

Université Virtuelle Africaine

INFORMATIQUE APPLIQUÉE: CSI 4205

GÉNIE LOGICIEL

Dr. Cherif Diallo

Avant propos

L'Université virtuelle africaine (UVA) est fier de participer à l'amélioration de l'accès à l'éducation dans les pays africains à travers la production de matériel didactique de qualité. Nous sommes également fiers de contribuer aux connaissances mondiales comme nos ressources pédagogiques sont pour la plupart accessibles de l'extérieur du continent africain.

Ce module a été développé dans le cadre d'un programme à un diplôme en informatique appliquée, en collaboration avec 18 institutions partenaires africaines de 16 pays. Un total de 156 modules ont été élaborés pour assurer la disponibilité ou traduit en anglais, français et portugais. Ces modules ont également été mis à disposition en tant que ressources éducatives libres (REL) sur oer.avu.org.

Au nom de l'Université virtuelle africaine et notre patron, nos institutions partenaires, la Banque africaine de développement, je vous invite à utiliser ce module dans votre établissement, pour votre propre formation, de partager le plus largement possible et à participer activement à l'avu les communautés de pratique de votre intérêt. Nous nous engageons à être en première ligne de l'élaboration et le partage de ressources éducatives libres.

L'Université virtuelle africaine (UVA) est une organisation intergouvernementale panafricaine créée par la location avec le mandat d'accroître sensiblement l'accès à un enseignement supérieur de qualité et de formation à l'aide de l'information technologies de la communication. Une Charte, l'établissement de l'avu en tant qu'organisation intergouvernementale, a été signé ce jour par dix-neuf (19) Les gouvernements africains - le Kenya, le Sénégal, la Mauritanie, le Mali, la Côte d'Ivoire, Tanzanie, Mozambique, République démocratique du Congo, Bénin, Ghana, République de Guinée, Burkina Faso, Niger, Soudan du Sud, Soudan, l'Éthiopie, la Gambie, la Guinée-Bissau et le Cap-Vert.

Les institutions suivantes ont participé au programme d'informatique appliquée : (1) Université d'Abomey Calavi au Bénin ; (2) Université de Ougagadougou au Burkina Faso ; (3) l'Université Lumière de Bujumbura au Burundi ; (4) l'Université de Douala au Cameroun ; (5) Université de Nouakchott en Mauritanie ; (6) l'Université Gaston Berger au Sénégal ; (7) Université des Sciences, des Techniques et technologies de Bamako au Mali (8) Ghana Institute of Management and Public Administration ; (9) Université des Sciences et Technologies de Kwame Nkrumah au Ghana ; (10) l'Université Kenyatta au Kenya ; (11) l'Université d'Egerton au Kenya ; (12) l'Université d'Addis Abeba en Ethiopie (13) Université du Rwanda (14) ; Université de Dar es Salaam en Tanzanie ; (15) l'Université Abdou Moumouni de Niamey au Niger ; (16) l'Université Cheikh Anta Diop de Sénégal ; (17) Universidade Pedagógica au Mozambique ; et (18) l'Université de la Gambie en Gambie.

Bakary Diallo,

Recteur de l'

Université virtuelle africaine

Crédits de production

Auteur

Dr. Cherif Diallo

Pair Réviseur

Jacqueline Sogoba Konate

UVA – Coordination Académique

Dr. Marilena Cabral

Coordinateur global Sciences Informatiques Apliquées

Prof Tim Mwololo Waema

Coordinateur du module

Robert Oboko

Concepteurs pédagogiques

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Equipe Média

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Droits d'auteur

Ce document est publié dans les conditions de la Creative Commons

Http://fr.wikipedia.org/wiki/Creative_Commons

Attribution http://creativecommons.org/licenses/by/2.5/

Le gabarit est copyright African Virtual University sous licence Creative Commons Attribution-ShareAlike 4.0 International License. CC-BY, SA

Supporté par

Projet Multinational II de l'UVA financé par la Banque africaine de développement.

