de conception.

✧Comme un moyen d'organiser le travail de l'équipe de développement.

✧Comme un moyen d'évaluer les composants pour la réutilisation.

✧Comme un vocabulaire pour parler des types d'applications.

Types d'application

✧Applications de traitement de données
- Applications pilotées par les données qui traitent les données par lots sans intervention explicite de l'utilisateur pendant le traitement.

✧Applications de traitement de transactions
- Applications centrées sur les données qui traitent les demandes des utilisateurs et mettent à jour les informations dans une base de données système.

✧Systèmes de traitement d'événement
- Applications dans lesquelles les actions du système dépendent de l'interprétation des événements de l'environnement du système.

✧Systèmes de traitement du langage
- Applications où les intentions des utilisateurs sont spécifiées dans un langage formel qui est traité et interprété par le système

Exemples de type d'application

✧Deux architectures d'application génériques très répandues sont les systèmes de traitement des transactions et les systèmes de traitement de la langue.

✧Systèmes de traitement des transactions
- Systèmes de commerce électronique;
- Systèmes de réservation

✧Systèmes de traitement du langage
- Compilateurs;

- Interpréteurs de commande.

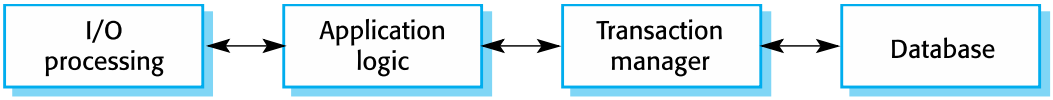
Systèmes de traitement des transactions

✧Traiter les requêtes des utilisateurs à partir d'une base de données ou les requêtes de mise à jour de la base de données.

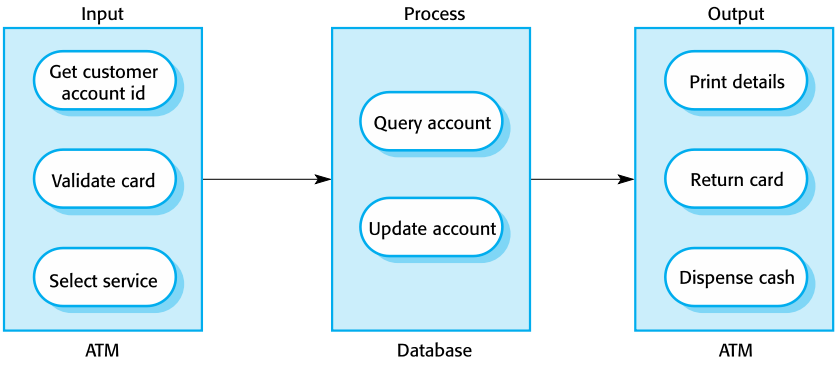
✧Du point de vue de l'utilisateur, une transaction est:
- Toute séquence cohérente d'opérations qui satisfait un but;
- Par exemple - trouvez les horaires des vols de Londres à Paris.

✧Pour un service, les utilisateurs effectuent des requêtes asynchrones qui sont ensuite traitées par un gestionnaire de transactions.

La structure des applications de traitement des transactions



L'architecture logicielle d'un système ATM



Architecture des systèmes d'information

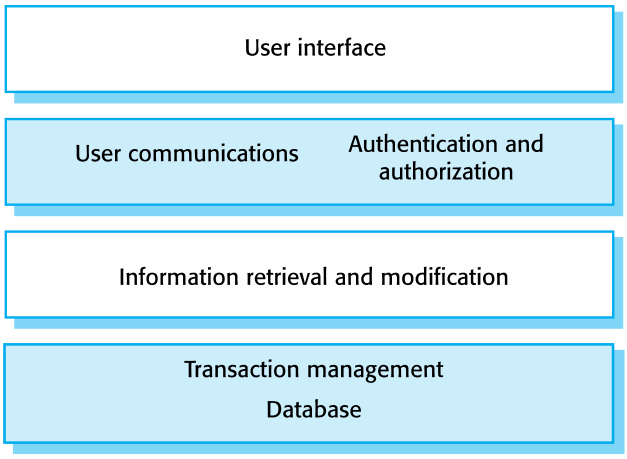
✧Les systèmes d'information ont une architecture générique qui peut être organisée comme une architecture en couches.

✧Ce sont des systèmes basés sur des transactions car l'interaction avec ces systèmes implique généralement des transactions de base de données.

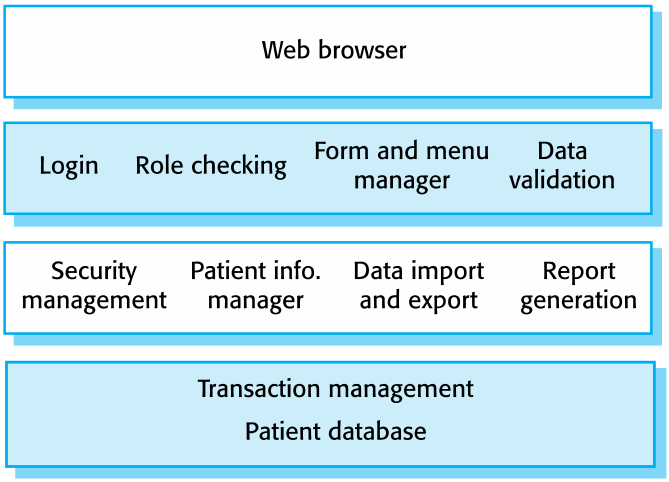
✧Les couches comprennent:
- L'interface utilisateur
- Communications de l'utilisateur

- Recherche (récupération) de l'information
- Base de données système

Architecture en couches de système d'information



L'architecture du système Mentcare



Systèmes d'information basé sur le Web

✧Les systèmes de gestion des informations et des ressources sont maintenant généralement des systèmes basés sur le Web où les interfaces utilisateur sont implémentées à l'aide d'un navigateur Web.

✧Par exemple, les systèmes de commerce électronique sont des systèmes de gestion des ressources basés sur Internet qui acceptent les commandes électroniques de biens ou de services, puis organisent la livraison de ces biens ou services au client.

✧Dans un système de commerce électronique, la couche spécifique à l'application comprend des fonctionnalités supplémentaires supportant un «panier» dans lequel les utilisateurs peuvent placer un certain nombre d'articles dans des transactions distinctes, puis les payer tous ensemble en une seule transaction.

Implémentation du serveur

✧Ces systèmes sont souvent implémentés en tant que architecture client-serveur multi-niveaux

- Le serveur Web est responsable de toutes les communications de l'utilisateur, l'interface utilisateur implémenté à l'aide d'un navigateur Web;
- Le serveur d'application est responsable de l'implémentation de la logique spécifique à l'application ainsi que des requêtes de stockage et de récupération d'informations;
- Le serveur de base de données déplace les informations vers et depuis la base de données et gère la gestion des transactions.

Systèmes de traitement du langage

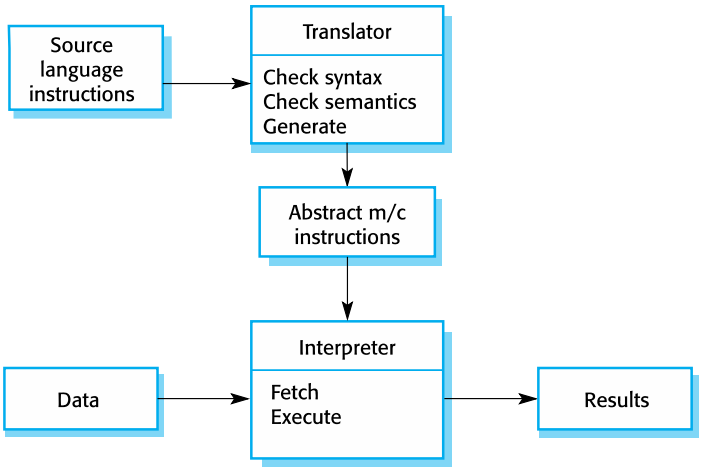
✧Accepter un langage naturel ou artificiel comme entrée et générer une autre représentation de ce langage.

✧Peut inclure un interpréteur pour agir selon les instructions écrites en langage en cours de traitement.

✧Utilisé dans les situations où le moyen le plus simple de résoudre un problème est de décrire un algorithme ou de décrire les données du système.

- Les outils de « MetaCASE tools[12] » traitent les descriptions d'outils, les règles de méthode, etc. et génèrent des outils.

L'architecture d'un système de traitement du langage



Composants du compilateur

✧Un analyseur lexical, qui prend les tokens (jetons) de langage d'entrée et les convertit en une forme interne.

✧Une table de symboles, qui contient des informations sur les noms des entités (variables, noms de

[12] Un outil **MetaCASE** est un type de logiciel d'application qui offre la possibilité de créer une ou plusieurs méthodes de modélisation, langages ou notations à utiliser dans le processus de développement logiciel. Souvent, le résultat est un outil de modélisation pour ce langage. Les outils MetaCASE sont donc une sorte de workbench (table de travail) de langage, généralement considéré comme focalisé sur les langages de modélisation graphique.

classes, noms d'objets, etc.) utilisés dans le texte en cours de traduction.

✧Un analyseur de syntaxe, qui vérifie la syntaxe du langage en cours de traduction.

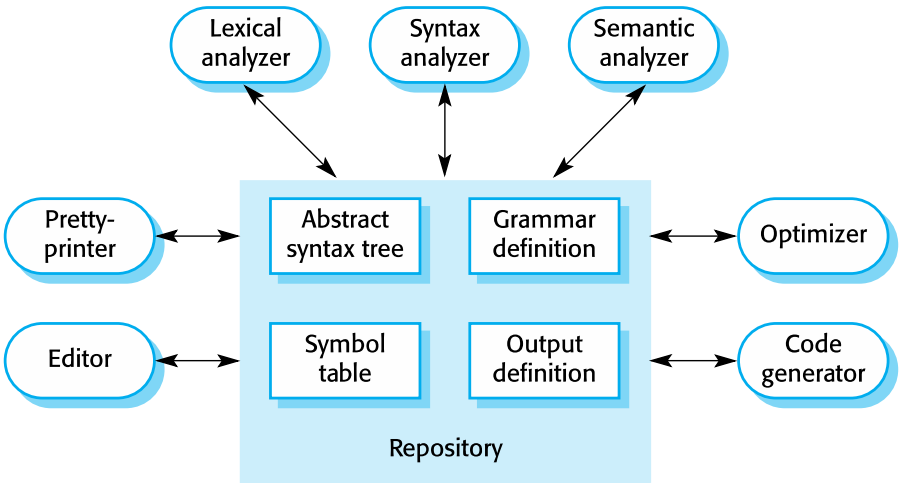
✧Un arbre de syntaxe, qui est une structure interne représentant le programme en cours de compilation.

Composants du compilateur

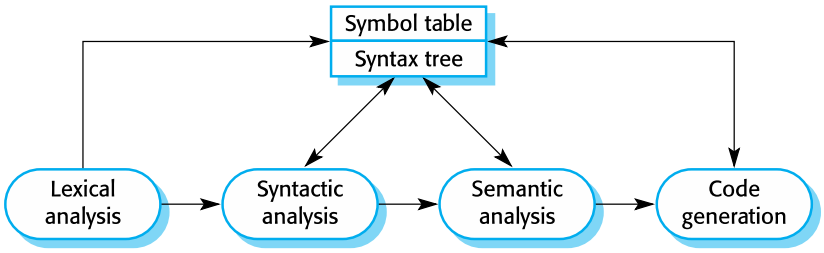
✧Un analyseur sémantique qui utilise les informations de l'arbre de syntaxe et de la table des symboles pour vérifier la correction sémantique du texte de la langue d'entrée.

✧Un générateur de code qui «parcourt» l'arbre de syntaxe et génère un code machine abstrait.

Architecture de référentiel pour un système de traitement du langage



Une architecture de compilateur de tuyaux et de filtres



6. Points clés

✓ Une architecture logicielle est une description de la façon dont un système logiciel est organisé.
✓ Les décisions de conception architecturale comprennent des décisions sur le type d'application, la

distribution du système, les styles architecturaux à utiliser.

- ✓ Les architectures peuvent être documentées à partir de plusieurs perspectives ou vues différentes, telles qu'une vue conceptuelle, une vue logique, une vue de processus et une vue de développement.
- ✓ Les modèles architecturaux sont un moyen de réutiliser les connaissances sur les architectures de systèmes génériques. Ils décrivent l'architecture, expliquent quand elle peut être utilisé et décrire ses avantages et ses inconvénients.
- ✓ Les modèles d'architectures de systèmes d'application nous aident à comprendre et à comparer les applications, à valider les conceptions de systèmes d'applications et à évaluer les composants à grande échelle en vue de leur réutilisation.
- ✓ Les systèmes de traitement des transactions sont des systèmes interactifs qui permettent à un certain nombre d'utilisateurs d'accéder et de modifier à distance les informations d'une base de données.
- ✓ Les systèmes de traitement de la langue sont utilisés pour traduire des textes d'une langue dans une autre et pour exécuter les instructions spécifiées dans la langue d'entrée. Ils comprennent un traducteur et une machine abstraite qui exécute le langage généré.

7. Exercices

I) Quiz :

Choisir la bonne réponse :

1. La conception architecturale est un processus créatif qui ne satisfait que les exigences fonctionnelles d'un système.

   a) Vrai
   b) Faux

2. Une vue _____ montre le matériel du système (le hardware) et la manière dont les composants logiciels sont répartis entre les processeurs du système.

   a) physique
   b) logique
   c) processus

3. Le langage UML a été conçu pour décrire _____.

   a) les systèmes orientés objet
   b) la conception architecturale
   c) SRS
   d) Les systèmes orientés objet et la conception architecturale

4. Laquelle des vues ci-dessous montre que le système se compose des processus d'interaction au moment de l'exécution?

   a) physique
   b) développement
   c) logique
   d) processus

5. Lequel des éléments suivants est un conflit d'architecture?

a) L'utilisation de composants à gros grains améliore les performances mais réduit la maintenabilité
b) L'introduction de données redondantes améliore la disponibilité mais rend la sécurité plus difficile
c) La localisation des fonctionnalités liées à la sécurité signifie généralement plus de communication, donc des performances dégradées
d) Tous ce qui est mentionné

6. Lequel des éléments suivants n'est pas inclus dans les décisions de la conception architecturale?

   a) type d'application
   b) la distribution du système
   c) les styles architecturaux
   d) Les tests du système

7. L'architecture, une fois établie, peut également être appliquée à d'autres produits.

   a) Vrai
   b) Faux

8. Lequel des modèles suivants est la base de la gestion des interactions dans des nombreux systèmes Web?

   a) l'architecture
   b) modèle de dépôt
   c) modèle-vue-contrôleur
   d) système d'exploitation différent

9. Qu'est-ce qui décrit comment un ensemble de composants en interaction peut partager des données?

   a) modèle-vue-contrôleur
   b) patron d'architecture
   c) modèle de dépôt
   d) Aucun des mentionnés

10. Quelle vue dans la conception architecturale montre les abstractions clés dans le système en tant qu'objets ou classes d'objets?

    a) physique
    b) développement
    c) logique
    d) processus

11. Lequel des types suivants est un modèle architectural?

    a) Modèle structurel statique
    b) Modèle de processus dynamique
    c) Modèle de distribution
    d) Tous les mentionnés

II) Questions de recherche :

1. Lorsque vous décrivez un système, expliquez pourquoi vous devrez peut-être concevoir l'architecture du système avant que la spécification des exigences ne soit complète.

2. Un système d'information doit être développé pour maintenir les informations sur les biens appartenant à une entreprise de services publics tels que bâtiments, véhicules, équipements, etc. Il est prévu que le personnel travaillant sur le terrain puisse les mettre à jour en utilisant les appareils mobiles. La société dispose de plusieurs bases de données existantes qui devraient être intégrées à travers ce système. Concevoir une architecture en couches pour ce système de gestion des biens basé sur l'architecture du système d'information générique illustré dans le chapitre 6.

3. En utilisant le modèle générique d'un système de traitement de langage présenté dans ce chapitre, concevoir l'architecture d'un système qui accepte les commandes en langage naturel et les traduire en requêtes de base de données dans un langage tel que SQL.

4. En utilisant le modèle de base d'un système d'information tel que présenté dans ce chapitre, suggérez les composants qui pourraient faire partie d'une application pour un appareil mobile qui affiche des informations sur les vols arrivant et partant d'un aéroport particulier.

8. Solutions

I) Quiz :

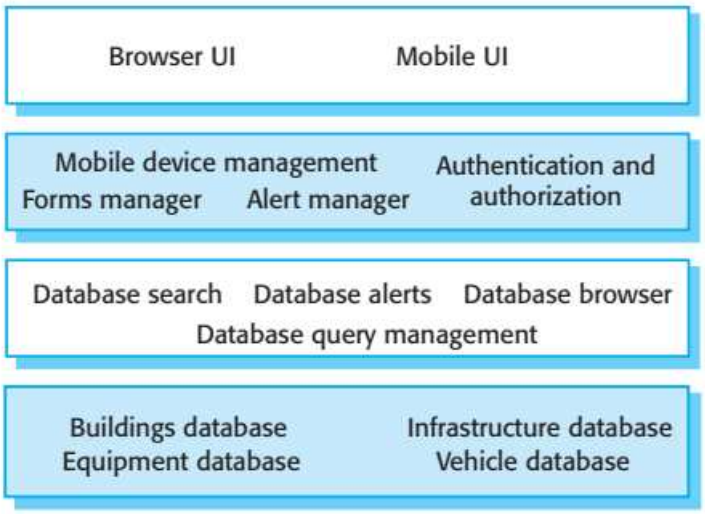
1 : b, En conception architecturale, vous concevez une organisation système répondant aux exigences fonctionnelles et non fonctionnelles d'un système.
2 : a, Une vue physique est implémentée par les ingénieurs système mettant en œuvre le matériel du système.
3 :d, Le langage UML a été conçu pour décrire les systèmes orientés objet et, au stade de la conception architecturale, vous voulez souvent décrire les systèmes à un niveau d'abstraction plus élevé.
4 : d, Cette vue est utile pour porter des jugements sur les caractéristiques non fonctionnelles du système telles que la performance et la disponibilité.
5 : d, L'architecture à haute disponibilité peut être affectée par plusieurs facteurs de conception qui doivent être maintenus pour garantir qu'aucun point de défaillance unique n'existe dans une telle conception.
6 : d, Les décisions de conception architecturale comprennent des décisions sur le type d'application, la distribution du système, les styles architecturaux à utiliser et les façons dont l'architecture doit être documentée et évaluée.
7 : b, Les systèmes du même domaine ont souvent des architectures similaires qui reflètent les concepts de domaine.
8 : c, Le modèle Model-View-Controller est la base de la gestion des interactions dans de nombreux systèmes basés sur le Web.
9 : c, La majorité des systèmes qui utilisent de grandes quantités de données sont organisés autour d'une base de données ou d'un référentiel partagé.
10 : c, Il est possible de relier les exigences système aux entités dans une vue logique.

11 : d, Tous ces modèles reflètent la stratégie de base utilisée pour structurer un système.

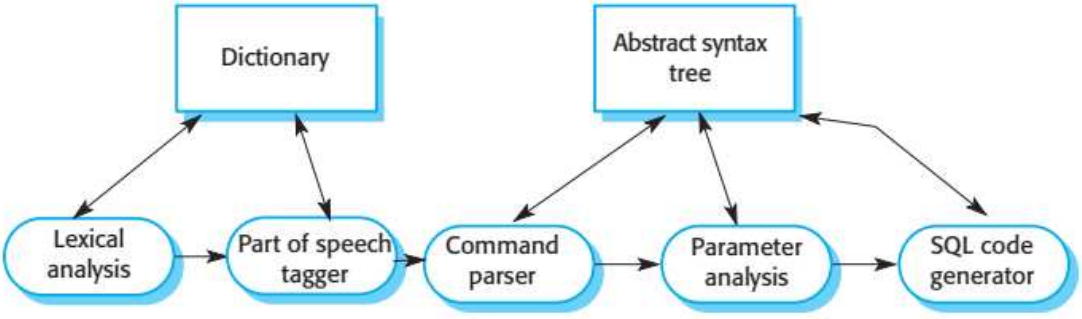
II) Questions de recherche :

1) L'architecture peut devoir être conçue avant que les spécifications ne soient écrites pour fournir un moyen de structurer la spécification et développer simultanément différentes spécifications de sous-système, pour permettre la fabrication de matériel par des sous-traitants et fournir un modèle pour l'évaluation du système.

2) L'architecture proposée :



3) L'architecture proposée :



4)

C'est un système hybride avec quelques éléments du système hébergé sur un serveur distant et certains éléments de l'application elle-même.

Vous devez prendre en compte les niveaux dans le système d'information et identifier les composants qui peuvent être inclus à chaque niveau. Des exemples de ces composants pourraient être:

Niveau 1 (niveau de la base de données)
Base de données de vol; Base de données de statut de vol; Informations sur l'aéroport

Niveau 2: (Niveau de recherche/récupération de l'information)
Gestion de statut; Gestion de vol; Chercher;

Niveau 3: (Niveau d'interaction de l'utilisateur)
Authentification; gestion de session; traitement des formulaires

Niveau 4 (interface utilisateur)
Application UI

Vous devez ensuite décider quels éléments du système d'information doivent être hébergés sur l'appareil mobile et lesquels doivent être hébergés à distance.

1. Niveau de base de données
Il n'a évidemment pas de sens d'essayer d'héberger des composants de base de données majeurs sur l'application, donc sur l'application, ils sont remplacés par un composant de requête de base de données qui donne accès à ces bases de données.

2. Le niveau de recherche/récupération de l'information
Il doit y avoir un composant de recherche dans l'application, mais il devrait vraiment être une interface vers une recherche de base de données plus étendue qui s'exécute sur le serveur. Les informations sur les vols d'intérêt pour l'utilisateur et leur statut doivent être collectés et gérés localement dans l'application.

3. Niveau d'interaction de l'utilisateur
Cela doit également être traité principalement dans l'application bien qu'il soit basé sur des informations stockées, par ex. l'authentification repose sur le stockage des informations d'identification de l'utilisateur et l'authentification automatique lorsque l'application est appelée.

4. Niveau de l'interface utilisateur
Exclusivement implémenté dans l'application.

Chapitre VII

Conception et Implémentation des Logiciels

Objectifs

- comprendre les activités les plus importantes dans un processus de conception général orienté objet;
- comprendre quelques modèles utilisés pour documenter une conception orientée objet;
- connaître l'idée de patrons de conception et comment ceux-ci sont un moyen de réutiliser les connaissances et l'expérience de conception;
- parmi les issues de l'implémentation d'un logiciel, on devrait comprendre la réutilisation de logiciels et le développement de logiciels libres.