Table des matières

Avant propos 2

Crédits de production 3

Droits d'auteur 4

Supporté par 4

Aperçu du cours 9

Bienvenue à Génie Logiciel 9

Prérequis. 9

Matériaux 9

Objectifs du cours 9

Unités 10

Évaluation 11

Plan 12

Lectures et autres ressources 12

Unité 0. Évaluation diagnostique 15

Introduction à l'unité. 15

Objectifs de l'unité 15

Évaluation de l'unité 15

Termes clés 15

Lectures et autres ressources 16

Unité 1. Fondamentaux du génie logiciel 17

Introduction à l'unité. 17

Objectifs de l'unité 17

Termes clés 18

Activités d'apprentissage 18

Activité 1: Logiciel. 18

Introduction 18

Détails de l'activité 18

Activité 2: Qu'est-ce que le Génie Logiciel 21

Introduction 21

Évaluation 21

Activité 3: Qu'est-ce que le Génie Logiciel 23

Introduction 23

Évaluation 23

Évaluation 25

Évaluation 27

Activité 4: Cycle de vie du développement 28

Évaluation 31

Activité 5: Approches de développement de logiciel 32

Évaluation 36

Lectures et autres ressources 37

Résumé de l'unité 37

Évaluation de l'unité 37

Unité 2. Planification d'un projet logiciel et Analyse et spécifications des besoins 38

Introduction à l'unité. 38

Objectifs de l'unité 38

Termes clés 38

Activités d'apprentissage 39

Activité 1: Ingénierie des exigences. 39

Introduction 39

Activité 2: Planification de projet logiciel 45

Introduction 45

Évaluation 45

Lectures et autres ressources 51

Évaluation 51

Résumé de l'unité 51

Évaluation de l'unité 51

Unité 3. Conception de logiciels 52

Introduction à l'unité. 52
Objectifs de l'unité 52
Termes clés 52
Activités d'apprentissage 53
Activité 1: Conception de logiciel ou Software Design 53
Introduction 53
Activité 2: Le processus de conception 59
Introduction 59
Évaluation 59
Évaluation 70
Activité 3: Autres aspects de la conception de logiciels 71
Évaluation 73
Résumé de l'unité 73
Lectures et autres ressources 74
Évaluation de l'unité 74
Unité 4. Implémentation et tests 75
Introduction à l'unité. 75
Objectifs de l'unité 75
Termes clés 75
Activités d'apprentissage 76
Activité 1: Implémentation ou Codage du logiciel 76
Introduction 76
Activité 2: Fondamentaux du Test de Logiciel 79
Évaluation 79
Évaluation 81
Activité 3: Niveaux de Test 82
Évaluation 86
Activité 4: Test boite blanche, Test boite noire (White-Box, Black Box-Testing). . 87
Évaluation 92
Résumé de l'unité 92

Lectures et autres ressources 93

Évaluation de l'unité 93

Unité 5. Maintenance et Gestion de projet **94**

Introduction à l'unité. 94

Objectifs de l'unité 94

Termes clés 94

Activités d'apprentissage 95

Activité 1: La phase de maintenance du logiciel 95

Introduction 95

Activité 2: Analyse et gestion des risques de logiciels 98

Introduction 98

Évaluation 98

Lectures et autres ressources 103

Évaluation 103

Résumé de l'unité 103

Évaluation de l'unité 103

Aperçu du cours

Bienvenue à Génie Logiciel

Le Génie Logiciel concerne tous les aspects de la production de logiciels à partir des premières étapes de la spécification du système jusqu'à la maintenance du système après qu'il soit entré en utilisation (Laurie Williams 2004). Il permet d'adopter une pratique d'un point de vue technique, une approche systématique et organisée pour développer des systèmes.

Le cours expose les étudiants sur la façon d'utiliser plusieurs pratiques spécifiques (ou techniques) pour le développement de logiciels. Il est nécessaire de familiariser les apprenants à des outils de modélisation industriels dans le but de les exposer à des pratiques de l'état de l'art en matière de développement de logiciels. Ce faisant, l'étudiant aura une meilleure compréhension de la complexité ainsi que les subtilités des différentes activités de développement de logiciels qui incluent la collaboration dans une équipe ou un groupe. Les étudiants apprendront à construire un logiciel efficace, appliquer les bonnes pratiques, les techniques de conception efficaces et des outils de développement. Le Génie Logiciel est nécessaire pour développer toutes sortes de projets de logiciels, y compris des projets de logiciels complexes.

Prérequis

- Introduction aux bases de données
- Introduction à la programmation structurée

Matériaux

Les matériaux nécessaires pour compléter ce cours comprennent les :

- Un PC équipé d'un système d'exploitation, de préférence connecté à internet;
- Livres
- Outils logiciels
- Internet, vidéos en ligne, Wiki, Forum de discussion

Objectifs du cours

Préparer les étudiants à bien maîtriser les techniques et concepts de base du génie logiciel.

À la fin de ce cours, vous serez capable de :

- utiliser les techniques informatiques et les principes d'ingénierie pour les projets de développement des logiciels.

- concevoir, développer et tester les logiciels systèmes et applications aussi bien pour l'industrie, les affaires que pour les besoins individuels.
- appliquer des méthodes éprouvées et les bonnes pratiques pour produire des logiciels de qualité dans le strict respect du budget et du calendrier.
- extraire, analyser et spécifier les besoins à travers une bonne productivité de travail avec différentes parties prenantes et une communication efficace dans le respect de l'éthique et des règles professionnelles du génie logiciel.

Unités

Unité 0: Évaluation diagnostique

Cette unité fournit une vue d'ensemble du module Génie Logiciel. Elle passe en revue les bases du Génie Logiciel et précise l'organisation des modules, les contenus, activités et évaluations.

Unité 1: Les fondamentaux du Génie Logiciel

L'Unité a pour objectif de présenter et d'expliquer l'importance du Génie Logiciel dans le développement logiciel. Il décrit les problèmes de développement des logiciels et pose des questions éthiques et professionnelles nécessaires dans la pratique du génie logiciel. Le développement de logiciels a besoin d'une approche systématique. Cette partie explique également une série d'étapes (une feuille de route) qui aide à créer des produits logiciels de qualité. Il décrit également des modèles de processus génériques.

Unité 2: Planification de projet, analyse et spécification des besoins

Cette unité présente les bases de production d'un cahier des charges en utilisant des mécanismes d'ingénierie propres aux spécifications des besoins. Il explique comment comprendre ce que les clients veulent, analyser et valider leurs besoins et enfin produire le cahier des charges. L'unité élabore également un ensemble d'activités liées à la planification de projet logiciel. La planification du projet touche également l'estimation du coût ainsi que le calendrier du projet.

Unité 3: Conception de logiciel

Le but de la conception du logiciel est de produire un modèle ou une représentation d'une entité qui sera construit plus tard. Il peut être attribuée à des exigences d'un client et en même temps évalué pour la qualité d'un ensemble de critères prédéfinis pour «bon» design. Cette unité décrit les objectifs de conception et les principes de conception.

Il élabore des processus logiciels de conception qui comprennent: Architectural Design, conception abstraite, User-Interface Design, Low Level-Design. conception orientée-Fonction, conception orientée objet, la notation de la conception et la spécification et la vérification de la conception ont également été présentés.

Unité 4: Implémentation et tests

La phase de test est un élément essentiel de l'assurance qualité des logiciels et représente l'examen final de la spécification, de la conception et de la génération de code. Le Test est l'assurance des réponses apportées aux spécifications depuis la phase de conception jusqu'à la mise en œuvre finale.

Cette unité traite en général, tous les aspects liés à la mise en œuvre des logiciels et des tests y afférents. Il décrit les techniques utilisées pour la mise en œuvre, ainsi que les mécanismes utilisés pour effectuer les tests des logiciels développés.

Unité 5: Management de projet et maintenance

Cette unité se concentrera sur tous les aspects liés à la gestion et à la maintenance du logiciel. L'unité explique pourquoi la maintenance du logiciel est nécessaire. Elle explique également les différentes catégories de maintenance, un aperçu des coûts de maintenance et les différents facteurs affectant la maintenance. Enfin, l'unité traite les problématiques d'analyse et de gestion des risques logiciels.

Évaluation

Les évaluations formatives (vérification de progrès) sont incluses dans chaque unité.