Themes couverts

- Conception orientée objet en utilisant UML
- Patrons de conception
- Issues de l'implémentation
- Développement open source

Chapitre 7:

Conception et Implémentation des Logiciels

1. Introduction

Conception et Implémentation

✧La conception et l'implémentation du logiciel est une étape dans le processus de génie logiciel à laquelle un système de logiciel exécutable est développé.

✧Les activités de conception et de mise en œuvre (réalisation) d'un logiciel sont toujours entrelacées.

- La conception de logiciels est une activité créative dans laquelle vous identifiez les composants logiciels et leurs relations, en fonction des exigences du client.
- La mise en œuvre est le processus de réalisation de la conception comme un programme.

Construire ou acheter

✧Dans un large éventail de domaines, il est maintenant possible d'acheter des systèmes sur l'étagère (Commercial Off-The-Shelf ou COTS systems) qui peut être adapté aux besoins des utilisateurs.

- Par exemple, si vous souhaitez mettre en œuvre un système de dossiers médicaux, vous pouvez acheter un paquet qui est déjà utilisé dans les hôpitaux. Il peut être moins cher et plus rapide à utiliser cette approche plutôt que de développer un système dans un langage de programmation classique.

✧Lorsque vous développez une application de cette manière, le processus de conception se préoccupe de l'utilisation des fonctionnalités de configuration de ce système pour répondre aux exigences du système.

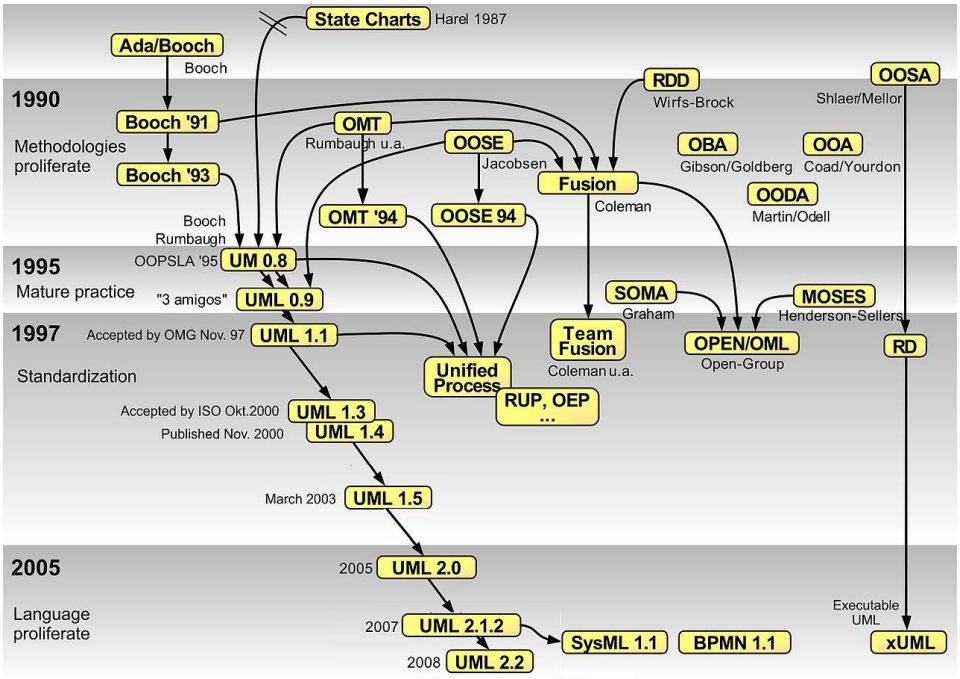
2. Conception orientée objet en utilisant UML

Méthodes existantes de conception

1) Méthodes fonctionnelles (ou cartésiennes, des années 70) : sont les premières méthodes d'analyse et

de conception qui se basent sur un processus de décomposition Top-Down des fonctions (ex. SADT)
2) Méthodes systémiques (des années 80): double démarche = Modélisation des données + modélisation des traitements (Merise, Axial,...).
3) L'émergence des méthodes orientée-objet (1990-1995)
Plus de 50 méthodes objet sont apparues durant cette période (Booch (Grady Booch), Classe-Relation, Fusion, HOOD, OMT (James Rumbaugh), OOA, OOD, OOM, OOSE (Ivar Jacobson)…)[13]

Méthodes Orientée Objet



N.B : Méthode = concepts + notation + processus

Un processus de conception orienté objet

✧Les processus structurés de conception orientée objet impliquent le développement d'un certain nombre de différents modèles de système.

✧Ils nécessitent beaucoup d'efforts pour le développement et la maintenance de ces modèles et, pour les petits systèmes, cela peut ne pas être rentable.

✧Cependant, pour les grands systèmes développés par différents groupes, les modèles de conception constituent un mécanisme de communication important.

Etapes du processus

✧Il existe une variété de différents processus de conception orientée objet qui dépendent de

[13] (1) et (2): approche descendante, 3: approche ascendante

l'organisation utilisant le processus.

✧Les activités communes dans ces processus comprennent:

1. Définir le contexte et les interactions externes avec le système.
2. Concevoir l'architecture du système;
3. Identifier les principaux objets du système;
4. Développer des modèles de conception;
5. Spécifier les interfaces d'objets.

✧Le processus illustré dans ce chapitre utilise une conception d'une station météorologique dans le désert (ou zone sauvage).

1. Contexte du système et interactions

✧Comprendre les relations entre le logiciel en cours de conception et son environnement externe est essentiel pour décider comment fournir les fonctionnalités requises du système et comment structurer le système pour communiquer avec son environnement.

✧La compréhension du contexte vous permet également d'établir les limites du système. La mise des limites du système vous permet de décider quelles fonctions sont implémentées dans le système en cours de conception et quelles fonctions sont présentes dans d'autres systèmes associés.

✧Pour l'exemple de station météo, vous devez décider comment la fonctionnalité est distribuée entre le système de contrôle pour toutes les stations météorologiques et le logiciel intégré dans la station météorologique elle-même.

Modèles de contexte et d'interaction

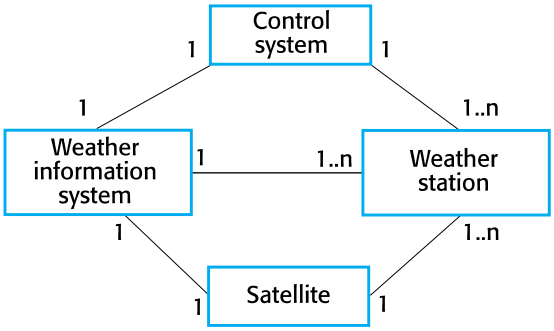
✧Le modèle de contexte du système et les modèles d'interaction présentent des vues complémentaires des relations entre un système et son environnement:

6. 1) Un modèle de contexte du système est un modèle structurel qui démontre les autres systèmes de l'environnement du système en cours de développement.
7. 2) Un modèle d'interaction est un modèle dynamique qui montre comment le système interagit avec son environnement tel qu'il est utilisé.

✧Le modèle de contexte d'un système peut être représenté en utilisant des associations. Les associations montrent simplement qu'il existe des relations entre les entités impliquées dans l'association. La nature des relations est maintenant spécifiée. Vous pouvez donc documenter l'environnement du système en utilisant un diagramme simple, en montrant les entités du système et leurs associations.

✧Les systèmes de l'environnement de chaque station météorologique sont un système d'information météorologique, un système de satellite embarqué et un système de contrôle. Les informations de cardinalité sur le lien montrent qu'il existe un système de contrôle mais plusieurs stations météorologiques, un satellite et un système d'information météorologique général.

Contexte du système de la station météo « Weather station »



Modèles de contexte et d'interaction

✧Lorsque vous modélisez les interactions d'un système avec son environnement, vous devez utiliser une approche abstraite qui n'inclut pas trop de détails. Une façon de faire est d'utiliser un modèle de cas d'utilisation.

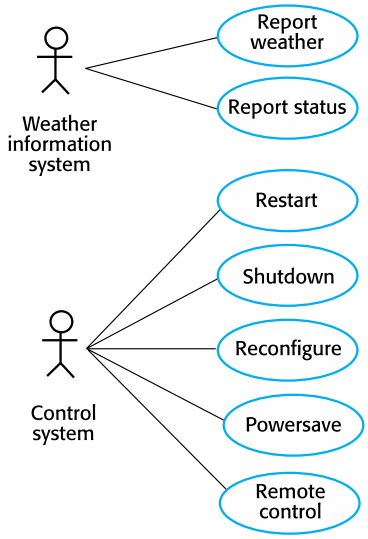
✧Chaque cas d'utilisation représente une interaction avec le système.

✧Chaque interaction possible est nommée dans une ellipse et l'entité externe impliquée dans l'interaction est représentée par une figure de « Stick man».

✧Le modèle de cas d'utilisation de la station météorologique est illustré à la figure suivante.

✧Cela montre que la station météorologique interagit avec le système d'information météorologique pour rapporter les données météorologiques et l'état du matériel de la station météorologique.

✧D'autres interactions sont avec un système de contrôle qui peut émettre des commandes de contrôle de stations météorologiques spécifiques.



✧Une figure « Stick » est utilisé dans l'UML pour représenter d'autres systèmes ainsi que des utilisateurs humains

✧Chacun de ces cas d'utilisation doit être décrit en langage naturel structuré. Cela aide les concepteurs à identifier les objets dans le système et leur permet de comprendre ce que le système est censé faire.

Les cas d'utilisation de la station météo

Report weather: send weather data to the weather information system
Report status: send status information to the weather information system
Restart: if the weather station is shut down, restart the system

Shutdown: shut down the weather station
Reconfigure: reconfigure the weather station software
Powersave: put the weather station into power-saving mode.
Remote control: send control commands to any weather station subsystem

Description du cas d'utilisation « Report weather »

System	Weather station
Use case	Report weather
Actors	Weather information system
Description	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum, and average wind speeds; the total rainfall; and the wind direction as sampled at five-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and may be modified in the future.

2. Conception architecturale

✧Une fois les interactions entre le système et son environnement sont comprises, vous utilisez ces informations pour la conception de l'architecture du système.

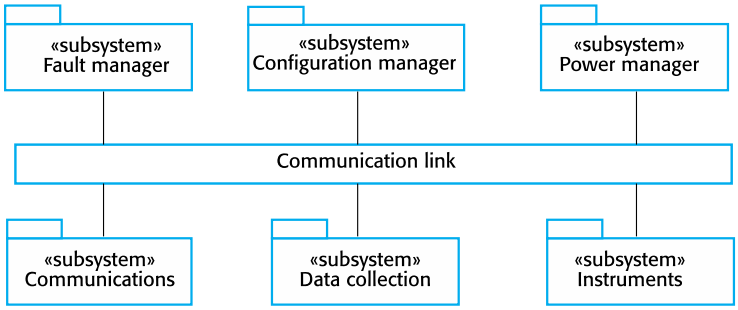
✧Vous identifiez les principaux éléments qui composent le système et leurs interactions, et peut ensuite organiser les composants à l'aide d'un modèle architectural (patron) comme un modèle en couches ou client-serveur.

✧La station météo est composée de sous-systèmes indépendants qui communiquent par la diffusion de messages sur une infrastructure commune (Ceci est un autre style architectural couramment utilisé).

✧Chaque sous-système écoute les messages sur cette infrastructure et récupère les messages qui leur

sont destinés.

Architecture de haut niveau de la station météo

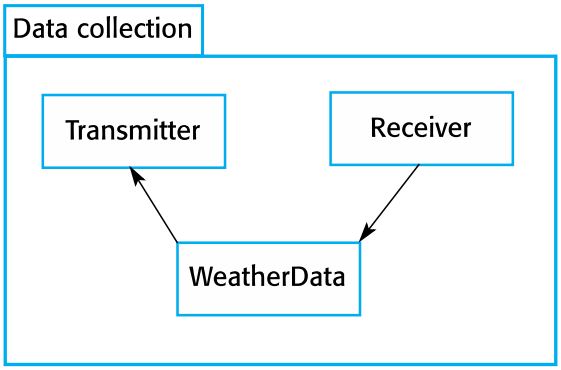


Architecture de haut niveau de la station météo

✧Par exemple, lorsque le sous-système de communication reçoit une commande de contrôle, telle que shutdown, la commande est récupérée par chacun des autres sous-systèmes, qui se ferment ensuite correctement. Le principal avantage de cette architecture est qu'il est facile de prendre en charge différentes configurations de sous-systèmes car l'expéditeur d'un message n'a pas besoin d'adresser le message à un sous-système particulier.

✧La Figure suivante montre l'architecture du sous-système de collecte de données « Data Collection », qui est incluse dans la Figure précédente. Les objets « Transmitter » et « Receiver » sont concernés par la gestion des communications et l'objet « WeatherData » encapsule les informations collectées par les instruments et transmises au système d'informations météorologiques. Cet arrangement suit le modèle classique de producteur-consommateur.

Architecture de système de collecte des données



3. Identification des classes d'objets

✧Identifier les classes d'objets est souvent une partie difficile dans la conception orientée objet.

✧Il n'y a aucune «formule magique» pour l'identification de l'objet. Elle s'appuie sur la compétence, l'expérience

et le domaine des connaissances des concepteurs de système.

✧L'identification de l'objet est un processus itératif. Il est peu probable de l'obtenir dès la première fois.

✧À ce stade du processus de conception, vous devriez avoir quelques idées sur les objets essentiels du système que vous concevez. Au fur et à mesure que votre compréhension de la conception se développe, vous affinez ces idées sur les objets du système.

✧La description du cas d'utilisation aide à identifier les objets et les opérations dans le système.

✧Avec ces objets en tête, vous pouvez commencer à identifier les classes d'objets dans le système.

Approches d'identification

1) Utiliser une analyse grammaticale basée sur une description en langue naturelle du système à construire (Abbott, 1983). Les objets et les attributs sont des noms; les opérations ou les services sont des verbes.

2) Utiliser des entités tangibles dans le domaine d'application telles que les avions, les rôles tels que gérant ou médecin, les événements tels que les demandes, les interactions telles que les réunions, les lieux tels que les bureaux, les unités organisationnelles telles que les entreprises, les données telles que les informations capturées, et ainsi de suite. (Coad and Yourdon, 1990, Shlaer et Mellor, 1988, Wirfs-Brock et al., 1990).

3) Utilisez une approche comportementale et identifier les objets en fonction de ce qui participe à un comportement.

4) Utiliser une analyse basée sur des scénarios où différents scénarios d'utilisation du système sont identifiés et analysés. Comme chaque scénario est analysé, l'équipe responsable de l'analyse doit identifier les objets, les attributs et les opérations requis (Beck et Cunningham, 1989).

✧En pratique, vous devez utiliser plusieurs sources de connaissances pour découvrir des classes d'objets.

✧Les classes d'objets, les attributs et les opérations initialement identifiés à partir de la description informelle du système peuvent constituer un point de départ pour la conception.

✧Des informations supplémentaires provenant de la connaissance du domaine d'application ou de l'analyse de scénario peuvent ensuite être utilisées pour affiner et étendre les objets initiaux.

✧Ces informations peuvent être collectées à partir de documents d'exigences, de discussions avec les utilisateurs ou d'analyses de systèmes existants.

Les classes d'objets de la station Météo

✧L'identification de la classe d'objets dans le système de station météorologique peut être basée sur le matériel tangible et les données du système:

✧Thermomètre de sol, anémomètre, baromètre

- Les objets du domaine d'application qui sont des objets «matériels» liés aux instruments du

système.

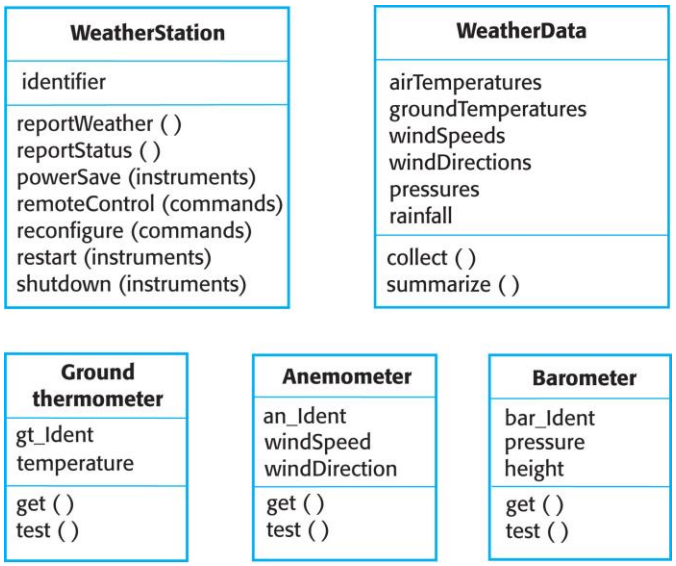
✧**Station météo**

- L'interface de base de la station météo avec son environnement. Il reflète donc les interactions identifiées dans le modèle de cas d'utilisation.

✧**Données météo**

- Encapsule les données résumées des instruments.

Les classes d'objets de la station Météo



N.B: Il pourrait y avoir d'autres classes d'objets selon la description du système

Les classes d'objets de la station Météo

✧La classe d'objets WeatherStation fournit l'interface de base de la station météo avec son environnement. Ses opérations reflètent les interactions illustrées dans la figure précédente. Dans ce cas, une seule classe d'objets est utilisée pour encapsuler toutes ces interactions, mais dans d'autres conceptions, vous pouvez concevoir l'interface système en plusieurs classes différentes.

✧La classe d'objets WeatherData est responsable du traitement de la commande de rapport météo. Il envoie les données résumées des instruments de la station météorologique au système d'information météorologique.

Les classes d'objets de la station Météo

✧Les classes d'objets Thermometer, Anemometer, et Barometer sont directement liées aux instruments du système. Ils reflètent les entités matérielles tangibles dans le système et les opérations sont concernées par le contrôle de ce matériel. Ces objets fonctionnent de manière autonome pour collecter des données à la fréquence spécifiée et stocker les données collectées localement. Ces données sont fournies à l'objet WeatherData sur demande.

Les classes d'objets de la station Météo

✧Vous utilisez la connaissance du domaine d'application pour identifier d'autres objets, attributs et services. Nous savons que les stations météorologiques sont souvent situées dans des endroits éloignés et comprennent divers instruments qui parfois vont mal. Les pannes d'instrument doivent être signalées automatiquement.

✧Cela implique que vous avez besoin d'attributs et d'opérations pour vérifier le bon fonctionnement des instruments. Il y a beaucoup de stations météorologiques à distance, donc chaque station météorologique devrait avoir son propre identifiant.

Les classes d'objets de la station Météo

✧À ce stade du processus de conception, vous devriez vous concentrer sur les objets eux-mêmes, sans penser à la façon dont ils pourraient être implémentés.

✧Une fois que vous avez identifié les classes d'objets, affinez la conception de l'objet. Vous recherchez des fonctionnalités communes, puis concevez la hiérarchie d'héritage pour le système.

✧Par exemple, vous pouvez identifier une super-classe Instrument, qui définit les caractéristiques communes à tous les instruments, tels qu'un identifiant, et les opérations « get » et « test ». Vous pouvez également ajouter de nouveaux attributs et opérations à la super-classe, tels qu'un attribut qui conserve la fréquence de la collecte de données.

4. Modèles de conception

✧Les modèles de conception ou de système montrent les objets ou les classes d'objets dans un système. Ils montrent également les associations et les relations entre ces entités.

✧Ces modèles sont le pont entre les exigences du système et l'implémentation. Ils doivent être abstraits afin que les détails inutiles ne cachent pas les relations entre eux et les exigences du système. Cependant, ils doivent également inclure suffisamment de détails pour permettre aux programmeurs de prendre des décisions de mise en œuvre.

✧Généralement, vous contournez ce type de conflit en développant des modèles à différents niveaux de détail.

✧Où il y a des liens étroits entre les ingénieurs de spécification des exigences, les concepteurs et les programmeurs, alors les modèles abstraits peuvent être tout ce qui est exigé.

✧Des décisions de conception spécifiques peuvent être prises lors de l'implémentation du système, avec des problèmes résolus par des discussions informelles.

✧Lorsque les liens entre les ingénieurs de spécification du système, les concepteurs et les programmeurs sont indirects (par exemple, lorsqu'un système est conçu dans une partie d'une organisation mais implémenté ailleurs), des modèles plus détaillés sont susceptibles d'être nécessaires.

✧Une étape importante dans le processus de conception consiste donc à décider des modèles de conception dont vous avez besoin et le niveau de détail requis dans ces modèles.

✧Cela dépend du type de système en cours de développement. Vous concevez un système de

traitement de données séquentiel (data-processing system) d'une manière différente d'un système en temps réel embarqué (embedded real-time system), et donc vous aurez besoin de différents modèles de conception.

✧Le langage UML prend en charge 13 types de modèles différents, mais vous utilisez rarement tous ces modèles.

✧Minimiser le nombre de modèles produits réduit les coûts de conception et le temps nécessaire pour terminer le processus de conception.

✧Lorsque vous utilisez le langage UML pour développer une conception, vous développez normalement deux types de modèle de conception:

- 1) Modèles structurels, qui décrivent la structure statique du système en utilisant des classes d'objets et leurs relations. Les relations importantes qui peuvent être documentées à ce stade sont les relations de généralisation (héritage), les relations uses/used-by et les relations de composition.
- 2) Modèles dynamiques, qui décrivent la structure dynamique du système et montrent les interactions entre les objets du système. Les interactions qui peuvent être documentées incluent la séquence des demandes de service effectuées par les objets et les changements d'état qui sont déclenchés par ces interactions d'objet.

Exemples de modèles de conception

✧Dans les premières étapes du processus de conception, SOMMERVILLE pense qu'il y a trois modèles qui sont particulièrement utiles pour ajouter des détails aux cas d'utilisation et aux modèles architecturaux:

- 1) **Modèles de sous-système**, qui montrent les regroupements logiques des classes d'objets dans des sous-systèmes cohérents. Ceux-ci sont représentés en utilisant une forme de diagramme de classes avec chaque sous-système représenté sous la forme d'un package avec des classes d'objets incluses. Les modèles de sous-système sont des modèles statiques (structurels).
- 2) **Modèles de séquence**, qui montrent la séquence des interactions de l'objet. Ceux-ci sont représentés à l'aide d'un diagramme UML de séquence ou de collaboration. Les modèles de séquence sont des modèles dynamiques.

- 3) **Modèles de machine à états**, qui montrent comment les objets individuels changent d'état en réponse aux événements. Ceux-ci sont représentés dans l'UML en utilisant des diagrammes d'états. Les modèles de machines d'état sont des modèles dynamiques

Modèles de sous-systeme

✧Un modèle de sous-système est un modèle statique utile car il montre comment une conception est organisée en groupes d'objets logiquement liés (comme dans la figure du slide #20).

✧En plus des modèles de sous-systèmes, vous pouvez également concevoir des modèles de classes d'objets détaillés, montrant tous les objets dans les systèmes et leurs associations (héritage, généralisation, agrégation, etc.).

✧Cependant, il y a un risque à faire trop de modélisation. Vous ne devriez pas prendre de décisions détaillées sur la mise en œuvre qui devrait vraiment être laissée aux programmeurs du système.

Modèles de séquence

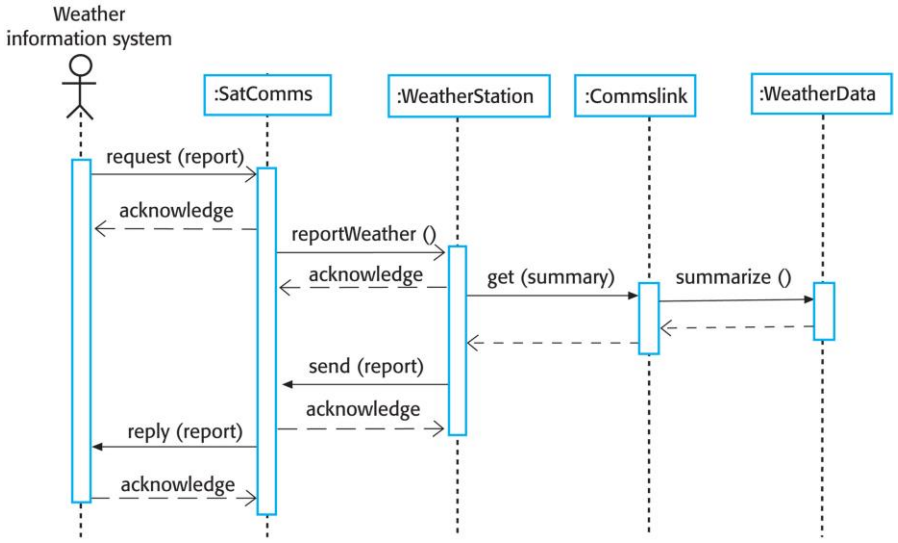
✧Les modèles de séquence sont des modèles dynamiques qui décrivent, pour chaque mode d'interaction, la séquence des interactions d'objets qui ont lieu.

✧Lors de la documentation d'une conception, vous devez produire un modèle de séquence pour chaque interaction significative. Si vous avez développé un modèle de cas d'utilisation, il devrait y avoir un modèle de séquence pour chaque cas d'utilisation que vous avez identifié.

✧Modèles de séquences montrent la séquence des interactions d'objets qui ont lieu

- Les objets sont disposés horizontalement dans la partie supérieure;
- Le temps est représenté verticalement et les modèles sont lues de haut en bas;
- Les interactions sont représentées par des flèches marquées, différents styles de flèche représentent différents types d'interaction;
- Un mince rectangle dans une ligne de vie de l'objet représente le moment où l'objet est l'objet de contrôle dans le système.

Diagramme de séquence décrivant la collecte de données



✧Le diagramme précédent montre la séquence des interactions qui se produisent lorsqu'un système externe demande les données résumées de la station météorologique. Vous lisez les diagrammes de séquence de haut en bas:

1. The SatComms object receives a request from the weather information system to collect a weather report from a weather station. It acknowledges receipt of this request. The stick arrowhead on the sent message indicates that the external system does not wait for a reply but can carry on with other processing.
2. SatComms sends a message to WeatherStation, via a satellite link, to create a summary of the collected weather data. Again, the stick arrowhead indicates that SatComms does not suspend itself waiting for a reply.
3. WeatherStation sends a message to a Commslink object to summarize the weather data. In this case, the squared-off style of arrowhead indicates that the instance of the WeatherStation object class waits for a reply.
4. Commslink calls the summarize method in the object WeatherData and waits for a reply.

5. The weather data summary is computed and returned to WeatherStation via the Commslink object.
6. WeatherStation then calls the SatComms object to transmit the summarized data to the weather information system, through the satellite communications system.

Diagramme de séquence décrivant la collecte de données

✧Les objets SatComms et WeatherStation peuvent être implémentés en tant que processus concurrents, dont l'exécution peut être suspendue et reprise. L'instance d'objet SatComms écoute les messages du système externe, décode ces messages et lance les opérations de la station météo.

✧Les diagrammes de séquence sont utilisés pour modéliser le comportement combiné d'un groupe d'objets, mais vous pouvez également résumer le comportement d'un objet ou d'un sous-système en réponse à des messages et des événements.

✧Pour ce faire, vous pouvez utiliser un modèle de machine d'état qui montre comment l'instance d'objet change d'état en fonction des messages qu'il reçoit. L'UML comprend des diagrammes d'états, initialement inventés par Harel (1987) pour décrire les modèles de machine d'état.

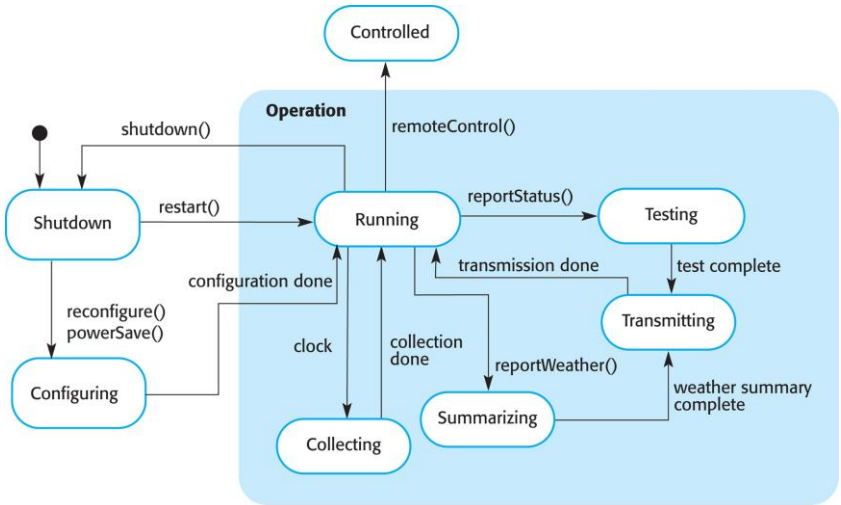
Les diagrammes d'états

✧Les diagrammes d'états sont utilisés pour montrer comment les objets répondent aux différentes demandes de service et les transitions d'état déclenchées par ces demandes.

✧Les diagrammes d'états sont des modèles utiles de haut niveau d'un système ou d'un comportement d'exécution d'un objet.

✧Vous n'avez pas généralement besoin d'un diagramme d'état pour tous les objets dans le système. Beaucoup des objets dans un système sont relativement simples et un modèle d'état ajoute des détails inutiles à la conception.

Diagramme d'état de station météo



✧La figure précédente est un diagramme d'état du système de station météorologique qui montre comment il répond aux demandes de divers services.

✧Vous pouvez lire ce diagramme comme suit:

1. If the system state is Shutdown then it can respond to a restart(), a reconfigure(), or a powerSave() message. The unlabeled arrow with the black blob indicates that the Shutdown state is the initial state. A restart() message causes a transition to normal operation. Both the powerSave() and reconfigure() messages cause a transition to a state in which the system reconfigures itself. The state diagram shows that reconfiguration is only allowed if the system has been shut down.
2. In the Running state, the system expects further messages. If a shutdown() message is received, the object returns to the shutdown state.
3. If a reportWeather() message is received, the system moves to the Summarizing state. When the summary is complete, the system moves to a Transmitting state where the information is transmitted to the remote system. It then returns to the Running state.
4. If a reportStatus() message is received, the system moves to the Testing state, then the Transmitting state, before returning to the Running state.
5. If a signal from the clock is received, the system moves to the Collecting state, where it collects data from the instruments. Each instrument is instructed in turn to collect its data from the associated sensors.
6. If a remoteControl() message is received, the system moves to a controlled state in which it responds to a different set of messages from the remote control room. These are not shown on this diagram.

5. Spécification d'interface

✧Une partie importante de tout processus de conception est la spécification des interfaces entre les composants de la conception. Vous devez spécifier des interfaces pour que les objets et les sous-systèmes puissent être conçus en parallèle. Une fois qu'une interface a été spécifiée, les développeurs d'autres objets peuvent supposer que l'interface sera implémentée

✧La conception de l'interface concerne la spécification du détail de l'interface à un objet ou à un groupe d'objets. Cela signifie définir les signatures et la sémantique des services fournis par l'objet ou par un groupe d'objets. Les interfaces peuvent être spécifiées dans le langage UML en utilisant la même notation qu'un diagramme de classes. Cependant, il n'y a pas de section d'attribut et le stéréotype UML « interface » devrait être inclus dans la partie name. La sémantique de l'interface peut être définie à l'aide du langage OCL (Object Contraint Language).

✧Vous ne devez pas inclure les détails de la représentation des données dans une conception d'interface, car les attributs ne sont pas définis dans une spécification d'interface. Cependant, vous devez inclure des opérations pour accéder et mettre à jour les données. Comme la représentation des données est masquée, elle peut être facilement modifiée sans affecter les objets qui utilisent ces données. Cela conduit à une conception qui est intrinsèquement plus maintenable. Par exemple, une représentation de tableau d'une pile peut être changée en une représentation de liste sans affecter les autres objets qui utilisent la pile.

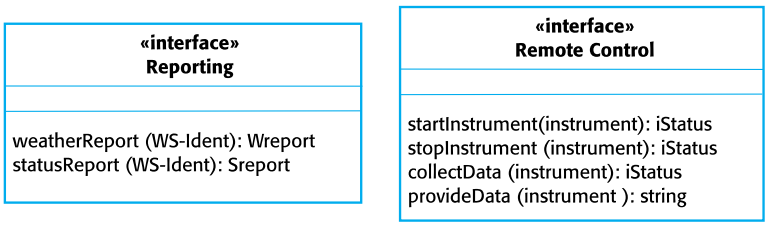
✧En revanche, il est souvent judicieux d'exposer les attributs dans un modèle de conception statique, car c'est le moyen le plus compact d'illustrer les caractéristiques essentielles des objets.

✧Il n'y a pas de simple relation 1:1 entre les objets et les interfaces. Le même objet peut avoir plusieurs

interfaces, dont chacune est un point de vue sur les opérations qu'il fournit. Ceci est pris en charge directement dans Java, où les interfaces sont déclarées séparément des objets et des interfaces d'implémentation des objets. De même, un groupe d'objets peut être accessible via une seule interface.

✧La Figure suivante montre deux interfaces qui peuvent être définies pour la station météorologique. L'interface de gauche est une interface de création de rapports qui définit les noms d'opération utilisés pour générer les rapports de météo et d'état. Ceux-ci correspondent directement aux opérations dans l'objet WeatherStation. L'interface de contrôle à distance fournit quatre opérations, qui sont mappées sur une méthode unique dans l'objet WeatherStation. Dans ce cas, les opérations individuelles sont codées dans la chaîne de commande associée à la l'opération remoteControl, illustrée à la Figure précédente.

Interfaces de station Météo



3. Patrons de conception « Design Patterns »

Les Patrons de Conception « Design Patterns »

✧Un patron de conception est un moyen de réutiliser des connaissances abstraites sur un problème et sa solution.

✧Un patron est une description du problème et de l'essence de sa solution.

✧Il devrait être suffisamment abstrait pour être réutilisé dans différents contextes.

✧Les descriptions de patrons utilisent généralement des caractéristiques orientées objet telles que l'héritage et le polymorphisme.

✧"Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience."

1. Nom
   - Un identificateur de motif valable.
2. Description du problème.
3. Description de la solution.
   - N'est pas une conception concrète mais un modèle (template) pour une solution de conception qui peut être instancié de différentes manières.
4. Conséquences
   - Les résultats et les compromis de l'application du modèle.

Exemple : Le modèle Observateur

1. Nom

- Observateur.

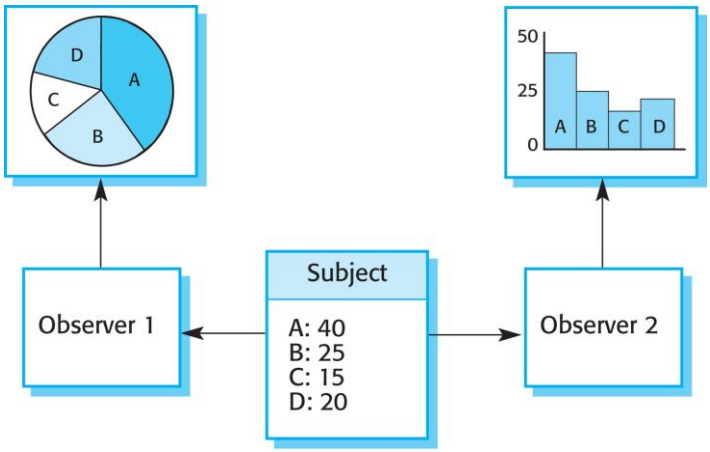
2. Description
   - Sépare l'affichage d'état de l'objet de l'objet lui-même.
3. Description du problème
   - Utilisé lorsque plusieurs affichages d'états sont nécessaires.
4. Description de la solution
   - Voir la description UML suivante.
5. Conséquences
   - Les optimisations pour améliorer les performances d'affichage sont impraticables.

Le modèle Observateur

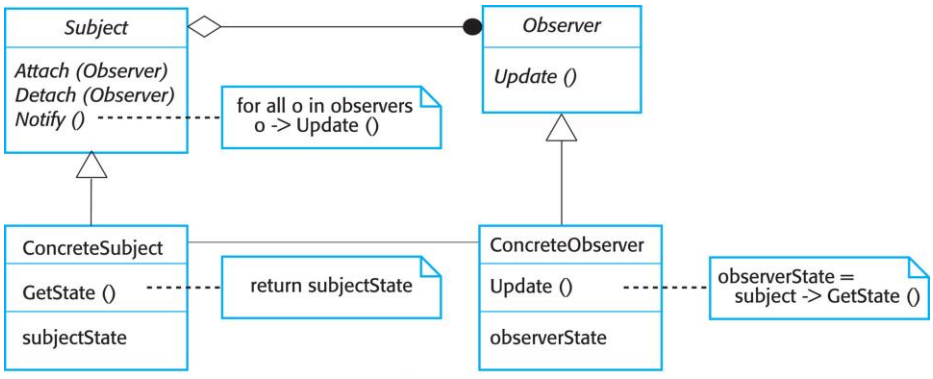
Description	Sépare l'affichage de l'état d'un objet de l'objet lui-même et permet de fournir d'autres affichages. Lorsque il y a des changements d'état de l'objet, tous les affichages sont automatiquement informés et mis à jour pour refléter le changement.
Description du Problème	Dans nombreuses situations, vous devez fournir plusieurs affichages d'informations d'état, tels qu'un affichage graphique et un affichage sous forme de tableau. Tous ne peuvent pas être connus lorsque l'information est spécifiée. Toutes les présentations alternatives doivent prendre en charge l'interaction et, lorsque l'état est modifié, tous les affichages doivent être mis à jour. Ce modèle peut être utilisé dans toutes les situations où plus d'un format d'affichage, pour les informations d'état, est requis et où il n'est pas nécessaire que l'objet qui conserve les informations d'état connaisse les formats d'affichage spécifiques utilisés.
Description de la solution	Cela implique deux objets abstraits, Subject et Observer, et deux objets concrets, ConcreteSubject et ConcreteObject, qui héritent des attributs des objets abstraits associés. Les objets abstraits incluent les opérations générales applicables dans toutes les situations. L'état à afficher est maintenu dans ConcreteSubject, qui hérite des opérations de Subject lui permettant d'ajouter et de supprimer des Observers (chaque observateur correspond à un affichage) et d'émettre une notification lorsque l'état a changé. ConcreteObserver conserve une copie de l'état de ConcreteSubject et implémente l'interface Update () d'Observer qui permet de conserver ces copies à l'étape. ConcreteObserver affiche automatiquement l'état et reflète les modifications chaque fois que l'état est mis à jour.
Conséquences	Le Subject ne connaît que l'Observer abstrait et ne connaît pas les détails de la classe concrète. Par conséquent, il existe un couplage

minimal entre ces objets. En raison de ce manque de connaissances, les optimisations qui améliorent les performances d'affichage sont peu pratiques. Les modifications apportées au Subject peuvent entraîner la génération d'un ensemble de mises à jour liées aux Observers, dont certaines peuvent ne pas être nécessaires.

Multiples Affichages Utilisant le Patron Observateur



Un modèle UML du modèle Observateur



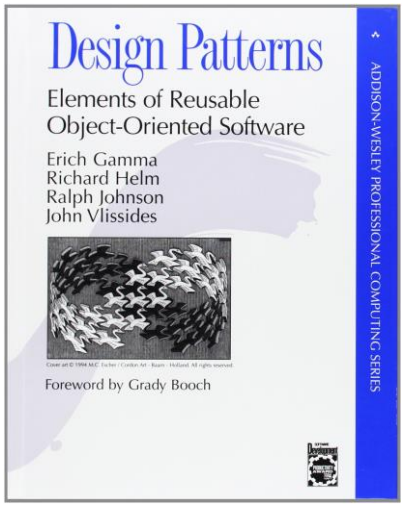
Les problèmes de conception

✧Pour utiliser des patrons dans votre conception, vous devez reconnaître que tout problème de conception vous rencontrez peut avoir un patron associé qui peut être appliqué.

- Dire à plusieurs objets que l'état d'un autre objet a changé (patron Observateur).
- Ranger les interfaces vers un certain nombre d'objets connexes qui ont souvent été développés de manière incrémentale (patron Façade).
- Fournir un moyen standard d'accéder aux éléments d'une collection, quelle que soit la manière dont cette collection est implémentée (patron Iterator).
- Permettre la possibilité d'étendre la fonctionnalité d'une classe existante au moment de

l'exécution (patron Decorator).

Reference sur les patrons de conception



Gamma, Helm, Johnson, Vlissides
Design Patterns : Elements of Reusable Object-Oriented Software.
Addison-Wesley, 1995.

auteurs AKA "*Gang of Four*" (GoF)

Classification des patrons de conception

✧GoF ont explicité trois grandes classes de patrons dans leur livre, chacune spécialisée dans :

- Création d'objets (creational patterns)
- Structure des relations entre objets (structural patterns)
- Comportement des objets (behavioral patterns)

✧23 patrons ont été définis dans ces classes (le détail sera donné dans le 2S, Module: Spécification et Conception des logiciels)

4. Issues de l'Implémentation

Problèmes de l'Implémentation

✧La focalisation ici n'est pas sur la programmation, même si c'est évidemment important, mais sur d'autres issues d'implémentation qui sont souvent non couverts dans les textes de programmation:

- **Réutilisation**: la plupart des logiciels modernes est construit en réutilisant des composants ou des systèmes existants. Lorsque vous développez des logiciels, vous devez utiliser autant que possible de code existant.
- **Gestion de la configuration**: Au cours du processus de développement, vous devez garder la trace des nombreuses versions différentes de chaque composant logiciel dans un système

de gestion de configuration.

- **Développement hôte-cible**: Logiciel de production n'exécute pas habituellement sur le même ordinateur de développement de logiciel. Plutôt, vous développez sur un ordinateur (le système hôte) et de l'exécuter sur un ordinateur séparé (le système cible).

Réutilisation (reuse)

✧Des années 1960 aux années 1990, la plupart des nouveaux logiciels ont été développé à partir de zéro, en écrivant tout le code dans un langage de programmation de haut niveau.

- La seule réutilisation importante des logiciels était la réutilisation des fonctions et des objets dans les bibliothèques des langages de programmation.

✧Les coûts et la pression du calendrier signifient que cette approche devient de plus en plus non viable, en particulier pour les systèmes commerciaux et basés sur Internet.

✧Une approche du développement fondée autour de la réutilisation des logiciels existants a émergé et est maintenant généralement utilisé pour les entreprises et logiciels scientifiques.

Niveaux de réutilisation

✧Le niveau d'abstraction

- A ce niveau, vous ne réutiliser pas le logiciel directement, mais utiliser les connaissances des abstractions utiles dans la conception de votre logiciel.

✧Le niveau d'objet

- A ce niveau, vous réutilisez directement des objets à partir d'une bibliothèque plutôt que d'écrire vous-même le code.

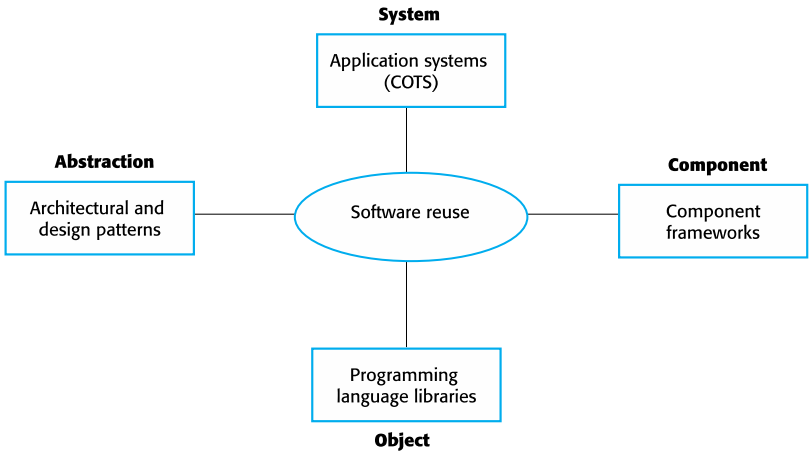
✧Le niveau des composants

- Les composants sont des collections d'objets et classes d'objets que vous réutilisez dans les systèmes d'application.

✧Le niveau de système

- À ce niveau, vous réutilisez des systèmes d'application entiers.

Réutilisation de logiciels



Coûts de réutilisation

✧Les coûts du temps passé à chercher des logiciels à réutiliser et à évaluer s'ils répondent ou non à vos besoins.

✧Le cas échéant, les coûts d'achat du logiciel réutilisable. Pour les grands systèmes prêts à l'emploi, ces coûts peuvent être très élevés.

✧Les coûts d'adaptation et de configuration des composants ou des systèmes logiciels réutilisables pour refléter les exigences du système que vous développez.

✧Les coûts d'intégration des éléments logiciels réutilisables entre eux (si vous utilisez des logiciels de différentes sources) et avec le nouveau code que vous avez développé.