Les évaluations sommatives (tests et travaux finaux) sont fournies à la fin de chaque module et traitent des connaissances et compétences du module.

Les évaluations sommatives sont gérées à la discrétion de l'établissement qui offre le cours. Le plan d'évaluation proposé est le suivant:

20%	Exploitation des documents et autres ressources	1
40%	Exercices pratiques	2
40%	Evaluations sommatives et formatives	3

Plan

Unité	Sujets et Activités	Durée estimée
Unité 0	Connaissances préliminaires	10h
Unité 1	Travaux individuels théoriques, et pratiques. Tutorats	24h
Unité 2	Travaux individuels théoriques, et pratiques. Tutorats	24h
Unité 3	Travaux individuels théoriques, et pratiques. Tutorats	24h
Unité 4	Travaux individuels théoriques, et pratiques. Tutorats	24h
Unité 5	Travaux individuels théoriques, et pratiques. Tutorats	24h

Lectures et autres ressources

Les lectures et autres ressources dans ce cours sont indiquées ci-dessous.

Unité 0

Lectures et autres ressources obligatoires:

- Christian Soutou, Frédéric Brouard. « Modélisation de bases de données : UML et les modèles entité-association », Eyrolles, 2015

Unité 1

Lectures et autres ressources obligatoires:

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783

Lectures et autres ressources optionnelles:

- Zhiming Liu, (2001), "Object-Otiented Software Development Using UML", The United Nations University, UNU-IIST International Institute for Software Technology, Tech Report 229
- Sommerville Ian (2000), "Software Engineering (6th Edition)". Addison-Wesley, Boston USA

Unité 2

Lectures et autres ressources obligatoires:

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783

Lectures et autres ressources optionnelles:

- Zhiming Liu, (2001), "Object-Otiented Software Development Using UML", The United Nations University, UNU-IIST International Institute for Software Technology, Tech Report 229
- Sommerville Ian (2000), "Software Engineering (6th Edition)". Addison-Wesley, Boston USA

Unité 3

Lectures et autres ressources obligatoires:

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783

Lectures et autres ressources optionnelles:

- Sommerville Ian (2000), "Software Engineering (6th Edition)". Addison-Wesley, Boston USA.

Unité 4

Lectures et autres ressources obligatoires:

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783

Lectures et autres ressources optionnelles:

- Sommerville Ian (2000), "Software Engineering (6th Edition)". Addison-Wesley, Boston USA
- Alain Abran and James W. Moore, (2004), "IEEE Guide to the Software Engineering Body of Knowledge (SWEBOK)",
- David Gustafson (2002), "Theory and Problems of Software Engineering", Schaum's Outline Series, McGraw-Hill, 0-07-140620-4

Unité 5

Lectures et autres ressources obligatoires:

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783

Lectures et autres ressources optionnelles:

- Sommerville Ian (2000), "Software Engineering (6th Edition)". Addison-Wesley, Boston USA
- Alain Abran and James W. Moore, (2004), "IEEE Guide to the Software Engineering Body of Knowledge (SWEBOK)",
- David Gustafson (2002), "Theory and Problems of Software Engineering", Schaum's Outline Series, McGraw-Hill, 0-07-140620-4

Unité 0. Évaluation diagnostique

Introduction à l'unité

Cette unité vous permettra de vérifier les connaissances que vous devez avoir avant de commencer le cours. Vous pouvez faire l'évaluation de l'unité avant de faire des activités d'apprentissage pour aider à rafraîchir vos connaissances. Cette unité permet d'évaluer si les apprenants ont été initiés aux principes fondamentaux du Génie Logiciel

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de :

- décrire une vue d'ensemble des SGBD ;
- décrire les avantages de SGBD ;
- décrire et stocker des données dans un SGBD,
- décrire la structure d'un SGBD,
- comprendre le modèle entité-relation,
- comprendre le cycle de vie du développement logiciel.

Termes clés

SGBD: Système de gestion de bases de données

UML: Unified Modeling Language

Évaluation de l'unité

Vérifiez votre compréhension!