Gestion de la configuration

✧La gestion de la configuration est le nom donné au processus général de gestion d'un système logiciel en évolution.

✧Le but de la gestion de la configuration est de soutenir le processus d'intégration du système afin que tous les développeurs puissent accéder au code et aux documents du projet de manière contrôlée, découvrir les modifications apportées, et compiler et lier les composants pour créer un système.

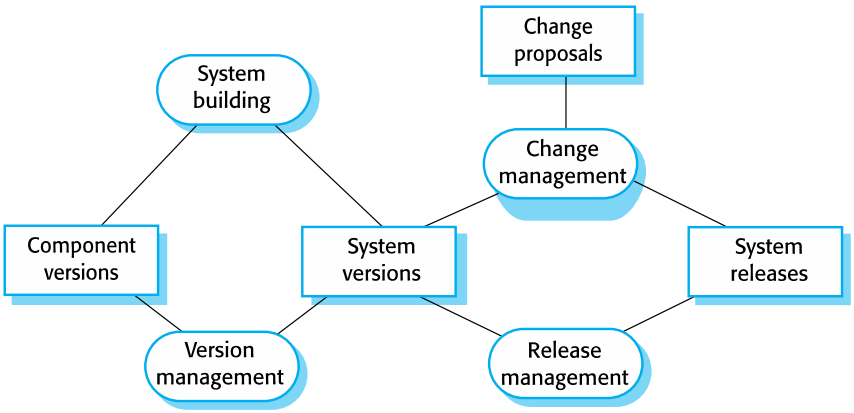
Activités de gestion de configuration

✧Gestion des versions, où un support est fourni pour suivre les différentes versions des composants logiciels. Les systèmes de gestion de versions incluent des fonctionnalités permettant de coordonner le développement de plusieurs programmeurs.

✧Intégration du système, où la prise en charge est fournie pour aider les développeurs à définir les versions de composants utilisées pour créer chaque version d'un système. Cette description est ensuite utilisée pour construire automatiquement un système en compilant et reliant les composants requis.

✧Suivi des problèmes, où le soutien est fourni pour permettre aux utilisateurs de signaler les bugs et autres problèmes, et de permettre à tous les développeurs de voir qui travaille sur ces problèmes et quand ils sont fixés.

Interaction d'outil de gestion de configuration



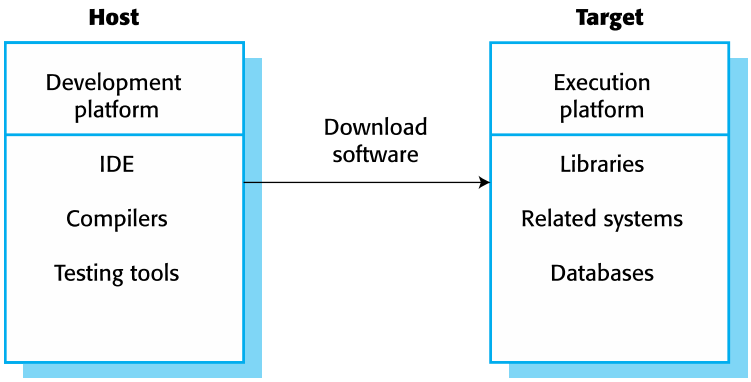
Développement Hôte-Cible

✧La plupart des logiciels sont développés sur un ordinateur (l'hôte),< mais fonctionnent sur une machine séparée (la cible).

✧Plus généralement, on peut parler d'une plate-forme de développement et d'une plate-forme d'exécution.

- Une plate-forme est plus que du matériel.
- Il comprend le système d'exploitation installé ainsi que d'autres logiciels de soutien tels qu'un système de gestion de base de données ou, pour les plateformes de développement, un environnement de développement interactif.

✧Plate-forme de développement a généralement différents logiciels installés que la plate-forme d'exécution; ces plates-formes peuvent avoir des architectures différentes.



Outils de la plate-forme de développement

✧Un compilateur intégré et un système d'édition orienté vers la syntaxe qui vous permet de créer, d'éditer et de compiler du code.

✧Un système de débogage linguistique.

✧Des outils d'édition graphiques, tels que des outils pour éditer des modèles UML.

✧Des outils de test, tels que Junit, qui peuvent exécuter automatiquement un ensemble de tests sur une nouvelle version d'un programme.

✧Des outils de support de projet qui vous aident à organiser le code pour différents projets de développement.

Environnements de développement intégrés (IDE)

✧Les outils de développement logiciel sont souvent regroupés pour créer un environnement de développement intégré (IDE).

✧Un IDE est un ensemble d'outils logiciels qui prennent en charge différents aspects du développement logiciel, dans un cadre commun et une interface utilisateur.

✧Les IDE sont créés pour prendre en charge le développement dans un langage de programmation

spécifique tel que Java. L'IDE du langage peut être développé spécialement, ou peut être une instanciation d'un IDE à usage général, avec des outils de support de langage spécifiques.

Facteurs de déploiement de composants/systèmes

✧Si un composant est conçu pour une architecture matérielle spécifique ou repose sur un autre système logiciel, il doit évidemment être déployé sur une plate-forme fournissant le support matériel et logiciel requis.

✧Les systèmes à haute disponibilité peuvent nécessiter le déploiement de composants sur plusieurs plates-formes. Cela signifie que, en cas de défaillance de la plate-forme, une implémentation alternative du composant est disponible.

✧Si le trafic de communication entre les composants est important, il est généralement judicieux de les déployer sur la même plate-forme ou sur des plates-formes physiquement proches les unes des autres. Cela réduit le délai entre l'envoi d'un message par un composant et sa réception par un autre.

5. Développement open source

✧Le développement open source est une approche du développement de logiciels dans laquelle le code source d'un système logiciel est publié et les volontaires sont invités à participer au processus de développement.

✧Ses racines sont dans la Free Software Foundation (www.fsf.org), qui préconise que le code source ne devrait pas être propriétaire mais plutôt être toujours disponible pour les utilisateurs à examiner et modifier comme ils le souhaitent.

✧Les logiciels libres ont étendu cette idée en utilisant Internet pour recruter une plus grande population de développeurs bénévoles. Beaucoup d'entre eux sont aussi des utilisateurs du code.

Systèmes Open source

✧Le produit open source le plus connu est, bien sûr, le système d'exploitation Linux qui est largement utilisé comme système serveur et, de plus en plus, comme environnement de bureau.

✧D'autres produits open source importants sont Java, le serveur web Apache et le système de gestion de base de données mySQL.

Problémes d'Open source

✧Le produit en cours de développement doit-il utiliser des composants open source?

✧Une approche open source devrait-elle être utilisée pour le développement du logiciel?

Métier open source

✧De plus en plus de sociétés de produits utilisent une approche open source pour le développement.

✧Leur modèle métier ne repose pas sur la vente d'un produit logiciel, mais sur la vente de support pour ce produit.

✧Ils croient que l'implication de la communauté open source permettra de développer des logiciels à moindre coût, plus rapidement et créera une communauté d'utilisateurs pour le logiciel.

Licence Open Source

✧Un principe fondamental du développement open-source est que le code source doit être disponible gratuitement, cela ne signifie pas que n'importe qui peut faire ce qu'il veut avec ce code.

- Légalement, le développeur du code (soit une entreprise ou un individu) possède toujours le code. Ils peuvent imposer des restrictions sur la façon dont il est utilisé en incluant des conditions juridiquement contraignantes dans une licence logicielle open source.
- Certains développeurs open source pensent que si un composant open source est utilisé pour développer un nouveau système, alors ce système devrait également être open source.
- D'autres sont prêts à autoriser l'utilisation de leur code sans cette restriction. Les systèmes développés peuvent être propriétaires et vendus comme des systèmes à source fermée.

Modèles de License

✧La licence publique générale GNU (GPL). Il s'agit d'une licence dite «réciproque» qui signifie que si vous utilisez un logiciel open source sous licence GPL, vous devez rendre ce logiciel open source.

✧La licence publique générale limitée (LGPL) de GNU est une variante de la licence GPL dans laquelle vous pouvez écrire des composants liés au code source libre sans avoir à publier la source de ces composants.

✧La licence Berkley Standard Distribution (BSD). Il s'agit d'une licence non réciproque, ce qui signifie que vous n'êtes pas obligé de republier les modifications ou modifications apportées au code source ouvert. Vous pouvez inclure le code dans les systèmes propriétaires vendus.

Gestion des licences

✧Établir un système de gestion des informations sur les composants open-source téléchargés et utilisés.

✧Tenez-vous au courant des différents types de licences et comprenez comment un composant est sous licence avant d'être utilisé.

✧Soyez conscient des voies d'évolution des composants.

✧Éduquer les gens sur l'open source.

✧Avoir des systèmes d'audit en place.

✧Participez à la communauté open source.

6. Points clés

- ✓ La conception et la mise en œuvre de logiciels sont des activités entrelacées. Le niveau de détail de la conception dépend du type de système et si vous utilisez une approche planifiée ou agile.
- ✓ Le processus de conception orientée objet comprend des activités permettant de concevoir l'architecture du système, d'identifier les objets dans le système, de décrire la conception à l'aide de différents modèles d'objets et de documenter les interfaces des composants.

- ✓ Une gamme de modèles différents peut être produite au cours d'un processus de conception orienté objet. Ceux-ci incluent des modèles statiques (modèles de classe, modèles de généralisation, modèles d'association) et des modèles dynamiques (modèles de séquence, modèles de machine d'état).
- ✓ Les interfaces de composants doivent être définies précisément pour que d'autres objets puissent les utiliser. Un stéréotype d'interface UML peut être utilisé pour définir des interfaces.
- ✓ Lorsque vous développez un logiciel, vous devez toujours envisager la possibilité de réutiliser les logiciels existants, que ce soit en tant que composants, services ou systèmes complets.
- ✓ La gestion de la configuration est le processus de gestion des modifications apportées à un système logiciel en évolution. Il est essentiel lorsqu'une équipe de personnes coopère pour développer des logiciels.
- ✓ La plupart des développements de logiciels sont des développements de cibles d'hôtes. Vous utilisez un IDE sur une machine hôte pour développer le logiciel, qui est transféré à une machine cible pour exécution.
- ✓ Le développement Open Source consiste à rendre le code source d'un système accessible au public. Cela signifie que de nombreuses personnes peuvent proposer des modifications et des améliorations au logiciel.

7. Exercices

I) Quiz :

Choisir la bonne réponse :

1. Choisissez l'instruction incorrecte en termes d'objets.

   a) Les objets sont des abstractions du monde réel
   b) Les objets ne peuvent pas gérer eux-mêmes
   c) Les objets encapsulent les informations d'état et de représentation
   d) Tous les mentionnés

2. Qu'est-ce qui encapsule les fonctions de manipulation de données et de données?

   a) Objet
   b) Classe
   c) Super classe
   d) Sous-classe

3. Lequel des mécanismes suivants permet à plusieurs objets d'une hiérarchie de classes d'avoir des méthodes différentes portant le même nom?

   a) Agrégation
   b) Polymorphisme
   c) Héritage
   d) Tous les mentionnés

4. Les classes d'objets héritées sont autonomes.

   a) Vrai
   b) Faux

5. Lequel des points suivants liés au développement orienté objet (OOD) est vrai?

    a) OOA est concerné par le développement d'un modèle objet du domaine d'application
    b) OOD est concerné par le développement d'un modèle de système orienté objet pour mettre en œuvre les exigences.
    c) a et b
    d) Aucun des mentionnés
6. Comment la généralisation est-elle implémentée dans les langages de programmation orientés objet?

    a) Héritage
    b) Polymorphisme
    c) Encapsulation
    d) Les classes abstraites

7. Lequel des éléments suivants est un désavantage de l'OOD?

    a) Une maintenance plus facile
    b) Les objets peuvent être compris comme des entités autonomes
    c) Les objets sont des composants potentiellement réutilisables
    d) Aucun des mentionnés

8. L'objet qui recueille des données sur la demande plutôt que de manière autonome est connu comme :

    a) Objet actif
    b) Objet passif
    c) Plusieurs instances
    d) Aucun des mentionnés

9. Les objets sont exécutés

    a) séquentiellement
    b) en parallèle
    c) séquentiellement et parallèle
    d) aucun des mentionnés

10. Laquelle des méthodes OOD suivantes incorpore à la fois un «processus de développement micro» et un «processus de développement macro»?

    a) Méthode Booch
    b) Méthode de Rumbaugh
    c) Méthode de Wirfs-Brock
    d) Méthode Coad et Yourdon

11. Grady Booch, James Rumbaugh et Ivar Jacobson ont combiné les meilleures caractéristiques de leur analyse orientée objet individuelle dans une nouvelle méthode pour la conception orientée objet connue sous le nom de

    a) HTML

b) XML
c) UML
d) SGML

12. Lequel des modèles suivants est un modèle dynamique qui montre comment le système interagit avec son environnement tel qu'il est utilisé?

a) le modèle de contexte du système
b) le modèle d'interaction
c) modèle environnemental
d) à la fois le contexte du système et l'interaction

13. Lequel des modèles suivants est un modèle structurel qui illustre les autres systèmes dans l'environnement du système en cours de développement?

a) le modèle de contexte du système
b) le modèle d'interaction
c) modèle environnemental
d) à la fois le contexte du système et l'interaction

14. Quel modèle montre le flux des interactions d'objet?

a) Modèle de séquence
b) Modèle de sous-système
c) Modèle dynamique
d) Modèle séquentiel et dynamique

15. In chapter 7, if the system state is Shutdown then it can respond to which of the following message?

a) restart()
b) reconfigure()
c) powerSave()
d) all of the mentioned

16. In chapter 7, which message is received so that the system moves to the Testing state, then the Transmitting state, before returning to the Running state?

a) signalStatus()
b) remoteControl()
c) reconfigure()
d) reportStatus()

17. Which mechanism is applied to use a design pattern in an OO system?

a) Inheritance
b) Composition
c) All of the mentioned
d) None of the mentioned

18. Design patterns do not follow the concept of software reuse.

a) True
b) False

19. Le développement open source consiste à rendre le code source d'un système accessible au public.

a) Vrai
b) Faux

II) Exercices: Conception avec UML

Exercice 1 :

a) À l'aide de la notation structurée illustrée dans le chapitre 7 (slide #18), spécifiez les cas d'utilisation de la station météo « **Report Status** » et « **Reconfigure** ». Vous devez faire des hypothèses raisonnables sur la fonctionnalité requise ici.

b) Supposons que le système Mentcare soit développé en utilisant une approche orientée objet. Dessinez un diagramme de cas d'utilisation indiquant au moins six cas d'utilisation possibles pour ce système.

Exercice 2 :

En utilisant la notation graphique UML pour les classes d'objets, concevoir les classes d'objets suivantes, identifier les attributs et les opérations. Utilisez votre propre expérience pour décider sur les attributs et les opérations qui doivent être associés à ces classes d'objets.

- Un système de messagerie sur un téléphone mobile ou une tablette
- Une imprimante pour un ordinateur personnel
- Un système stéréo personnel
- Un compte bancaire
- Un catalogue de bibliothèque

Exercice 3 :

En vue de développer une conception orientée objet pour le système suivant:

« Une station de remplissage (station d'essence, gas station) doit être mise en place pour un fonctionnement entièrement automatisé. Les conducteurs glissent leur carte de crédit via un lecteur connecté à la pompe; la carte est contrôlée par la communication avec un ordinateur de société de crédit (par exemple, une banque), et une limite de carburant est établie. Le conducteur peut alors prendre le carburant nécessaire. Lorsque la livraison de carburant est terminée et le tuyau de la pompe est retourné à son étui, la carte de crédit sur le compte du conducteur est débitée du coût du carburant pris. La carte de crédit est retournée après le débit. Si la carte est invalide, la pompe la retourne sans distribuer le carburant».

1. Définir le contexte et les modes d'utilisation du système : proposer un diagramme de cas d'utilisation couvrant une des exigences du système (en utilisant la structure décrite dans le chapitre precedent).
2. Identifier les classes d'objets possibles dans ce système.

Exercice 4 :

Soit le système suivant :

« Un système d'agenda de groupe et de gestion du temps (group diary and time management system) est destiné à soutenir le calendrier des réunions et des rendez-vous dans un groupe de collègues. Lorsque un rendez-vous doit être fait et qui implique un certain nombre de personnes, le système trouve un emplacement commun dans chacun de leurs agendas et organise le rendez-vous pour ce moment. Si aucun emplacement commun n'est disponible, il interagit avec l'utilisateur pour réorganiser son agenda personnel pour faire place au rendez-vous.»

1. Donner un diagramme de séquence montrant les interactions d'objets dans ce système d'agenda de groupe quand un groupe de personnes se sont planifiés pour une rencontre.
2. Proposer un diagramme UML d'état montrant les changements d'état possibles pour l'agenda de groupe.

8. Solutions

I) Quiz :

1 : b, Les objets sont indépendants.
2 : a,
3 : b, Dans le polymorphisme, les instances de chaque sous-classe seront libres de répondre aux messages en appelant leur propre version de la méthode.
4 : b, Les classes d'objets héritées ne sont pas autonomes. Ils ne peuvent être compris sans référence à leurs super-classes.
5 : c, La réponse est en faveur de l'OOD.
6 : a,
7 : d, Toutes les options définissent les caractéristiques de OOD.
8 : b, Un objet passif contient des données mais n'initie pas de contrôle.
9 : c, Les objets peuvent être distribués et peuvent s'exécuter séquentiellement ou en parallèle.
10 : a, Le processus de développement de macros inclut le processus de planification architecturale et de développement micro qui définit les règles qui régissent l'utilisation des opérations et des attributs ainsi que les stratégies spécifiques au domaine pour la gestion de la mémoire, la gestion des erreurs et autres fonctions d'infrastructure.
11 : c, Le langage de modélisation unifié (UML) est devenu largement utilisé dans l'industrie comme l'approche standard de OOD.
12 : b,
13 : a, Le modèle de contexte d'un système peut être représenté en utilisant des associations. Les associations montrent simplement qu'il existe des relations entre les entités impliquées dans l'association.
14 : a, Les modèles de séquence sont représentés à l'aide d'une séquence UML ou d'un diagramme de collaboration et sont des modèles dynamiques.
15 : d, A restart() message causes a transition to normal operation. Both the powerSave() and reconfigure() messages cause a transition to a state in which the system reconfigures itself.
16: d,
17: c, Using inheritance, an existing design pattern becomes a template for a new subclass.Composition is a concept that leads to aggregate objects.
18 : b, Design patterns allow the designer to create the system architecture by integrating reusable components.

19 : a, Cela signifie que de nombreuses personnes peuvent proposer des modifications et des améliorations au logiciel.

II) Exercices

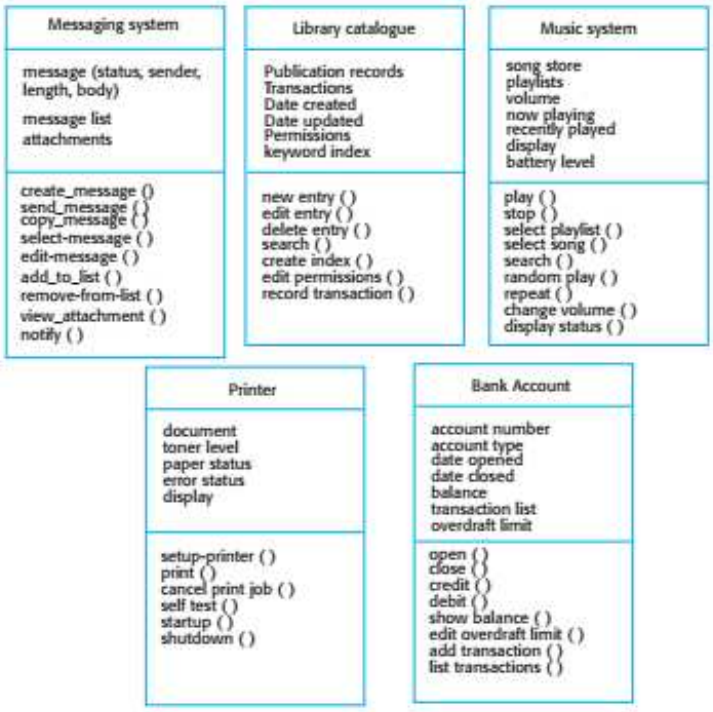
Exercice 1 :

a) **System**: Weather station
Use case: Report status
Actors: Weather information system
Data: The weather station sends a status update to the weather information system giving information about the status of its instruments, computers and power supply.
Stimulus: The weather information system establishes a satellite link with the weather station and requests status information.
Response: A status summary is uploaded to the weather information system.
Comments: System status is usually requested at the same time as the weather report.

System: Weather station
Use case: Reconfigure
Actors: Weather information system
Data: The weather information station sends a reconfiguration command to the weather station. This places it into remote control mode where further commands may be sent from the remote system to update the weather station software.
Stimulus: A command from the weather information system.
Response: Confirmation that the system is in remote control mode
Comments: Used occasionally when software updates have to be installed.

Exercice 2 :

Il existe de nombreux modèles possibles ici et beaucoup de complexité peuvent être ajoutés aux objets. Cependant, on présente ci-dessous que des objets simples qui encapsulent les principales exigences de ces artefacts. Les conceptions possibles de ces objets sont indiquées dans le diagramme suivant :



Exercice 3 :

1. Généralement ce type de systèmes n'a qu'une seule exigence fonctionnelle à faire : **Distribuer le carburant** dont le concepteur de ce système devrait assurer la sécurité des cartes de crédit par l'exigence d'introduire le code PIN.

   - **Cas d'utilisation** : Distribuer de carburant

     Acteurs: Conducteur, Pompe, Société de crédit (par exemple, une banque)

     Entrées: Carte de Crédit, PIN, Détails du compte Bancaire.