Activité de lecture : lire la référence citée dans la rubrique Lectures et autres ressources, et expliquer en détails les concepts clés des SGBD, du langage UML et des modèles entité-association

Lectures et autres ressources

- Christian Soutou, Frédéric Brouard. « Modélisation de bases de données : UML et les modèles entité-association », Eyrolles, 2015

Unité 1. Fondamentaux du génie logiciel

Introduction à l'unité

L'Unité commence par expliquer ce qu'est un logiciel ainsi que ses différentes topologies. Il décrit la genèse et les mythes de développement de logiciels. L'unité introduit la nécessité du génie logiciel et le processus de développement du logiciel en expliquant le cycle de vie du développement logiciel. Les modèles de processus génériques, en cascade et évolutifs seront abordés.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de :

- décrire le terme logiciel et ses types.
- définir le génie logiciel et expliquer pourquoi l'ingénierie est utilisée dans le développement de logiciels.
- illustrer les phases du processus de développement des logiciels, c'est à dire le cycle de vie du développement de logiciel.
- décrire les modèles du processus de développement de logiciels.

Termes clés

Logiciel: Le logiciel est un ensemble d'instructions avec toutes les données de documentation et de configuration associés utilisées pour acquérir des intrants et les manipuler pour produire la sortie désirée en termes de fonctions et de performances tel que déterminé par l'utilisateur du logiciel.

Génie Logiciel: Le génie logiciel est l'application d'une discipline, approche systématique, quantifiable pour le développement, l'exploitation et la maintenance des logiciels. Il comprend également l'étude de ces approches.

Produit logiciel: Les produits logiciels sont des systèmes logiciels livrés à un client avec la documentation qui décrit comment installer et utiliser le système.

Processus: définit un cadre pour un ensemble de secteurs clés qui doivent être mis en place pour la livraison efficace des technologies de génie logiciel

Activités d'apprentissage

Activité 1: Logiciel

Introduction

Dans cette activité, la définition des logiciels, les types et les catégories de logiciels sont présentés. Nous aborderons également les mythes et la crise de logiciels.

Détails de l'activité

1 Qu'est-ce qu'un logiciel ?

Le logiciel n'est pas seulement le programme mais il inclut également toutes les données (de la documentation et de la configuration associée) nécessaires à leur bon fonctionnement. Tel que défini par Agarwal et al. (2010), le logiciel est un ensemble d'instructions utilisées pour acquérir des intrants et de les manipuler pour produire la sortie désirée en termes de fonctions et de performances tel que déterminé par l'utilisateur du logiciel. Il comprend également un ensemble de documents, tels que le manuel du logiciel, destiné à aider les utilisateurs à comprendre le système logiciel. Le logiciel d'aujourd'hui est composé du code source, des exécutables, des documents et plans de conception, des opérations d'installation et de mise en œuvre du système et les manuels. Autrement dit, un système logiciel est constitué de :

- Instructions (un certain nombre de programmes informatiques distincts) qui, lorsqu'elles sont exécutées fournissent les fonctions et performances souhaitées
- Les fichiers de configuration qui sont utilisés pour mettre en place ces programmes et les structures de données qui permettent aux programmes de manipuler adéquatement les informations
- La documentation du système qui décrit la structure du système
- La documentation utilisateur qui explique l'utilisation des programmes et du système

2 Crise du logiciel

Pourquoi les projets de développement de logiciels ne produisent pas des livrables conformes aux spécifications, dans le respect des délais, du budget et de la qualité spécifiés ?

Le développement de logiciels était en crise (catastrophe) parce que les méthodes (s'il y en avait) utilisées ne sont pas assez bonnes :

- techniques applicables aux petits systèmes ne pouvant pas être adaptées à large échelle ;
- les principaux projets accusaient parfois beaucoup de retard, et coûtaient beaucoup plus cher que prévu à l'origine ;
- les logiciels développés étaient peu fiables, des résultats médiocres et étaient difficiles à maintenir.

Au cours du développement de logiciels, de nombreux problèmes se posent et cet ensemble de problèmes est connu comme la crise du logiciel. Lorsque le logiciel est en cours d'élaboration, on rencontre des problèmes associés aux étapes de développement.

a) Les Problèmes

Les problèmes rencontrés comprennent :

- les estimations budgétaires et calendaires sont souvent grossièrement inexactes ;
- la «productivité» des personnels du logiciel n'a pas suivi le rythme des exigences de service ;
- la qualité des logiciels est parfois insuffisante ;
- en l'absence d'indication solide de la productivité, nous ne pouvons pas évaluer avec précision l'efficacité des nouveaux outils, méthodes ou normes ;
- la communication entre le client ou l'utilisateur et le développeur de logiciels est souvent médiocre ;
- les tâches de maintenance du logiciel submergent la majorité de tous les fonds de logiciels.

b) Les Causes

On distingue les causes des problèmes suivants :

- La qualité du logiciel n'est pas bonne parce que la plupart des développeurs utilisent des données issues des historiques pour développer le logiciel.
- En cas de retard dans un processus ou à un stade donné (l'analyse, la conception, le codage et les tests), alors les délais ne correspondront pas avec le calendrier préétabli.

- La communication entre les managers et les clients, les développeurs de logiciels, le personnel de support, etc., peut se briser compte tenu du fait que les caractéristiques particulières des logiciels ainsi que les problèmes associés à son développement sont souvent mal compris.
- Les personnes responsables de l'exploitation et de la maintenance des logiciels changent souvent et résistent également au moindre changement introduit.

c) La crise du logiciel du point de vue des programmeurs

Du point de vue des programmeurs, les problèmes comprennent : Problème de compatibilité, problème de la portabilité, problème dans la documentation, problème de piratage de logiciels, problème dans la coordination des travaux des différentes personnes, problème de maintenance ou d'entretien approprié.

d) La crise du logiciel du point de vue des utilisateurs

Du point de vue des utilisateurs les problèmes comprennent : les coûts de logiciels souvent élevés, la détérioration du matériel, le manque de spécialisation dans le développement, problème des différentes versions du logiciel, problème de portée, et le problème de bugs.

e) Les mythes du logiciel

De nombreuses causes des problèmes du logiciel peuvent être attribuées à une mythologie. Mythes (au sens propre du terme) qui ont surgi au tout début de l'histoire du développement des logiciels ; quelques-uns de ces mythes sont :

- Nous avons déjà plusieurs normes et procédures pour la construction de logiciels, cela suffirait il pour fournir tout ce dont on a besoin de savoir ?
- Une déclaration générale des objectifs » est tout ce qu'il faut pour commencer l'écriture de programmes, nous pouvons remplir les détails plus tard.
- Si nous accusons du retard, nous pouvons ajouter plus de programmeurs pour le rattraper.
- Il faut que je commence le codage parce que nous sommes déjà en retard!
- Les exigences du projet changent continuellement, mais les changements peuvent être facilement implémentés parce que le logiciel est flexible.
- Il est impossible d'évaluer la qualité d'un programme jusqu'à ce qu'il livré en production.
- Tout ingénieur compétent peut écrire des programmes.
- Exécutons quelques cas de test, puis nous aurons terminé!
- Je sais ce que le programme fait, je n'ai pas le temps de le documenter. Le seul produit de travail livrable pour un projet réussi est le programme de travail.
- Les gens ne disposent pas d'outils de développement de logiciels à la pointe de l'art. Après tout, nous les achetons les nouveaux ordinateurs !

Conclusion

La solution aux problèmes soulevés au cours du développement du logiciel peut se faire en appliquant une approche d'ingénierie pour le développement de logiciels, qui inclut des procédures pour la planification, le développement, le contrôle de la qualité, la validation et la maintenance. L'approche doit être appliquée de manière cohérente, à travers tous les types de logiciels, d'où le nom "Software Engineering" i.e Génie Logiciel.

Évaluation

1. Définir ce qu'est le logiciel.
2. Quels sont les différents mythes et réalités sur le logiciel ?
3. Quels sont les différentes composantes du logiciel ?
4. Qu'est-ce que la crise du logiciel ? Expliquer les problèmes de la crise du logiciel.
5. Quels sont les mythes de logiciels?
6. Les notés qui se fanent lentement au cours du temps, mais d'autres naissent et prennent leur place. Essayez d'ajouter 1/2 "nouveaux" mythes du logiciel.