     Sorties: Carte de Crédit, Détails du Compte Bancaire du conducteur, Reçu

     Fonctionnement normal:

     Le conducteur glisse sa carte de crédit via un lecteur connecté à la pompe en introduisant un code PIN par un clavier sur la pompe; la carte est contrôlée premièrement pour vérifier sa **validité**. Ensuite, le conducteur est invité à saisir la quantité voulue du carburant, et la carte est contrôlée encore une fois, par la communication avec un ordinateur de société de crédit (par exemple, une banque), pour vérifie **s'il y a assez de solde** sur la compte bancaire du conducteur. Le conducteur peut alors prendre le carburant demandé. Lorsque la livraison de carburant est terminée et le conducteur retourne le tuyau de la pompe à son étui et la carte de crédit sur le compte du conducteur est débitée du coût du carburant pris. La carte de crédit est retournée par la pompe après le débit au conducteur avec un reçu.
     Exception:

- ***PIN incorrect*** : Le conducteur est demandé à recomposer la clé PIN. S'il est incorrect après trois tentatives, la carte est conservée par la pompe et le conducteur est invité à demander conseils.
- ***Carte invalide*** : Carte est retournée par la pompe avec un message d'invalidité.
- ***Solde insuffisant-Transaction terminée***. Carte retournée au conducteur

2. **Les classes d'objets possibles sont:**

Compte Bancaire
Numéro de compte Type de compte Libellé (nom et prénom) Adresse Date d'ouverture Date de Fermeture Solde Liste des Transactions
Ouvrir_compte () Fermer_compte () Créditer () Débiter () Afficher_solde () Ajouter_transaction () Afficher_transactions()

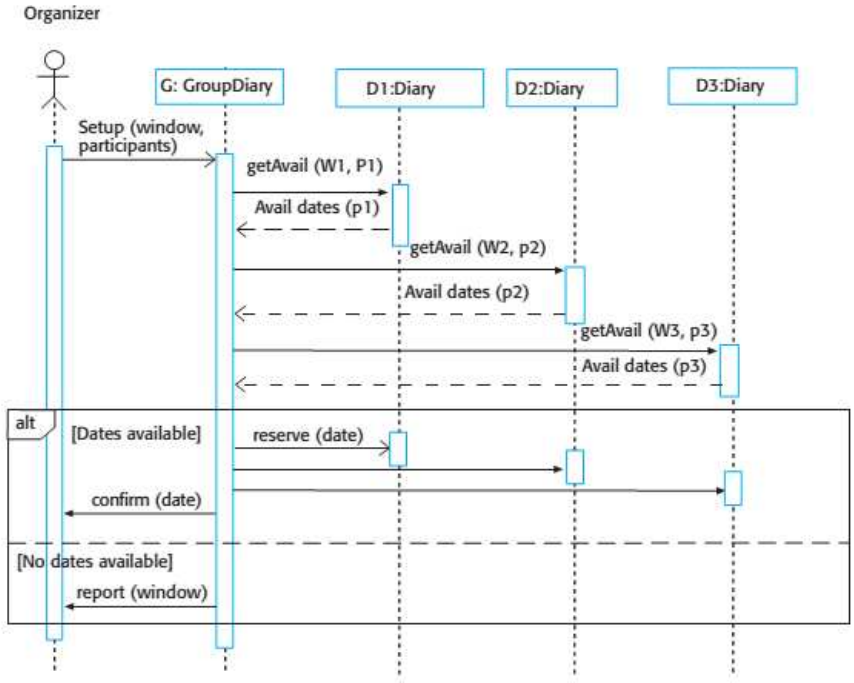
Carte de Crédit
Numéro de la carte PIN Date de Validation Limite_retrait
Modifier_limite _retrait () Changer_PIN ()

Pompe d'Essence
Numero Ville Quantité_Carburant_Totale Quantité_Carburant_Pris Limite_retrait Montant
Approvisionner_Carburant Distribuer_Carburant () Afficher_Carte_Invalide () Conserver_Carte () Calculer_Montant_CarburantPris () Afficher_Solde_Insuffisant ()

Les associations qui peuvent exister entre ces classes sont:

Associer : entre « Compte Bancaire » et « Carte de Crédit »
Verifier_Solde : entre « Pompe d'Essence » et « Compte Bancaire »

Exercice 4 :



Le diagramme ci-dessus suppose qu'il y a trois participants à la réunion, dont l'un est l'organisateur de la réunion. L'organisateur suggère un «créneau» (window en anglais) où la réunion doit avoir lieu et les participants seront impliqués. L'agenda de groupe communique avec les agendas des participants, à leur tour, de modifier en conséquence le créneau selon la disponibilité. Donc, si l'organisateur suggère un créneau du 18-19 Juin, l'agenda du groupe consulte l'agenda de l'organisateur (D1) pour trouver une disponibilité sur ces jours. D2 est ensuite mis en contact avec une telle disponibilité.

S'il n'y a pas de dates mutuellement disponibles pour ce créneau, le système signale cela à l'organisateur. Sinon, une date est sélectionné, entrée dans tous les agendas et confirmée à l'organisateur.

Chapitre VIII

Test du Logiciel

Objectifs

- comprendre les étapes du test depuis les tests pendant le développement jusqu'aux tests d'acceptation par les clients du système;
- ont été introduites à des techniques qui vous aident à choisir les cas de test visant à détecter les défauts du programme;
- comprendre le développement test-first, où vous concevez des tests avant d'écrire du code et exécutez ces tests automatiquement;
- connaître les différences importantes entre les tests de composants, de systèmes et de versions, et être au courant des processus et des techniques de test des utilisateurs.

Themes couverts

- Test de développement
- Développement piloté par les tests
- Test de sortie «Release »
- Test d'utilisateur

Chapitre 8:

Test du Logiciel

1. Introduction

✧Le test est une activité importante dont le but est d'arriver à un produit « zéro défaut ».

✧C'est la limite idéaliste vers laquelle on tend pour la qualité du logiciel.

✧Généralement 40% du budget global est consacrée à l'effort de test.

Quelques Bugs connus

✧Spatial - Juin 1996

En 4 juin 1996, la fusée Ariane 5 s'explose après 37s de vol :

- Erreur de conversion entre données numériques!
- **$370 Millions de dégâts!**

✧Transport – septembre 1998

- La ville de Dublin connaît un embouteillage monstrueux parce que la mise à jour du système de feux tricolores a abouti à la déconnexion de 140 carrefours.

✧Défense – mars 2002

- 3 soldats américains sont tués et 20 autres sont grièvement blessés parce qu'un tir a été mal positionné par un système d'arme

✧**Médical - avril 2008**

- Au Royaume Unis, le National Program for IT (NPfIT) révèle que, depuis 2005, environ 300 incidents ont mis des patients en danger lors d'utilisation d'appareils médicaux informatisés.

✧ **Banque - juillet 2009**

- Démantèlement d'un réseau international de fraude à la carte bancaire ayant réalisé plus de 35.000 transactions frauduleuses d'un montant total d'environ
  6,5 millions d'euros.

✧ **Informatique – il n'y a pas très longtemps**

- Le PC que l'on doit rebouter parce qu'une application l'a bloqué ou que le système d'exploitation a un bug.

✧ **Télécommunications – hier ou avant-hier**

- L'appel que l'on doit recommencer parce que la conversation a été interrompue inopinément

Définition du test

✧« Program testing can be used to show the presence of bugs, but never to show their absence! » Edsger Wybe Dijkstra

✧Selon l'IEEE: « Le test est l'exécution ou l'évaluation d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus »

✧Le test est une recherche d'anomalie dans le comportement de logiciel. C'est une activité paradoxale: il vaut mieux que ce ne soit pas la même personne qui développe et qui teste le soft. D'où le fait qu'un bon test est celui qui met à jour une erreur (non encore rencontrée).

✧Il faut arriver à gérer une suite de test la plus complète possible à un coup minimal.

Qu'est-ce qu'un test réussi ?

✧ « Un test réussi n'est pas un test qui n'a pas trouvé de défauts, mais un test qui a effectivement trouvé un défaut (ou une anomalie) » G.J. Myers

Erreur, défaut, anomalie

✧ Une anomalie (ou défaillance) est un comportement observé différent du comportement attendu ou spécifié. Exemple. Le 4 juin1996, on a constaté…

- On constate une anomalie
- Due à un défaut du logiciel
- Le défaut est causé généralement par une faute ou erreur (ex. erreur de programmation : confusion d'un « I » avec un « 1 »)
- Chaîne de causalité : erreur => défaut => anomalie
- Nature de l'erreur: spécification, conception, programmation

Tests de programme

✧Les tests visent à montrer qu'un programme fait ce qu'il est censé faire et à découvrir les défauts du programme avant qu'il ne soit mis en service.

✧Lorsque vous testez un logiciel, vous exécutez un programme en utilisant des données artificielles.

✧Vous vérifiez les résultats de l'analyse pour déceler des erreurs, des anomalies ou des informations sur les attributs non fonctionnels du programme.

✧Peut révéler la présence d'erreurs PAS leur Absence.

✧Les tests font partie d'un processus de vérification et de validation plus général, qui comprend également des techniques de validation statique.

Objectifs du test de programme

✧Démontrer au développeur et au client que le logiciel répond à ses exigences.

- Pour les logiciels personnalisés, cela signifie qu'il doit y avoir au moins un test pour chaque exigence dans le document d'exigences. Pour les produits logiciels génériques, cela signifie qu'il doit y avoir des tests pour toutes les fonctionnalités du système, ainsi que des combinaisons de ces fonctionnalités, qui seront incorporées dans la version du produit.

✧Découvrir des situations dans lesquelles le comportement du logiciel est incorrect, indésirable ou non conforme à sa spécification.

- Le test des défauts vise à éliminer les comportements indésirables du système tels que les pannes système, les interactions indésirables avec d'autres systèmes, les calculs incorrects et la corruption de données.

Test de validation et test de défaut

✧Le premier objectif conduit à des tests de validation

- Vous vous attendez à ce que le système fonctionne correctement en utilisant un ensemble donné de scénarios de test qui reflètent l'utilisation prévue du système.

✧Le deuxième objectif conduit à des tests de défauts

- Les cas de test sont conçus pour exposer les défauts. Les cas de test dans les tests de défauts peuvent être délibérément obscurs et n'ont pas besoin de refléter la façon dont le système est normalement utilisé.

Objectifs du processus de test

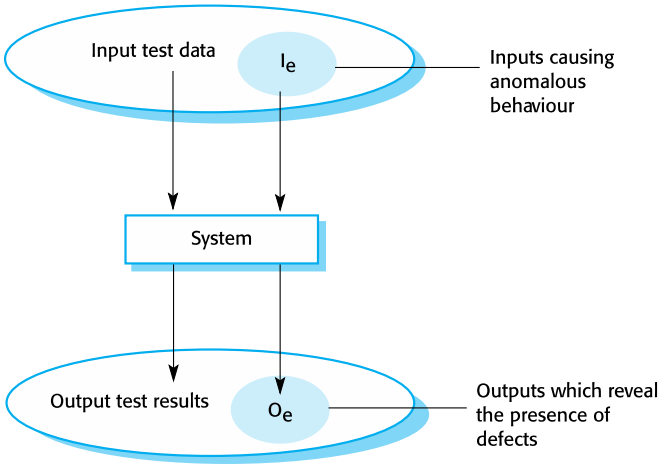
✧**Tests de validation**

- Démontrer au développeur et au client du système que le logiciel répond à ses exigences
- Un test réussi montre que le système fonctionne comme prévu.

✧**Tests de défauts**

- Découvrir les fautes ou les défauts du logiciel lorsque son comportement est incorrect ou non conforme à sa spécification
- Un test réussi est un test qui rend le système incorrect et expose donc un défaut du système.

Un modèle d'entrée-sortie des tests de programme



Vérification vs Validation

✧**Vérification:**

- assurer que le système fonctionne correctement selon la spécification du système.
- "Are we building the product right" (le bon produit).

✧**Validation:**

- Assurer que le système fonctionne selon les besoins de l'utilisateur (ce que l'utilisateur vraiment attend)
- "Are we building the right product". (le produit approprié)
- La validation est le moyen de confirmer le respect d'exigences déterminées pour une utilisation spécifique prévue.
- Validation : Faisons-nous le travail attendu ?

✧Phase de V&V est souvent plus longue que les phases de spécification, conception et implémentation réunies

✧Vérification et validation représente :

- environ 30% du développement d'un logiciel standard
- plus de 50% du développement d'un logiciel critique

La confiance de V & V

✧L'objectif de V & V est d'établir la confiance que le système doit être assez bon pour l'usage prévu.

✧Ceci dépend du but de système, des attentes des utilisateurs et de l'environnement de marketing

- But du logiciel
  - Le niveau de confiance dépend de l'importance du logiciel pour une organisation.
- Les attentes des utilisateurs
  - Les utilisateurs peuvent avoir de faibles attentes vis-à-vis de certains types de logiciels.
- Environnement de marketing
  - Mettre un produit sur le marché tôt peut être plus important que de trouver des défauts dans le programme.

Inspections et tests

✧Les inspections se concentrent principalement sur le code source d'un système, mais toute représentation lisible du logiciel, telle que ses exigences ou un modèle de conception, peut être inspectée.

✧Lorsque vous inspectez un système, vous utilisez la connaissance du système, de son domaine d'application et du langage de programmation ou de modélisation pour détecter les erreurs.

Inspections et tests

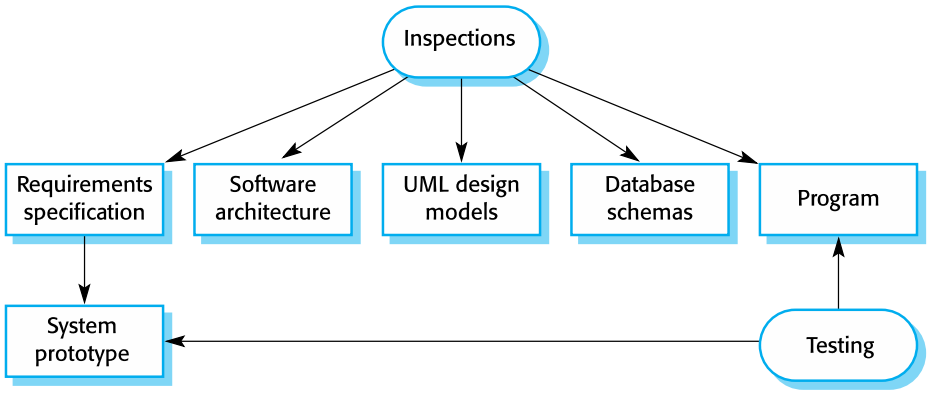
✧Inspections logicielles concernées par l'analyse de la représentation statique du système pour découvrir les problèmes (vérification statique)

- Peut être complété par une analyse documentaire et par code.

✧Tests logiciels préoccupés par l'exercice et l'observation du comportement du produit (vérification dynamique)

- Le système est exécuté avec des données de test et son comportement opérationnel est observé.

Inspections et tests



Inspections de logiciels

✧Celles-ci impliquent des personnes examinant la représentation du système dans le but de découvrir des anomalies et des défauts.

✧Les inspections ne nécessitent pas l'exécution d'un système, elles peuvent donc être utilisées avant l'implémentation.

✧Elles peuvent être appliquées à toute représentation du système (exigences, conception, données de configuration, données de test, etc.).

✧Elles ont été montrées pour être une technique efficace pour découvrir des erreurs de programme.

Avantages des inspections

1. Pendant les tests, les erreurs peuvent masquer (cacher) d'autres erreurs. Lorsqu'une erreur entraîne des sorties inattendues, vous ne pouvez jamais être sûr que les anomalies de sortie

ultérieures sont dues à une nouvelle erreur ou sont des effets secondaires de l'erreur d'origine. Parce que l'inspection est un processus statique, vous n'avez pas à vous soucier des interactions entre les erreurs. Par conséquent, une seule session d'inspection peut détecter de nombreuses erreurs dans un système.

2. Les versions incomplètes d'un système peuvent être inspectées sans coûts supplémentaires. Si un programme est incomplet, vous devez développer des tests spécialisés pour tester les pièces disponibles.
3. Outre la recherche de défauts de programme, une inspection peut également prendre en compte des attributs de qualité plus larges d'un programme, tels que le respect des normes, la portabilité et la maintenabilité. Vous pouvez rechercher des inefficacités, des algorithmes inappropriés et un style de programmation médiocre qui pourraient rendre le système difficile à maintenir et à mettre à jour.

Les inspections et les tests

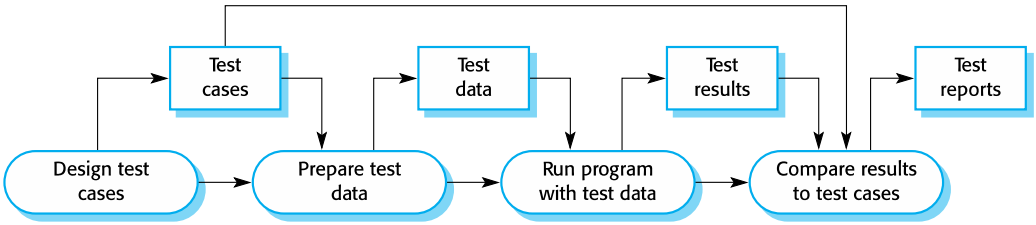
✧Les inspections et les tests sont des techniques de vérification complémentaires et non opposées.

✧Les deux devraient être utilisés pendant le processus de V & V.

✧Les inspections peuvent vérifier la conformité à une spécification mais ne sont pas conformes aux exigences réelles du client.

✧Les inspections ne peuvent pas vérifier certaines caractéristiques non fonctionnelles telles que la performance, la facilité d'utilisation, etc.

Un modèle du processus de test logiciel



Les niveaux de test

✧Tests de développement (Development testing), où le système est testé pendant le développement pour détecter les bogues (bugs) et les défauts.

✧Tests de sortie (Release testing), où une équipe de test distincte teste une version complète du système avant de la distribuer aux utilisateurs.

✧Tests d'utilisateur (User testing), où les utilisateurs potentiels d'un système testent le système dans leur propre environnement.

2. Tests de Développement

✧Les tests de développement incluent toutes les activités de test réalisées par l'équipe développant le système.

- Tests unitaires, où des unités de programme individuelles ou des classes d'objets sont testées. Les tests unitaires doivent se concentrer sur le test de la fonctionnalité des classes d'objets ou des méthodes.
- Tests de composants, où plusieurs unités individuelles sont intégrées pour créer des composants composites. Les tests de composants doivent se concentrer sur le test des interfaces de composants.
- Test du système, où certains ou tous les composants d'un système sont intégrés et le système est testé dans son ensemble. Les tests du système doivent être axés sur le test des interactions entre composants.

Tests unitaires

✧Le test unitaire est le processus de test de composants individuels isolés.

✧C'est un processus de test de défauts.

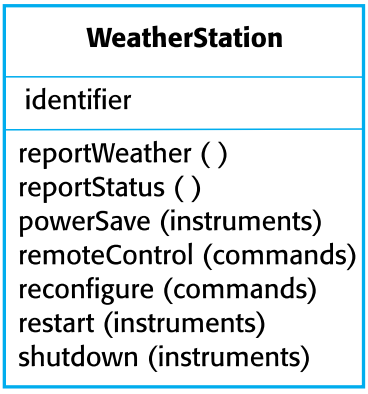
✧Les unités peuvent être:
- Fonctions individuelles ou méthodes au sein d'un objet
- Classes d'objets avec plusieurs attributs et méthodes
- Composants composites avec des interfaces définies utilisées pour accéder à leurs fonctionnalités.

Test de classe d'objet

✧La couverture complète des tests d'une classe implique
- Tester toutes les opérations associées à un objet
- Définition (set) et interrogation (get) de tous les attributs d'objet
- Faire des exercices sur tous les états possibles d'un objet.

✧L'héritage rend plus difficile la conception de tests de classes d'objets car les informations à tester ne sont pas localisées.

La classe d'objet de la station météo



Tests de la station météo

✧Nécessité de définir des cas de test pour « reportWeather », examiner, tester, démarrer et arrêter.

✧En utilisant un modèle d'état, identifier les séquences de transitions d'état à tester et les séquences d'événements à l'origine de ces transitions

✧Par exemple:
- Shutdown -> Running-> Shutdown
- Configuring-> Running-> Testing -> Transmitting -> Running
- Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

Tests automatisés

✧Dans la mesure du possible, les tests unitaires doivent être automatisés pour que les tests soient exécutés et vérifiés sans intervention manuelle.

✧Dans les tests unitaires automatisés, vous utilisez un framework d'automatisation de test (tel que JUnit) pour écrire et exécuter vos tests de programme.

✧Les frameworks de tests unitaires fournissent des classes de test génériques que vous étendez pour créer des cas de test spécifiques. Ils peuvent ensuite exécuter tous les tests que vous avez implémentés et signaler, souvent via une interface graphique, le succès des tests.

Composants du test automatisé

✧Une partie de configuration, dans laquelle vous initialisez le système avec le scénario de test, à savoir les entrées et les sorties attendues.

✧Une partie d'appel, où vous appelez l'objet ou la méthode à tester.

✧Une partie d'assertion, où vous comparez le résultat de l'appel avec le résultat attendu. Si l'assertion est évaluée à Vraie, le test a été un succès, si elle est fausse, il a échoué.

Choisir des cas de test unitaires

✧Les cas de test doivent montrer que, lorsqu'ils sont utilisés comme prévu, le composant que vous testez fait ce qu'il est censé faire.

✧S'il y a des défauts dans le composant, ceux-ci devraient être révélés par des cas de test.

✧Cela conduit à 2 types de cas de test unitaires:
- Le premier devrait refléter le fonctionnement normal d'un programme et devrait montrer que le composant fonctionne comme prévu.
- L'autre type de test devrait être basé sur l'expérience de test où des problèmes communs se posent. Il doit utiliser des entrées anormales pour vérifier que celles-ci sont correctement traitées et ne plantent pas le composant.

Stratégies de test

✧Tests de partition, dans lesquels vous identifiez des groupes des entrées ayant des caractéristiques communes et devant être traités de la même manière.
- Vous devriez choisir des tests à l'intérieur de chacun de ces groupes.

✧Test basé sur des lignes directrices, dans lequel vous utilisez des directives de test pour choisir les cas

de test.

- Ces directives reflètent l'expérience antérieure des types d'erreurs que les programmeurs font souvent lors du développement de composants.

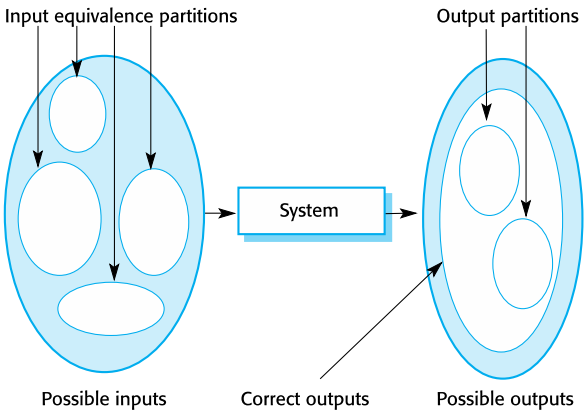
Tests de partition

✧Les données d'entrée et les résultats de sortie tombent souvent dans différentes classes où tous les membres d'une classe sont liés.