Activité 2: Qu'est-ce que le Génie Logiciel

Introduction

Cette activité fournit le concept de l'application des normes d'ingénierie dans le développement des logiciels, d'où le nom "Software Engineering" i.e. « Génie Logiciel ».

Définition

Tel que défini par Pollice, 2005 et IEEE 1993, le génie logiciel est l'application d'une approche disciplinée quantifiable systématique, au développement, à l'exploitation et à la maintenance des logiciels, ainsi que l'étude de ces approches.

Les critères clés pour le Génie Logiciel sont : une méthodologie bien définie, les jalons prévisibles, la traçabilité entre les étapes, la documentation, le contrôle et la maintenabilité. L'objectif du Génie Logiciel touche les théories, les méthodes et les outils qui sont nécessaires pour développer des logiciels.

Ainsi, le Génie Logiciel ne se contente pas seulement de produire un système ou logiciel en bon fonctionnement, mais aussi les documents y afférents tels que les guides d'utilisation, les plans de conception, etc.

Le Génie logiciel traite à la fois le processus de génie logiciel et le produit final. Le bon processus aidera à produire le bon produit, mais le produit désiré influencera également le choix du processus à utiliser.

Principes du Génie Logiciel

Les principes du Génie Logiciel traitent à la fois le processus de génie logiciel et le produit final. Le bon processus aidera à produire le bon produit, mais le produit désiré saura également influer sur le choix du processus à utiliser. Un problème classique en génie logiciel a été l'accent mis sur le processus ou le produit à l'exclusion de l'autre. Les deux sont importants. Pour appliquer les principes, l'ingénieur logiciel doit être équipé avec des méthodes appropriées (directives générales qui régissent l'exécution d'une activité, ils sont précis, systématique, et les approches disciplinés.) et des techniques spécifiques qui aident à incorporer les propriétés souhaitées dans les processus et les produits.

Produits logiciel

Les produits logiciels sont des systèmes logiciels livrés à un client avec la documentation qui décrit comment installer et utiliser le système. Les caractéristiques essentielles d'un produit logiciel comprennent :

Ergonomie : elle doit être utile et utilisable pour améliorer la vie des gens.

Le logiciel est flexible: Un programme peut être développé pour presque tout faire. La caractéristique peut aider à accueillir tout type de changement.

Maintenabilité : devrait être possible de faire évoluer le logiciel pour répondre aux besoins changeants des clients

Sécurité de fonctionnement : comprend une gamme de caractéristiques; la fiabilité, la sécurité et la sureté. logiciel fiable ne devrait pas causer des dommages physiques ou économiques en cas de défaillance du système.

Efficacité : ne devrait pas faire de gaspillage des ressources du système, tels que les cycles de processeur et mémoire.

Évaluation

1. Qu'est-ce que le génie logiciel?
2. Expliquer les caractéristiques du produit logiciel.
3. Comme le logiciel devient plus répandu, les risques pour le public (en raison des programmes défectueux) deviennent une préoccupation de plus en plus importante. Élaborer un scénario apocalyptique réaliste (autre que Y2K) où l'échec d'un programme informatique peut faire beaucoup de mal (économique ou humain).

Activité 3: Qu'est-ce que le Génie Logiciel

Champs d'application des logiciels

Les applications logicielles sont regroupées en huit domaines pour plus de commodité :

Introduction

(A) Logiciel système :

Une collection de programmes utilisés pour exécuter le système comme fonction d'assistance à d'autres programmes logiciels. On peut citer en exemples des compilateurs, des éditeurs, des utilitaires, des composants du système d'exploitation, des pilotes et des interfaces. Ce logiciel réside dans le système informatique et utilise ses ressources.

(B) Logiciel temps réel :

Les logiciels dynamiques concernent souvent un environnement dynamique. Tout d'abord, il recueille l'entrée en analogique et la convertit en un composant numérique, la commande qui répond à l'environnement externe et exécute l'action. Le logiciel est utilisé pour surveiller, contrôler et analyser les événements du monde réel comme ils se produisent. Les éléments du logiciel temps réel comprennent un composant de collecte de données qui collecte et formate les informations d'un environnement externe, un composant d'analyse qui transforme les informations requises par l'application, un composant de commande / sortie qui répond à l'environnement extérieur, et un composant de surveillance qui les coordonne de tous les autres composants de sorte que la réponse en temps réel (allant typiquement de 1 milliseconde à 1 seconde) peut être maintenue.