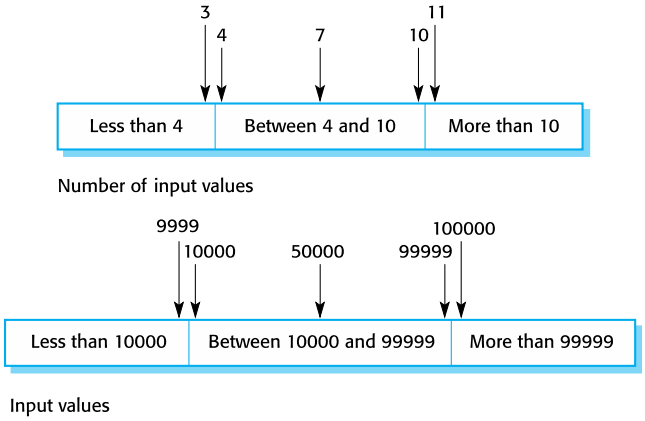
✧Chacune de ces classes est une partition d'équivalence ou un domaine dans lequel le programme se comporte de manière équivalente pour chaque membre de la classe.

✧Les cas de test doivent être choisis dans chaque partition.

Partitionnement d'équivalence



Partitions d'équivalence



Directives de test (séquences)

✧Choisissez des entrées qui forcent le système à générer tous les messages d'erreur

✧Concevoir des entrées qui provoquent le débordement des tampons d'entrée

✧Répétez la même entrée ou série d'entrées plusieurs fois

✧Forcer les sorties invalides à être générées

✧Forcez les résultats du calcul à être trop grand ou trop petit

Directives générales de test

✧Tester un logiciel avec des séquences qui n'ont qu'une seule valeur.

✧Utilisez des séquences de différentes tailles dans différents tests.

✧Dérivez les tests de façon à accéder aux premier, deuxième et dernier éléments de la séquence.

✧Testez avec des séquences de longueur nulle.

Tests de composants

✧Les composants logiciels sont souvent des composants composites constitués de plusieurs objets interagissant.

- Par exemple, dans le système de station météorologique, le composant de reconfiguration inclut des objets qui traitent chaque aspect de la reconfiguration.

✧Vous accédez aux fonctionnalités de ces objets via l'interface de composant définie.

✧Le test des composants composites doit donc se concentrer sur la démonstration que l'interface du composant se comporte conformément à ses spécifications.

- Vous pouvez supposer que les tests unitaires sur les objets individuels dans le composant ont été terminés.

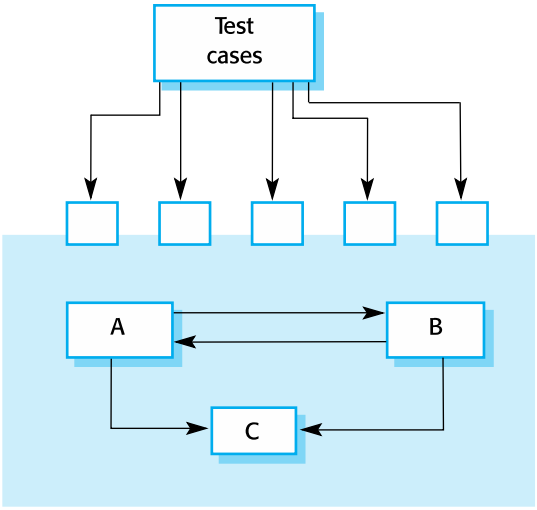
Tests d'interface

✧Les objectifs sont pour détecter les défauts dus à des erreurs d'interface ou à des hypothèses invalides sur les interfaces.

✧Types d'interface

- Interfaces de paramètres : Les données transmises d'une méthode ou d'une procédure à une autre.
- Interfaces de mémoire partagée: Le bloc de mémoire est partagé entre les procédures ou les fonctions.
- Interfaces procédurales: Le sous-système encapsule un ensemble de procédures à appeler par d'autres sous-systèmes.
- Interfaces de transmission de messages: Les sous-systèmes demandent des services à partir d'autres sous-systèmes

Tests d'interface



Erreurs d'interface

✧Mauvaise utilisation de l'interface

- Un composant appelant appelle un autre composant et fait une erreur dans son utilisation de son interface, par ex. paramètres dans le mauvais ordre.

✧Interface malentendu

- Un composant appelant incorpore des hypothèses sur le comportement du composant appelé qui sont incorrectes.

✧Erreurs de synchronisation

- Le composant appelé et le composant appelant fonctionnent à des vitesses différentes et des informations obsolètes sont accessibles.

Directives de test d'interface

✧Concevoir des tests pour que les paramètres d'une procédure appelée se trouvent aux extrémités de leurs plages.

✧Toujours tester les paramètres de pointeur avec des pointeurs NULL.

✧Concevoir des tests qui provoquent l'échec du composant.

✧Utilisez le test de stress dans les systèmes de transmission de messages.

✧Dans les systèmes de mémoire partagée, modifiez l'ordre dans lequel les composants sont activés.

Test du système

✧Le test du système pendant le développement implique l'intégration de composants pour créer une version du système, puis tester le système intégré.

✧L'objectif du test du système est de tester les interactions entre les composants.

✧Les tests du système vérifient que les composants sont compatibles, interagissent correctement et transfèrent les bonnes données au bon moment sur leurs interfaces.

✧Les tests du système testent le comportement émergent d'un système.

Test du système et de composants

✧Au cours des tests du système, les composants réutilisables qui ont été développés séparément et les systèmes prêts à l'emploi peuvent être intégrés avec des composants nouvellement développés. Le système complet est ensuite testé.

✧Les composants développés par différents membres de l'équipe ou sous-équipes peuvent être intégrés à ce stade. Les tests de système sont un processus collectif plutôt qu'un processus individuel.

- Dans certaines entreprises, les tests de système peuvent impliquer une équipe de test distincte sans implication des concepteurs et des programmeurs.

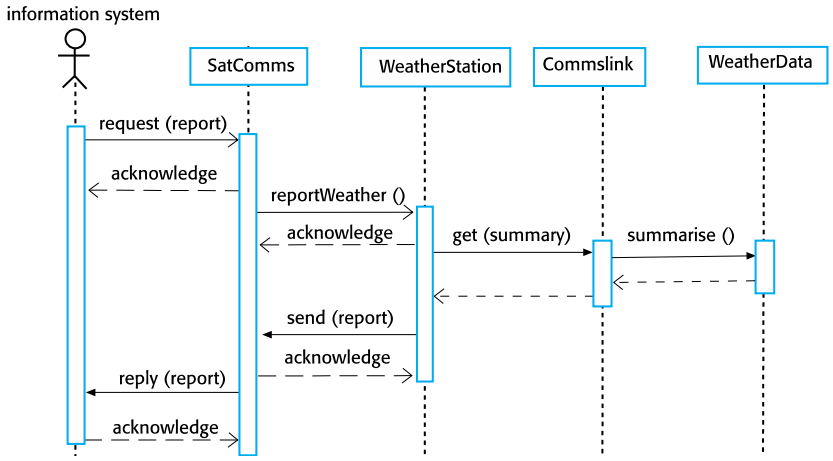
Test de cas d'utilisation

✧Les cas d'utilisation développés pour identifier les interactions du système peuvent être utilisés comme base pour les tests du système.

✧Chaque cas d'utilisation implique généralement plusieurs composants du système, donc tester le cas d'utilisation force ces interactions à se produire.

✧Les diagrammes de séquence associés au cas d'utilisation documentent les composants et les interactions en cours de test.

Diagramme de séquence pour Recueillir les données météorologiques



Cas de test dérivés du diagramme de séquence

✧Une entrée d'une demande de rapport doit avoir un accusé de réception associé. Un rapport devrait finalement être renvoyé de la demande.

- Vous devez créer des données récapitulatives qui peuvent être utilisées pour vérifier que le rapport est correctement organisé.

✧Une requête d'entrée pour un rapport à WeatherStation entraîne la génération d'un rapport

récapitulatif.

- Peut être testé en créant des données brutes correspondant au résumé que vous avez préparé pour le test de SatComms et en vérifiant que l'objet WeatherStation produit correctement ce résumé. Ces données brutes sont également utilisées pour tester l'objet WeatherData.

Politiques de test

✧Un test exhaustif du système est impossible, et donc des politiques de test définissant la couverture de test système requise peuvent être développées.

✧Exemples de politiques de test:

- Toutes les fonctions système accessibles via les menus doivent être testées.
- Les combinaisons de fonctions (par exemple le formatage de texte) auxquelles on accède via le même menu doivent être testées.
- Lorsque l'entrée de l'utilisateur est fournie, toutes les fonctions doivent être testées avec une entrée correcte et incorrecte.

3. Développement piloté par les tests

Développement piloté par les tests

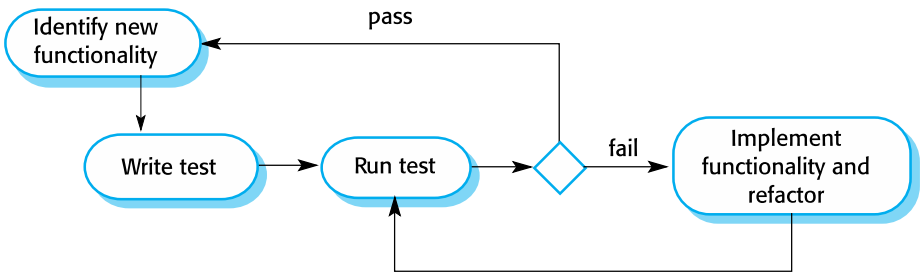
✧Le développement piloté par les tests (TDD, Test-Driven Development) est une approche du développement de programmes dans laquelle vous entrelacez le test et le développement de code.

✧Les tests sont écrits avant le code et "passer" les tests est le pilote critique du développement.

✧Vous développez le code de manière incrémentale, avec un test pour chaque incrément. Vous ne passez pas à l'incrément suivant tant que le code que vous avez développé n'a pas passé son test.

✧TDD a été introduit dans le cadre de méthodes agiles telles que Extreme Programming. Cependant, il peut également être utilisé dans les processus de développement planifiés.

Développement piloté par les tests



Activités du processus TDD

✧Commencez par identifier l'incrément de fonctionnalité requise. Cela devrait normalement être petit et facile à implémenter en quelques lignes de code.

✧Rédigez un test pour cette fonctionnalité et implémenter le en tant que test automatisé.

✧Exécutez le test, avec tous les autres tests qui ont été mis en œuvre. Au départ, vous n'avez pas implémenté la fonctionnalité, donc le nouveau test échouera.

✧Implémentez la fonctionnalité et réexécutez le test.

✧Une fois tous les tests exécutés avec succès, vous passez à l'implémentation du prochain segment de fonctionnalité.

Avantages du développement piloté par les tests

✧**Couverture de code**
- Chaque segment de code que vous écrivez a au moins un test associé de sorte que tout le code écrit a au moins un test.

✧**Les tests de régression**
- Une suite de tests de régression est développée progressivement à mesure qu'un programme est développé.

✧**Débogage simplifié**
- Quand un test échoue, il devrait être évident où le problème réside. Le code nouvellement écrit doit être vérifié et modifié.

✧**Documentation système**
- Les tests eux-mêmes sont une forme de documentation qui décrit ce que le code devrait faire.

Les tests de régression

✧Les tests de régression testent le système pour vérifier que les modifications n'ont pas «brisé» le code de travail précédent.

✧Dans un processus de test manuel, les tests de régression sont coûteux mais, avec des tests automatisés, ils sont simples et directs. Tous les tests sont réexécutés chaque fois qu'une modification est apportée au programme.

✧Les tests doivent être exécutés avec succès avant que la modification ne soit validée.

4. Les Tests de Sortie

Test de sortie

✧Le test de sortie (ou de version) est le processus de test d'une version particulière d'un système destiné à être utilisé en dehors de l'équipe de développement.

✧L'objectif principal du processus de test de version est de convaincre le fournisseur du système qu'il est assez bon pour l'utilisation.

✧Les tests de version doivent donc montrer que le système fournit ses fonctionnalités, ses performances et sa fiabilité, et qu'il ne tombe pas en panne lors d'une utilisation normale.

✧Les tests de version sont généralement un processus de test en boîte noire dans lequel les tests sont uniquement dérivés de la spécification du système.

Test de sortie et test du système

✧Les tests de version sont une forme de test du système.

✧Différences importantes:

- Une équipe distincte qui n'a pas été impliquée dans le développement du système devrait être responsable des tests de version.
- Les tests du système par l'équipe de développement doivent se concentrer sur la détection des bogues dans le système (test de défauts). L'objectif des tests de validation est de vérifier que le système répond à ses exigences et qu'il est suffisant pour une utilisation externe (test de validation).

Tests basés sur les exigences

✧Le test basé sur les exigences implique l'examen de chaque exigence et le développement d'un test ou des tests pour cela.

✧Exigences du système Mentcare:

- Si un patient est connu pour être allergique à un médicament en particulier, la prescription de ce médicament doit entraîner l'émission d'un message d'avertissement à l'utilisateur du système.
- Si un prescripteur choisit d'ignorer un avertissement d'allergie, il doit fournir une raison pour laquelle cela a été ignoré.

Tests d'exigences

✧Mettre en place un enregistrement patient sans allergies connues. Prescrire des médicaments pour les allergies qui existent. Vérifiez qu'un message d'avertissement n'est pas émis par le système.

✧Mettre en place un enregistrement patient avec une allergie connue. Prescrire le médicament auquel le patient est allergique et vérifier que l'avertissement est émis par le système.

✧Mettre en place un enregistrement patient dans lequel les allergies à deux médicaments ou plus sont enregistrées. Prescrire ces deux médicaments séparément et vérifier que l'avertissement correct pour chaque médicament est délivré.

✧Prescrire deux médicaments auxquels le patient est allergique. Vérifiez que deux avertissements sont correctement émis.

✧Prescrire un médicament qui émet un avertissement et annuler cet avertissement. Vérifiez que le système exige que l'utilisateur fournisse des informations expliquant pourquoi l'avertissement a été annulé.

Un scénario d'utilisation pour le système Mentcare

“George is a nurse who specializes in mental healthcare. One of his responsibilities is to visit patients at home to check that their treatment is effective and that they are not suffering from medication side effects.

On a day for home visits, George logs into the Mentcare system and uses it to print his schedule of home visits for that day, along with summary information about the patients to be visited. He requests

that the records for these patients be downloaded to his laptop. He is prompted for his key phrase to encrypt the records on the laptop.

One of the patients that he visits is Jim, who is being treated with medication for depression. Jim feels that the medication is helping him but believes that it has the side effect of keeping him awake at night. George looks up Jim's record and is prompted for his key phrase to decrypt the record. He checks the drug prescribed and queries its side effects. Sleeplessness is a known side effect so he notes the problem in Jim's record and suggests that he visits the clinic to have his medication changed. Jim agrees so George enters a prompt to call him when he gets back to the clinic to make an appointment with a physician. George ends the consultation and the system re-encrypts Jim's record.

After, finishing his consultations, George returns to the clinic and uploads the records of patients visited to the database. The system generates a call list for George of those patients who He has to contact for follow-up information and make clinic appointments".

Caractéristiques testées par scénario

✧Authentification en vous connectant au système.

✧Téléchargement (Downloading & uploading) des enregistrements patient spécifiques vers un ordinateur portable.

✧Calendrier de visite à domicile.

✧Cryptage et décryptage des enregistrements de patients sur un appareil mobile.

✧Enregistrer la récupération et la modification.

✧Liens avec la base de données sur les médicaments qui conservent des informations sur les effets secondaires.

✧Le système d'invite d'appel.

Test de performance

✧Une partie des tests de validation peut impliquer de tester les propriétés émergentes d'un système, telles que les performances et la fiabilité.

✧Les tests doivent refléter le profil d'utilisation du système.

✧Les tests de performance impliquent généralement la planification d'une série de tests où la charge est progressivement augmentée jusqu'à ce que les performances du système deviennent inacceptables.

✧Le test de résistance est une forme de test de performance où le système est délibérément surchargé pour tester son comportement en cas de défaillance.

5. Test d'utilisateur

✧Le test des utilisateurs ou des clients est une étape du processus de test dans lequel les utilisateurs ou les clients fournissent des entrées et des avis sur les tests du système.

✧Les tests utilisateur sont essentiels, même lorsque des tests complets du système et des versions ont été effectués.

- La raison en est que les influences de l'environnement de travail de l'utilisateur ont un effet majeur sur la fiabilité, la performance, la facilité d'utilisation et la robustesse d'un système. Ceux-ci ne peuvent pas être répliqués dans un environnement de test.

Types de tests utilisateur

✧**Test alpha**

- Les utilisateurs du logiciel travaillent avec l'équipe de développement pour tester le logiciel sur le site du développeur.

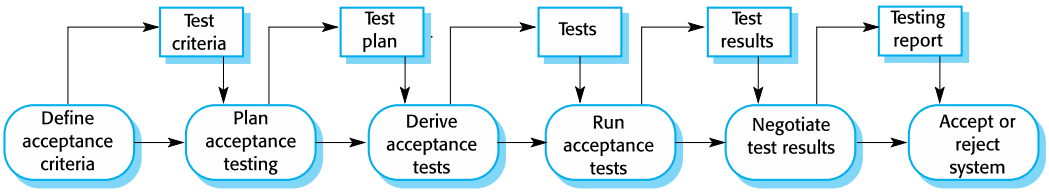
✧**Tests bêta**

- Une version du logiciel est mise à la disposition des utilisateurs pour leur permettre d'expérimenter et de soulever des problèmes qu'ils découvrent avec les développeurs du système.

✧**Tests d'acceptation**

- Les clients testent un système pour décider s'il est prêt ou non à être accepté des développeurs du système et déployé dans l'environnement client. Principalement pour les systèmes personnalisés.

Le processus de test d'acceptation



Les étapes du processus d'acceptation

1. Définir les critères d'acceptation
2. Planifier les tests d'acceptation
3. Dériver les tests d'acceptation
4. Exécuter des tests d'acceptation
5. Négocier les résultats des tests
6. Rejeter/accepter le système

Méthodes agiles et tests d'acceptation

✧Dans les méthodes agiles, l'utilisateur/client fait partie de l'équipe de développement et est responsable de prendre des décisions sur l'acceptabilité du système.

✧Les tests sont définis par l'utilisateur/client et sont intégrés à d'autres tests dans la mesure où ils sont exécutés automatiquement lors des modifications.

✧Il n'y a pas de processus de test d'acceptation séparé.

✧Le principal problème ici est de savoir si l'utilisateur intégré est «typique» et peut représenter les intérêts de toutes les parties prenantes du système.

6. Points clés

- ✓ Les tests peuvent seulement montrer la présence d'erreurs dans un programme. Il ne peut pas démontrer qu'il n'y a pas de failles restantes.
- ✓ Les tests de développement sont la responsabilité de l'équipe de développement logiciel. Une équipe distincte devrait être chargée de tester un système avant qu'il ne soit distribué aux clients.
- ✓ Les tests de développement incluent les tests unitaires, dans lesquels vous testez des objets individuels, et des tests de composants dans lesquels vous testez des groupes d'objets liés et des tests système, dans lesquels vous testez des systèmes partiels ou complets.
- ✓ Lorsque vous testez un logiciel, vous devez essayer de «casser» le logiciel en utilisant l'expérience et les directives pour choisir les types de cas de test qui ont été efficaces pour détecter les défauts dans d'autres systèmes.
- ✓ Autant que possible, vous devriez écrire des tests automatisés. Les tests sont intégrés dans un programme qui peut être exécuté chaque fois qu'une modification est apportée à un système.
- ✓ "Test-first development" est une approche du développement où les tests sont écrits avant le code à tester.
- ✓ Le test de scénario consiste à inventer un scénario d'utilisation typique et à l'utiliser pour dériver des cas de test.
- ✓ Le test d'acceptation est un processus de test d'utilisateur dont le but est de décider si le logiciel est suffisamment bon pour être déployé et utilisé dans son environnement opérationnel.

7. Exercices

1. Quel niveau de granularité vérifie le comportement de la coopération des modules?

    a) Tests unitaires
    b) Tests d'intégration
    c) Test d'acceptation
    d) Tests de régression

2. Expliquez pourquoi il n'est pas nécessaire qu'un programme soit complètement exempt de défauts avant d'être livré à ses clients.
3. Expliquez pourquoi les tests ne peuvent détecter que la présence d'erreurs et non leur absence.
4. Certaines personnes soutiennent que les développeurs ne devraient pas être impliqués dans le test de leur propre code, mais que tous les tests devraient être la responsabilité d'une équipe distincte. Donner des arguments pour et contre les tests par les développeurs eux-mêmes.
5. Vous avez été invité à tester une méthode appelée 'catWhiteSpace' dans un objet 'Paragraph' qui, dans le paragraphe, remplace les séquences de caractères vides par un seul caractère vide. Identifiez les partitions de test pour cet exemple et dérivez un ensemble de tests pour la méthode 'catWhiteSpace'.

6. Qu'est-ce qu'un test de régression? Expliquer comment l'utilisation de tests automatisés et d'un framework de test tel que JUnit simplifie les tests de régression.
7. Le système Mentcare est construit en adaptant un système d'information sur étagère (COTS). Selon vous, quelles sont les différences entre tester un tel système et tester un logiciel développé en utilisant un langage orienté objet tel que Java?
8. Rédiger un scénario qui pourrait être utilisé pour aider à concevoir des tests pour le système de stations météorologiques en zone sauvage.
9. Que comprenez-vous par le terme «stress testing»? Suggérez comment vous pourriez « stress tester » le système Mentcare.
10. Quels sont les avantages d'impliquer les utilisateurs dans les tests de version (sortie) à un stade précoce du processus de test? Y a-t-il des désavantages dans la participation des utilisateurs?
11. Une approche courante pour tester les systèmes consiste à tester le système jusqu'à ce que le budget de test soit épuisé, puis livrer le système aux clients. Discutez de l'éthique de cette approche pour les systèmes livrés à des clients externes.

8. Solutions

1. b, les tests d'intégration sont la phase des tests logiciels dans lesquels les modules logiciels individuels sont combinés et testés en tant que groupe.
3. Supposons qu'un test exhaustif d'un programme, où toutes les entrées valides possibles sont vérifiées, est impossible (vrai pour tous les programmes, sauf triviaux). Les cas de test ne révèlent pas une erreur dans le programme ou révèlent une erreur de programme. Si elles révèlent une erreur de programme, elles démontrent la présence d'une erreur. Cependant, s'ils ne révèlent pas un défaut, cela signifie simplement qu'ils ont exécuté une séquence de code qui, pour les entrées choisies, n'est pas défectueuse. Le prochain test de la même séquence de code - avec des entrées différentes - pourrait révéler un défaut.
5. **Les partitions de test sont**:

   Chaînes comportant uniquement des caractères vides
   Chaînes avec des séquences de caractères vides au milieu de la chaîne
   Chaînes avec des séquences de caractères vides au début/à la fin de la chaîne

   Exemples de tests:

   Le renard brun rapide a sauté sur le chien paresseux (seulement des blancs simples)
   Le renard brun rapide a sauté sur le chien paresseux (différents nombres de des blancs dans la séquence)
   Le renard brun rapide a sauté sur le chien paresseux (1er vide est une séquence)
   Le renard brun rapide a sauté sur le chien paresseux (Dernier blanc est une séquence)
   Le renard brun rapide a sauté par-dessus le chien paresseux (2 blancs au début)
   Le renard brun rapide a sauté sur le chien paresseux (plusieurs blancs au début)

Le renard brun rapide a sauté sur le chien paresseux (2 blancs à la fin)
Le renard brun rapide a sauté par-dessus le chien paresseux (plusieurs blancs à la fin)
Etc.

6. Le test de régression consiste à exécuter des tests pour les fonctionnalités qui ont déjà été implémentées lors de l'élaboration de nouvelles fonctionnalités ou du changement de système. Les tests de régression vérifient que les modifications du système n'ont pas introduit de problèmes dans le code précédemment implémenté.
   Des tests automatisés et un framework de test, tel que JUnit, simplifient radicalement les tests de régression car l'ensemble du test peut être exécuté automatiquement chaque fois qu'une modification est effectuée. Les tests automatisés comprennent leurs propres vérifications que le test a réussi ou non, de sorte que les coûts de vérification du succès ou non des tests de régression sont faibles.

8. Un scénario possible pour les tests de haut niveau du système de stations météorologiques est:
   John est un météorologue responsable de la production de cartes météorologiques pour l'état du Minnesota.
   Ces cartes sont produites à partir de données recueillies automatiquement à l'aide d'un système de cartographie météorologique et ils montrent des données différentes sur la météo au Minnesota. John sélectionne la zone pour laquelle la carte doit être produite, la période de temps de la carte et demande que la carte soit générée. Pendant la création de la carte, John effectue une vérification de la station météorologique qui examine toutes les données des stations météorologiques collectées à distance et recherche les lacunes dans ces données - ce qui impliquerait un problème avec la station météo distante.
   Il existe de nombreux scénarios alternatifs possibles ici. Ils devraient identifier le rôle des acteurs impliqués et discuter d'une tâche typique qui pourrait être accomplie par ce rôle.

9. Le test Stress (de résistance) est l'endroit où vous augmentez délibérément la charge sur un système au-delà de sa limite de conception pour voir comment il fait face à des charges élevées. Le système devrait se dégrader avec élégance plutôt que s'effondrer.
   Le système Mentcare a été conçu comme un système client-serveur avec la possibilité de télécharger vers un client. Pour tester le système, vous devez faire en sorte que (a) de nombreuses cliniques différentes essayent d'accéder au système en même temps et (b) qu'un grand nombre d'enregistrements soient ajoutés au système. Cela peut impliquer l'utilisation d'un système de simulation pour simuler plusieurs utilisateurs.

Chapitre IX

Evolution du Logiciel

Objectifs

- comprendre que le changement est inévitable si les systèmes logiciels doivent rester utiles et que le développement et l'évolution des logiciels peuvent être intégrés dans un modèle en spirale;
- comprendre les processus d'évolution des logiciels et leurs influences sur ces processus;
- avoir pris connaissance des différents types de maintenance logicielle et des facteurs qui influent sur les coûts de maintenance; et
- comprendre comment les systèmes existants peuvent être évalués pour décider s'ils doivent être mis au rebut, conservés, reconfigurés ou remplacés.

Themes couverts

- Processus d'évolution
- Systèmes hérités (Legacy systems)
- Maintenance logicielle

Chapitre 9:

Evolution du Logiciel

1. Introduction

Changement de logiciel

✧Le changement de logiciel est inévitable

- De nouvelles exigences apparaissent lorsque le logiciel est utilisé;
- L'environnement des affaires change;
- Les erreurs doivent être réparées.
- De nouveaux ordinateurs et équipements sont ajoutés au système;
- Les performances ou la fiabilité du système peuvent devoir être améliorées.

✧Un problème clé pour toutes les organisations est l'implémentation et la gestion du changement de leurs systèmes logiciels existants.

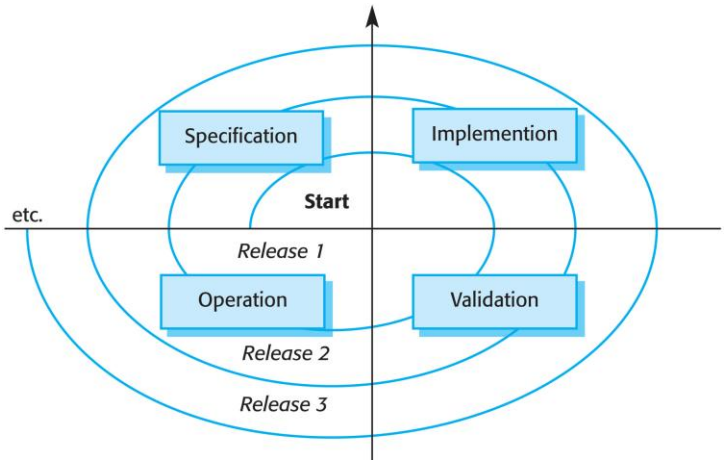
Importance de l'évolution

✧Les organisations investissent énormément dans leurs systèmes logiciels - ce sont des biens commerciaux critiques.

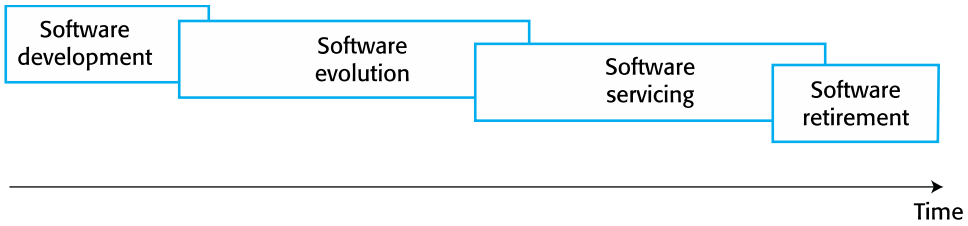
✧Pour maintenir la valeur de ces biens pour l'entreprise, ils doivent être modifiés et mis à jour.

✧La majeure partie du budget logiciel des grandes entreprises est consacrée au changement et à l'évolution des logiciels existants plutôt qu'au développement de nouveaux logiciels.

Un modèle en spirale de développement et d'évolution



Evolution et service



✧**Évolution**

- L'étape du cycle de vie d'un système logiciel où il est opérationnel et évolue à mesure que de nouvelles exigences sont proposées et implémentées dans le système.

✧**Entretien**

- A ce stade, le logiciel reste utile mais les seuls changements effectués sont ceux requis pour le rendre opérationnel, c'est-à-dire des corrections de bogues et des changements pour refléter les changements dans l'environnement du logiciel. Aucune nouvelle fonctionnalité n'est ajoutée.

✧**Élimination progressive**

- Le logiciel peut toujours être utilisé mais aucun autre changement ne lui est apporté.

2. Processus d'évolution

Processus d'évolution

✧Les processus d'évolution du logiciel dépendent de:

- Le type de logiciel maintenu;
- Les processus de développement utilisés;
- Les compétences et l'expérience des personnes impliquées.

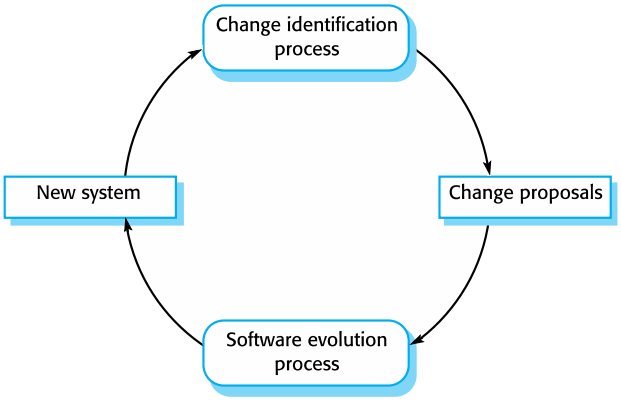
✧Les propositions de changement sont le moteur de l'évolution du système.

- Doit être lié aux composants affectés par le changement, ce qui permet d'estimer le coût et

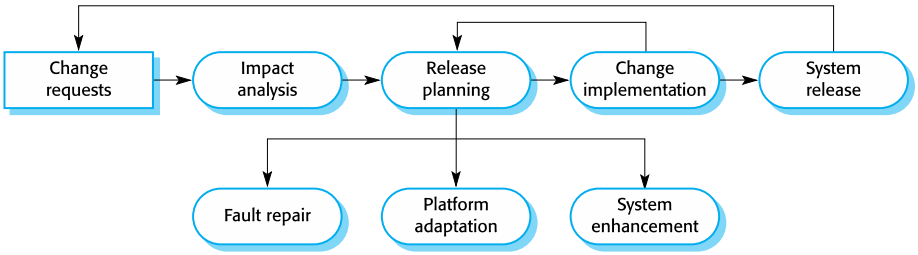
l'impact du changement.

✧L'identification et l'évolution du changement se poursuivent tout au long de la vie du système.

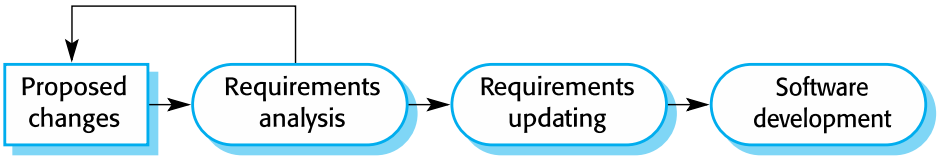
Identification du changement et les processus d'évolution



Le processus d'évolution du logiciel



Implémentation du changement



Implémentation du changement

✧Itération du processus de développement où les révisions du système sont conçues, mises en œuvre et testées.

✧Une différence essentielle est que la première étape de la mise en œuvre du changement peut impliquer la compréhension du programme, en particulier si les développeurs du système d'origine ne sont pas responsables de la mise en œuvre du changement.

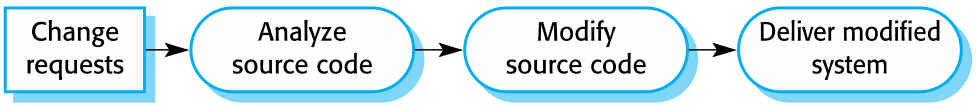
✧Pendant la phase de compréhension du programme, vous devez comprendre comment le programme est structuré, comment il offre des fonctionnalités et comment le changement proposé pourrait affecter le programme.

Demandes de changement urgentes

✧Des changements urgents peuvent devoir être mis en œuvre sans passer par toutes les étapes du processus d'ingénierie logicielle

- Si un défaut sérieux du système doit être réparé pour permettre le fonctionnement normal de continuer;
- Si des modifications de l'environnement du système (par exemple une mise à niveau du système d'exploitation) ont des effets inattendus;
- S'il existe des modifications métier (business) nécessitant une réponse très rapide (par exemple, la publication d'un produit concurrent).

Le processus de réparation d'urgence



Méthodes agiles et évolution

✧Les méthodes agiles sont basées sur un développement incrémental, de sorte que la transition du développement à l'évolution est transparente.

- L'évolution est simplement une continuation du processus de développement basé sur des versions fréquentes du système.

✧Les tests de régression automatisés sont particulièrement utiles lorsque des modifications sont apportées à un système.

✧Les modifications peuvent être exprimées en tant qu'histoires (stories) utilisateur supplémentaires.

Problèmes de livraison

✧Lorsque l'équipe de développement a utilisé une approche agile, l'équipe d'évolution ne connaît pas les méthodes agiles et préfère une approche basée sur les plans.

- L'équipe d'évolution peut s'attendre à une documentation détaillée pour prendre en charge l'évolution, ce qui n'est pas le cas dans les processus agiles.

✧Lorsqu'une approche basée sur un plan a été utilisée pour le développement, l'équipe d'évolution préfère utiliser des méthodes agiles.

- Il se peut que l'équipe d'évolution doive partir de zéro pour développer des tests automatisés et que le code du système n'ait pas été refaçonné et simplifié comme cela est prévu dans le développement agile.

3. Systèmes hérités (Legacy systems)

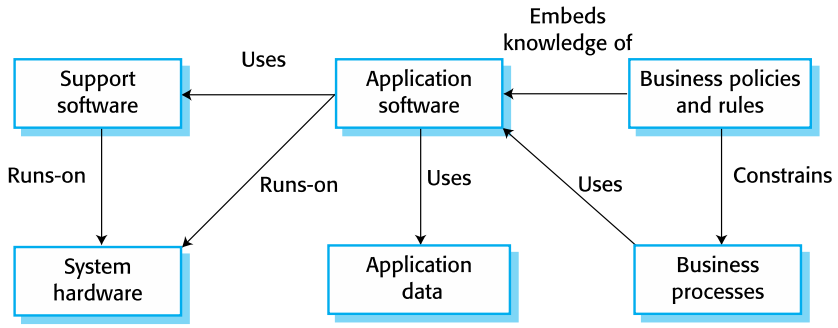
Systèmes hérités (ou existants)

✧Les systèmes hérités sont des systèmes plus anciens qui reposent sur des langages et des technologies qui ne sont plus utilisés pour le développement de nouveaux systèmes.

✧Les logiciels hérités peuvent dépendre d'équipements plus anciens, tels que les ordinateurs centraux, et peuvent avoir des processus et des procédures hérités associés.

✧Les systèmes hérités ne sont pas seulement des systèmes logiciels mais sont des systèmes socio-techniques plus larges qui comprennent du matériel, des logiciels, des bibliothèques et d'autres logiciels et processus opérationnels.

Les éléments d'un système existant



Composants des système existants

✧Matériel du système: Les systèmes hérités ont peut-être été écrits pour du matériel qui n'est plus disponible.

✧Logiciel de support: Le système existant peut s'appuyer sur une gamme de logiciels de support, qui peuvent être obsolètes ou non pris en charge.

✧Logiciel d'application: Le système d'application qui fournit les services métier est généralement composé d'un certain nombre de programmes d'application.

✧Données d'application: Il s'agit de données traitées par le système d'application. Ils peuvent être incohérents, dupliqués ou conservés dans des bases de données différentes.

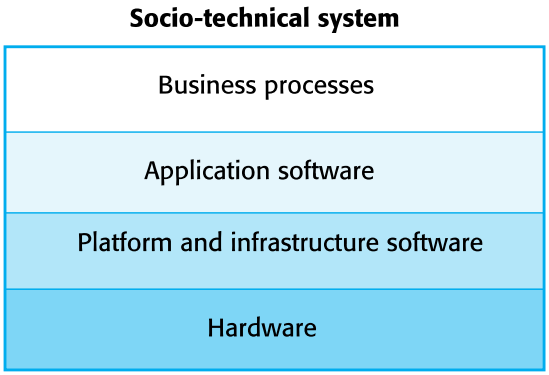
Composants des systèmes existants

✧Processus métier (d'affaires): Ce sont des processus qui sont utilisés dans l'entreprise pour atteindre certains objectifs d'affaires.

✧Les processus métier peuvent être conçus autour d'un système hérité et limités par la fonctionnalité qu'il fournit.

✧Politiques et règles métier: Ce sont des définitions de la manière dont l'activité doit être réalisée et des contraintes imposées à l'entreprise. L'utilisation du système d'application hérité peut être intégrée dans ces stratégies et règles.

Couches d'un système hérité



Remplacement du système hérité

✧Le remplacement du système existant est risqué et coûteux, les entreprises continuent donc d'utiliser ces systèmes

✧Le remplacement du système est risqué pour plusieurs raisons

- Manque de spécification complète du système
- Intégration étroite du système et les processus métiers
- Règles métier non documentées intégrées dans le système hérité
- Le développement de nouveaux logiciels peut être en retard et/ou dépasser le budget

Changement du système hérité

✧Les systèmes hérités sont coûteux à changer pour un certain nombre de raisons:

- Pas de style de programmation cohérent
- Utilisation de langages de programmation obsolètes avec peu de personnes disponibles avec ces compétences linguistiques
- Documentation système inadéquate
- Dégradation de la structure du système
- Les optimisations de programme peuvent les rendre difficiles à comprendre
- Erreurs de données, duplication et incohérence

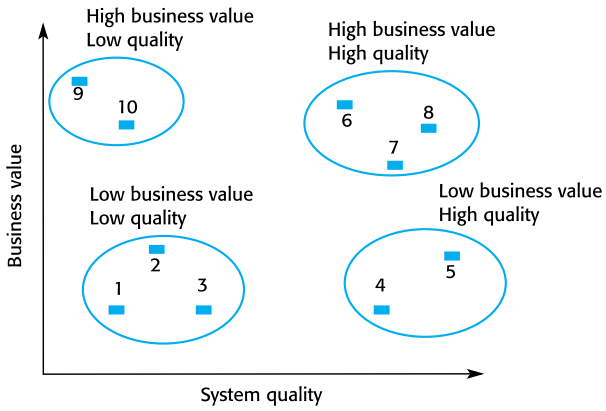
Gestion du système hérité

✧Les organisations qui s'appuient sur les systèmes hérités doivent choisir une stratégie pour faire évoluer ces systèmes

- Abandonner complètement le système et modifier les processus métier pour qu'ils ne soient plus nécessaires;
- Continuez à entretenir le système.
- Transformer le système en le réorganisant pour améliorer sa maintenabilité;
- Remplacez le système par un nouveau système.

✧La stratégie choisie doit dépendre de la qualité du système et de sa valeur commerciale.

Exemple d'évaluation d'un système hérité



Catégories de système héritées

✧**Faible qualité, faible valeur commerciale**
- Ces systèmes devraient être mis au rebut.

✧**Qualité faible, valeur commerciale élevée**
- Ceux-ci apportent une contribution importante de l'entreprise mais sont coûteux à entretenir. Devrait être reconfiguré ou remplacé si un système approprié est disponible.

✧**Haute qualité, faible valeur commerciale**
- Remplacez-les par des COTS, mettez-les au rebut complètement ou entretenez-les.

✧**Haute qualité, haute valeur commerciale**
- Continuer à fonctionner en utilisant la maintenance normale du système.

Évaluation de la valeur commerciale

✧L'évaluation devrait tenir compte de différents points de vue
- Utilisateurs finaux du système;
- Clients de l'entreprise;
- Les gestionnaires hiérarchiques;
- Les directeurs informatiques;
- Les cadres supérieurs

✧Interviewez différentes parties prenantes et rassemblez les résultats.

Problèmes d'évaluation de la valeur commerciale

✧**L'utilisation du système**
- Si les systèmes ne sont utilisés que de temps en temps ou par un petit nombre de personnes, ils peuvent avoir une faible valeur commerciale.

✧**Les processus métier pris en charge**
- Un système peut avoir une faible valeur commerciale s'il force l'utilisation de processus métiers inefficaces.

✧**La fiabilité du système**

- Si un système n'est pas fiable et que les problèmes affectent directement les clients professionnels, le système a une faible valeur commerciale.

✧**Les résultats du système**

- Si le métier (entreprise) dépend des résultats du système, le système a une valeur commerciale élevée.

Évaluation de la qualité du système

✧**Évaluation des processus d'affaires**

- Dans quelle mesure le processus opérationnel soutient-il les objectifs actuels de l'entreprise?

✧**Évaluation de l'environnement**

- Quelle est l'efficacité de l'environnement du système et à quel point est-il coûteux de le maintenir?

✧**Évaluation de l'application**

- Quelle est la qualité du système de logiciel d'application?

Évaluation des processus métiers

✧Utiliser une approche orientée point de vue (viewpoint-oriented approach) et chercher des réponses auprès des parties prenantes du système

- Existe-t-il un modèle de processus défini et est-il suivi?
- Est-ce que différentes parties de l'organisation utilisent des processus différents pour la même fonction?
- Comment le processus a-t-il été adapté?
- Quelles sont les relations avec les autres processus métiers et sont-elles nécessaires?
- Le processus est-il efficacement pris en charge par le logiciel d'application hérité?

✧Exemple - un système de commande de voyages peut avoir une faible valeur commerciale en raison de l'utilisation généralisée de la commande sur le Web.

Facteurs utilisés dans l'évaluation de l'environnement

Factor	Questions
Supplier stability	Is the supplier still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, does someone else maintain the systems?
Failure rate	Does the hardware have a high rate of reported failures? Does the support software crash and force system restarts?
Age	How old is the hardware and software? The older the hardware and support software, the more obsolete it will be. It may still function correctly but there could be significant economic and business benefits to moving to a more modern system.
Performance	Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support	What local support is required by the hardware and software? If there are high costs

requirements	associated with this support, it may be worth considering system replacement.
Maintenance costs	What are the costs of hardware maintenance and support software licences? Older hardware may have higher maintenance costs than modern systems. Support software may have high annual licensing costs.
Interoperability	Are there problems interfacing the system to other systems? Can compilers, for example, be used with current versions of the operating system? Is hardware emulation required?
Understandability	How difficult is it to understand the source code of the current system? How complex are the control structures that are used? Do variables have meaningful names that reflect their function?
Documentation	What system documentation is available? Is the documentation complete, consistent, and current?
Data	Is there an explicit data model for the system? To what extent is data duplicated across files? Is the data used by the system up to date and consistent?
Performance	Is the performance of the application adequate? Do performance problems have a significant effect on system users?
Programming language	Are modern compilers available for the programming language used to develop the system? Is the programming language still used for new system development?
Configuration management	Are all versions of all parts of the system managed by a configuration management system? Is there an explicit description of the versions of components that are used in the current system?
Test data	Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills	Are there people available who have the skills to maintain the application? Are there people available who have experience with the system?

Mesure du système

✧Vous pouvez collecter des données quantitatives pour évaluer la qualité du système d'application

✧Le nombre de demandes de changement de système; Plus cette valeur cumulée est élevée, plus la qualité du système est faible.

✧Le nombre d'interfaces utilisateur différentes utilisées par le système; Plus il y a d'interfaces, plus il est probable qu'il y aura des incohérences et des redondances dans ces interfaces.

✧Le volume de données utilisé par le système; Comme le volume de données (nombre de fichiers, taille de la base de données, etc.) traité par le système augmente, de même que les incohérences et les erreurs dans ces données.

✧Le nettoyage des anciennes données est un processus très coûteux et fastidieux

4. Maintenance logicielle

✧Modification d'un programme après sa mise en service

✧Le terme est principalement utilisé pour changer de logiciel personnalisé. Les produits logiciels

génériques évoluent pour créer de nouvelles versions.

✧La maintenance n'implique normalement pas de changements majeurs dans l'architecture du système.

✧Les modifications sont implémentées en modifiant les composants existants et en ajoutant de nouveaux composants au système.

Types de maintenance

✧**Réparations des défauts**

- Changer un système pour corriger les bogues/vulnérabilités et corriger les déficiences de manière conforme à ses exigences.

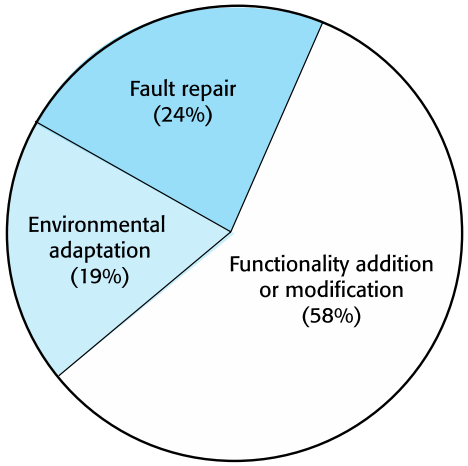
✧**Adaptation environnementale**

- Maintenance pour adapter le logiciel à un environnement d'exploitation différent
- Changer un système pour qu'il fonctionne dans un environnement différent (ordinateur, système d'exploitation, etc.) depuis son implémentation initiale.

✧**Ajout et modification de fonctionnalités**

- Modifier le système pour répondre aux nouvelles exigences.

Répartition de l'effort de maintenance



Coûts de maintenance

✧Habituellement supérieur aux coûts de développement (de 2* à 100* selon l'application).

✧Affecté par des facteurs techniques et non techniques.

✧Augmente à mesure que le logiciel est maintenu. La maintenance corrompt la structure du logiciel, rendant ainsi la maintenance plus difficile.

✧Le logiciel vieillissant peut avoir des coûts de support élevés (par exemple, les anciennes langues, les compilateurs, etc.).

✧Il est généralement plus coûteux d'ajouter de nouvelles fonctionnalités à un système pendant la

maintenance que d'ajouter les mêmes fonctionnalités au cours du développement.

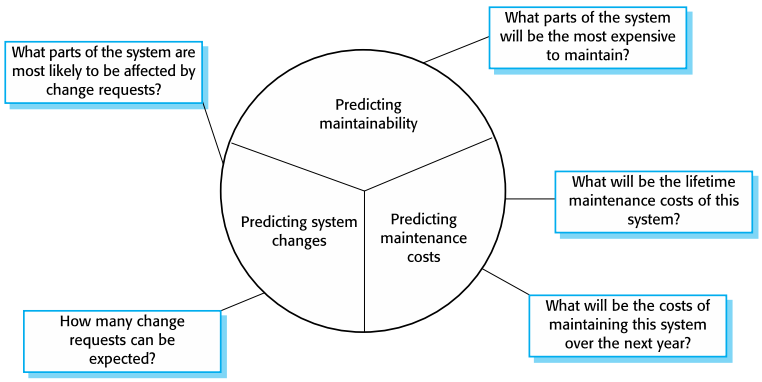
- Une nouvelle équipe doit comprendre les programmes maintenus
- La séparation de la maintenance et du développement signifie que l'équipe de développement n'a aucune motivation à écrire des logiciels maintenables
- Le travail de maintenance du programme est impopulaire
- Le personnel de maintenance est souvent inexpérimenté et possède une connaissance limitée du domaine.
- Au fur et à mesure que les programmes vieillissent, leur structure se dégrade et devient plus difficile à changer

Prédiction de maintenance

✧La prédiction de maintenance concerne l'évaluation des parties du système qui peuvent causer des problèmes et engendrer des coûts de maintenance élevés

- L'acceptation du changement dépend de la maintenabilité des composants affectés par le changement;
- La mise en œuvre de changements dégrade le système et réduit sa maintenabilité;
- Les coûts de maintenance dépendent du nombre de changements et les coûts de changement dépendent de la maintenabilité.

Prédiction de maintenance



Prédiction du changement

✧Prédire le nombre de changements nécessite et comprend les relations entre un système et son environnement.

✧Les systèmes couplés étroitement nécessitent des changements chaque fois que l'environnement est modifié.

✧Les facteurs influençant cette relation sont

- Nombre et complexité des interfaces système
- Nombre d'exigences système intrinsèquement volatiles
- Les processus métiers dans lesquels le système est utilisé

Métriques de complexité

✧Les prédictions de maintenabilité peuvent être faites en évaluant la complexité des composants du

système.

✧Des études ont montré que la plupart des efforts de maintenance sont consacrés à un nombre relativement faible de composants du système.

✧La complexité dépend de

- Complexité des structures de contrôle
- Complexité des structures de données
- Objet, méthode (procédure) et taille du module.

Mesures de processus

✧Les métriques de processus peuvent être utilisées pour évaluer la maintenabilité

- Nombre de demandes de maintenance corrective;
- Temps moyen requis pour l'analyse d'impact
- Temps moyen nécessaire pour mettre en œuvre une demande de changement
- Nombre de demandes de changements en attente

✧Si tout ou partie de ceux-ci augmente, cela peut indiquer une baisse de la maintenabilité.

Réingénierie logicielle

✧Restructurer ou réécrire tout ou partie d'un système existant sans changer sa fonctionnalité.

✧Applicable lorsque certains mais pas tous les sous-systèmes d'un système plus important nécessitent une maintenance fréquente.

✧La réingénierie consiste à ajouter des efforts pour les rendre plus faciles à entretenir. Le système peut être restructuré et re-documenté.

Avantages de la réingénierie

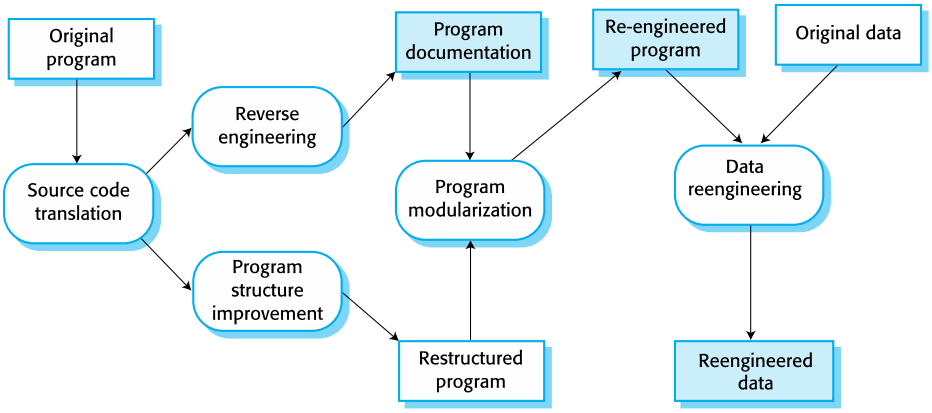
✧**Risque réduit**

- Il y a un risque élevé dans le développement de nouveaux logiciels. Il peut y avoir des problèmes de développement, des problèmes de personnel et des problèmes de spécification.

✧**Coût réduit**

- Le coût de la réingénierie est souvent nettement inférieur au coût de développement d'un nouveau logiciel.

Le processus de réingénierie



Activités du processus de réingénierie

✧**Traduction du code source**
- Convertir le code en un nouveau langage.

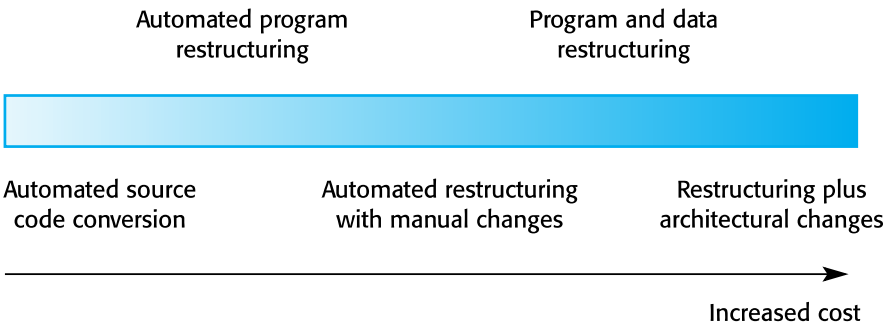
✧**Ingénierie inverse**
- Analyser le programme pour le comprendre

✧**Amélioration de la structure du programme**
- Restructurer automatiquement pour la compréhensibilité;

✧**Programme de modularisation**
- Réorganiser la structure du programme

✧**Ré-ingénierie des données**
- Nettoyer et restructurer les données du système.

Approches de réingénierie



Facteurs de coût de la réingénierie

✧La qualité du logiciel à restructurer.

✧Le support d'outil disponible pour la réingénierie.

✧L'étendue de la conversion de données requise.

✧La disponibilité du personnel expert pour la réingénierie.

- Cela peut poser un problème avec les anciens systèmes basés sur une technologie qui n'est plus largement utilisée.

Refactoring

✧Le refactoring est le processus d'amélioration d'un programme pour ralentir la dégradation par le changement.

✧Vous pouvez penser à refactoring comme «maintenance préventive» qui réduit les problèmes de changement futur.

✧Le refactoring consiste à modifier un programme pour améliorer sa structure, réduire sa complexité ou le rendre plus facile à comprendre.

✧Lorsque vous refactorisez un programme, vous ne devez pas ajouter de fonctionnalité mais plutôt vous concentrer sur l'amélioration du programme.

Refactoring et réingénierie

✧La réingénierie a lieu après la maintenance d'un système pendant un certain temps et les coûts de maintenance augmentent. Vous utilisez des outils automatisés pour traiter et réorganiser un système existant afin de créer un nouveau système plus facile à gérer.

✧Le refactoring est un processus continu d'amélioration tout au long du processus de développement et d'évolution. Il est destiné à éviter la dégradation de la structure et du code qui augmente les coûts et les difficultés de maintenance d'un système.

"Mauvaises odeurs" dans le code du programme

✧**Dupliquer le code**

- Le même code ou un code très similaire peut être inclus à différents endroits d'un programme. Cela peut être supprimé et implémenté comme une seule méthode ou fonction appelée comme requis.

✧**Longues méthodes**

- Si une méthode est trop longue, elle devrait être reconçue en plusieurs méthodes plus courtes.

✧**déclarations de switch (cas)**

- Ceux-ci impliquent souvent la duplication, où le switvh dépend du type d'une valeur. Les instructions switch peuvent être dispersées autour d'un programme. Dans les langages orientés objet, vous pouvez souvent utiliser le polymorphisme pour obtenir la même chose.

✧**Assemblage des données**

- Des groupes de données se produisent lorsque le même groupe d'éléments de données (champs dans les classes, paramètres dans les méthodes) se reproduisent à plusieurs endroits dans un programme. Ceux-ci peuvent souvent être remplacés par un objet qui encapsule toutes les données.

✧**Généralité spéculative**

- Cela se produit lorsque les développeurs incluent la généralité dans un programme au cas où cela serait nécessaire dans le futur. Cela peut souvent être simplement supprimé.

5. Points clés

- ✓ Le développement et l'évolution des logiciels peuvent être considérés comme un processus intégré et itératif qui peut être représenté à l'aide d'un modèle en spirale.
- ✓ Pour les systèmes personnalisés, les coûts de maintenance du logiciel dépassent généralement les coûts de développement du logiciel.
- ✓ Le processus d'évolution du logiciel est piloté par des demandes de changements et inclut l'analyse de l'impact du changement, la planification des versions et la mise en œuvre du changement.
- ✓ Les anciens systèmes sont des systèmes logiciels plus anciens, développés à l'aide de technologies matérielles et logicielles obsolètes, qui restent utiles pour une entreprise.
- ✓ Il est souvent moins coûteux et moins risqué de maintenir un système existant que de développer un système de remplacement utilisant la technologie moderne.
- ✓ La valeur commerciale d'un système existant et la qualité de l'application doivent être évaluées pour aider à décider si un système doit être remplacé, transformé ou maintenu.
- ✓ Il existe trois types de logiciels de maintenance, à savoir la correction de bogues, la modification de logiciels pour travailler dans un nouvel environnement et la mise en œuvre de nouvelles exigences ou de modifications.
- ✓ La réingénierie logicielle consiste à restructurer et à re-documenter les logiciels pour les rendre plus faciles à comprendre et à modifier.
- ✓ Le refactoring, en apportant des changements de programme qui préservent la fonctionnalité, est une forme de maintenance préventive.

6. Exercices

I) Quiz :

Choisir la bonne réponse :

1. L'évolution du logiciel ne comprend pas:

   a) Activités de développement
   b) Négocier avec le client
   c) Activités de maintenance
   d) Activités de réingénierie

2. Les processus d'évolution d'un produit logiciel dépendent:

   a) Type de logiciel à entretenir.
   b) Les processus de développement utilisés.
   c) Les compétences et l'expérience des personnes impliquées.
   d) Tout ce qui est mentionné

3. Quelle technique est appliquée pour assurer l'évolution continue des systèmes existants (legacy systems)?

   a) Ingénierie avancée (Forward engineering)

b) Ingénierie inverse (Reverse Engineering)
c) Réingénierie.
d) b et c

4. La modularisation du programme et la traduction du code source sont les activités de _____.

   a) Ingénierie avancée
   b) Ingénierie inverse
   c) Réingénierie.
   d) b et c

5. L'ingénierie inverse est la dernière activité d'un projet de réingénierie.

   a) Vrai
   b) Faux

6. Le coût de la réingénierie est souvent nettement inférieur au coût de développement de nouveaux logiciels.

   a) Vrai
   b) Faux

II) Exercices:

1. Expliquez pourquoi un système logiciel utilisé dans un environnement réel doit changer ou devenir progressivement moins utile.
2. À partir de la Figure 4 (Chapitre 9), vous pouvez voir que l'analyse d'impact est un sous-processus important dans le processus d'évolution du logiciel. À l'aide d'un diagramme, suggérez quelles activités pourraient être impliquées dans l'analyse de l'impact du changement.
3. En tant que chef de projet logiciel dans une entreprise spécialisée dans le développement de logiciels pour l'industrie pétrolière offshore, vous avez été chargé de découvrir les facteurs qui affectent la maintenabilité des systèmes développés par votre entreprise. Suggérez comment vous pourriez configurer un programme pour analyser le processus de maintenance et découvrir les mesures de maintenabilité appropriées pour votre entreprise.
4. Décrivez brièvement les trois principaux types de maintenance logicielle. Pourquoi est-il parfois difficile de les distinguer?
5. Quels sont les principaux facteurs qui influent sur les coûts de la réingénierie du système?
6. Dans quelles circonstances une organisation peut-elle décider de mettre au rebut un système lorsque l'évaluation du système indique qu'elle est de haute qualité et qu'elle a une grande valeur commerciale?
7. Quelles sont les options stratégiques pour l'évolution du système existant? Quand remplaceriez-vous normalement tout ou partie d'un système plutôt que de continuer la maintenance du logiciel?
8. Expliquez pourquoi les problèmes avec les logiciels de support peuvent signifier qu'une organisation doit remplacer ses systèmes existants.
9. Les ingénieurs en logiciel ont-ils la responsabilité professionnelle de produire un code qui peut être maintenu et modifié même si cela n'est pas explicitement demandé par leur employeur?

7. Solutions

I) Quiz:

1 : b, L'évolution des logiciels fait référence à l'étude et à la gestion du processus de modification des logiciels au fil du temps. Ainsi, il comprend les trois autres options données
2. d, Les processus utilisés pour l'évolution du logiciel dépendent de tous ces facteurs.
3. d, Les processus utilisés pour l'évolution du logiciel dépendent de ces deux techniques.
4. c, Réingénierie est l'examen et la modification d'un système pour le reconstituer sous une nouvelle forme et la mise en œuvre ultérieure de la nouvelle forme.
5. b, L'ingénierie inverse est souvent l'activité initiale d'un projet de réingénierie.
6. a, Il y a un risque élevé dans le développement de nouveaux logiciels. Il peut y avoir des problèmes de développement, des problèmes de personnel et des problèmes de spécification, ce qui augmente le coût.

II) Exercices

1. Les systèmes doivent changer ou devenir progressivement moins utiles pour un certain nombre de raisons:
    1. La présence du système modifie les façons de travailler dans son environnement et cela génère de nouvelles exigences. Si celles-ci ne sont pas satisfaites, l'utilité du système diminue.
    2. L'activité dans laquelle le système est utilisé change en réponse aux forces du marché, ce qui génère également de nouvelles exigences du système.
    3. L'environnement juridique et politique externe du système change et génère de nouvelles exigences.
    4. De nouvelles technologies deviennent disponibles offrant des avantages significatifs et le système doit changer pour en tirer parti.

3. C'est une question très ouverte, où il y a beaucoup de réponses possibles.
   Fondamentalement, les étudiants devraient identifier les facteurs qui affectent la maintenabilité tels que (complexité du programme et des données, utilisation d'identifiants significatifs, langage de programmation, documentation du programme, etc.). Ils devraient ensuite suggérer comment ils peuvent être évalués dans les systèmes existants dont le coût de maintenance est connu et discuter des problèmes d'interaction. L'approche devrait consister à découvrir les unités de programme qui ont des coûts d'entretien particulièrement élevés et à évaluer les facteurs de coût de ces composants et d'autres composants. Ensuite, vérifiez les corrélations.
   D'autres facteurs peuvent expliquer les anomalies, elles doivent donc être recherchées dans les composants du problème.

4. Les trois principaux types de maintenance logicielle sont:
    1. Maintenance corrective ou réparation de panne. Les modifications apportées au système consistent à réparer les erreurs signalées qui peuvent être des bogues de programme ou des erreurs ou omissions de spécification.

2. Maintenance adaptative ou adaptation environnementale. Changer le logiciel pour l'adapter aux changements de son environnement, par ex. changements à d'autres systèmes logiciels.
3. Maintenance perfective ou ajout de fonctionnalités. Cela implique l'ajout de nouvelles fonctionnalités ou fonctionnalités au système.

Ils sont parfois difficiles à distinguer parce que le même ensemble de changements peut couvrir les trois types de maintenance. Par exemple, un défaut signalé dans le système peut être réparé en mettant à niveau d'autres logiciels et en adaptant ensuite le système pour utiliser cette nouvelle version (corrective + adaptative). Le nouveau logiciel peut avoir des fonctionnalités supplémentaires et dans le cadre de la maintenance adaptative, de nouvelles fonctionnalités peuvent être ajoutées pour en tirer parti.

6. Des exemples de logiciels susceptibles d'être supprimés et réécrits sont:
   1. Lorsque le coût de la maintenance est élevé et que l'organisation a décidé d'investir dans du nouveau matériel. Cela impliquera de toute façon des coûts de conversion importants, ce qui permettra de réécrire le logiciel.
   2. Lorsqu'un processus métier est modifié et qu'un nouveau logiciel est nécessaire pour le processus.
   3. Lorsque le support des outils et du langage utilisés pour développer le logiciel n'est pas disponible. C'est un problème particulier avec les premiers 4GL où, dans de nombreux cas, les fournisseurs ne sont plus en affaires.

   Il existe d'autres raisons pour lesquelles le logiciel peut être mis au rebut, en fonction des circonstances.

7. Les options stratégiques pour l'évolution du système existant sont:
   1. Abandonnez la maintenance du système et remplacez-le par un nouveau système.
   2. Continuez à maintenir le système tel qu'il est.
   3. Effectuez une nouvelle ingénierie (amélioration du système) qui facilite l'entretien du système et la maintenance.
   4. Encapsulez la fonctionnalité existante du système dans une enveloppe (wrapper) et ajoutez une nouvelle fonctionnalité en écrivant un nouveau code qui appelle le système existant en tant que composant.
   5. Décomposer le système en unités séparées et les envelopper comme composants. Ceci est similaire à la solution ci-dessus mais donne plus de flexibilité dans la façon dont le système est utilisé.

   Normalement, vous choisissez l'option de remplacement dans les cas où la plateforme matérielle du système est remplacée, où l'entreprise souhaite normaliser une approche de développement qui n'est pas compatible avec le système actuel, où un sous-système majeur est remplacé (par exemple un système de base de données) ou lorsque la qualité technique du système existant est faible et qu'il n'existe pas d'outils actuels pour la réingénierie.

Bibliograhie

Références

1. Software Engineering, Ian Sommerville, 10th Edition, Addison-Wesley, 2015.
2. Software Engineering: A Practitioner's Approach, 8th Edition – Roger S. Pressman & Bruce R. Maxim, McGraw-Hill, 2015.
3. Software Engineering: Theory and Practice, Shari L. Pfleeger and Joanne M. Atlee, 4th Edition, Prentice Hall, 2009.
4. Analyse des besoins pour le développement logiciel, Jacques Lonchamp, Dunod, 2015.
5. Conception d'applications en java-jee, Jacques Lonchamp, Dunod, 2° édition, 2019.
6. Software Engineering: Modern Approaches, Eric J. Braude and Michael E. Bernstein, 2nd Edition, John Wiley & Sons, 2010.
7. Handbook of Software Engineering, Editors: Sungdeok Cha, Richard N. Taylor, Kyochul Kang, Springer, 2019.
8. The Essence of Software Engineering, Editors: Gruhn Volker**,** Striemer Rüdiger**,** Springer, 2018.
9. Génie logiciel, David Gustafson, EDISCIENCE, 2003.
10. Génie logiciel: principes, méthodes et techniques, Alfred Strohmeier et Didier Buchs, Eyrolles, 1999.
11. Introduction au génie logiciel, Pascal ANDRE, 2001.
12. Cours « Génie Logiciel », Pierre Gérard, University of Paris 13, France, 2007/2008.
13. Cours « Génie Logiciel avancé », Stefano Zacchiroli, Univesité Paris 7, 2013/2014.
14. Cours « Génie logiciel », Tom mens, University of Mons, Belguim
15. Cours « Introduction à l'eXtreme Programming et au développement agile », Gauthier Picard, Octobre 2009.
16. Cours « Analyse et Conception des Systèmes d'Information », Olivier Guibert, Université de Bordeaux 1, 2007.
17. Cours « Introduction au Génie Logiciel », Tony Wong, 2000.
18. Cours « introduction au Génie Logiciel », Mostefai Mohammed Amine, Batata Sofiane, ESI, Alger, 2018-2019.