(C) Logiciel Embarqué :

Un logiciel, lorsqu'il est écrit pour exécuter certaines fonctions dans des conditions de contrôle et est en outre intégré dans le matériel comme une partie intégrante des grands systèmes, est appelé logiciel embarqué. Le logiciel réside dans le Read-Only-Memory (ROM) et est utilisé pour commander les diverses fonctions des produits résidents. Les produits peuvent être une voiture, lave-linge, four micro-ondes, de transformation des produits industriels, des stations de gaz, les satellites, et une foule d'autres produits, là où le besoin est d'acquérir (entrée), analyser, d'identifier le statut, de décider et d'agir.

(D) Logiciels d'entreprise :

Logiciel conçu pour traiter les demandes d'affaires est appelé logiciel de gestion. Le logiciel de gestion peut être une application de données et de traitement de l'information. Il peut piloter le processus d'affaires à travers le traitement des transactions en ligne ou en mode temps réel. Ce logiciel est utilisé pour des opérations spécifiques, tels que les progiciels de comptabilité, les systèmes d'information de gestion, les paquets de paie, et la gestion des stocks.

(E) Logiciels informatiques personnels :

Le traitement de texte, les feuilles de calcul, les applications de l'infographie, du multimédia, de divertissement, de la gestion de bases de données, les applications personnelles et d'affaires financières ne sont que quelques-unes des centaines d'applications personnelles.

(F) Logiciel d'intelligence artificielle :

Le logiciel d'intelligence artificielle utilise des algorithmes non numériques, qui utilisent les données et les informations générées dans le système pour résoudre des problèmes complexes. Ces scénarios de problèmes ne sont généralement pas justiciables des procédures de résolution de problèmes, et nécessitent une analyse et l'interprétation du problème spécifique à résoudre. Les systèmes experts, également appelés systèmes basés sur la connaissance, la reconnaissance des formes (d'image et de la voix), les réseaux de neurones artificiels, théorèmes, et jouer au jeu sont représentatifs des applications au sein de cette catégorie.

(G) Logiciel basé sur le Web :

Le logiciel Web inclut les langues dont les pages Web sont traitées, à savoir, HTML, Java, CGI, Perl, DHTML, etc.

(H) Logiciel scientifique et d'ingénierie :

La conception et l'ingénierie scientifiques offre des logiciels avec les exigences de traitement dans leurs domaines spécifiques. Ils sont écrits pour des applications spécifiques en utilisant les principes et les formules de chaque champ.

Évaluation

1. Donner les différents domaines d'application des logiciels.
2. Expliquer le concept des « logiciels embarqués ».

Processus de développement logiciel

Couches de génie logiciel

Le génie logiciel est une technologie en couches. En se référant à la figure 1.2, toute approche d'ingénierie (y compris l'ingénierie logicielle) doit reposer sur un engagement organisationnel ayant comme pilier la qualité. Le substrat rocheux qui prend en charge l'ingénierie logicielle est un objectif de qualité.



Figure 1.2: Couches du génie logiciel

Pour construire de bons systèmes, nous avons besoin :

Un processus de développement bien défini avec des phases claires d'activités, dont chacune a un produit final. Un processus définit un cadre pour un ensemble de secteurs clés qui doivent être mis en place pour la livraison efficace des technologies de génie logiciel.

Les secteurs clés constituent la base pour le contrôle de la gestion des projets de logiciels et d'établir le contexte dans lequel les méthodes techniques sont appliquées, les produits de travail (modèles, documents, données, rapports, formulaires, etc.) sont produits, les jalons sont établis, la qualité est assurée, et le changement est bien géré.

Méthodes et techniques pour la réalisation des phases d'activité et de modélisation de leurs produits :

Méthodes englobent un large éventail de tâches qui comprennent l'analyse des besoins, la conception, la construction du programme, les tests, et le support. Les méthodes de génie logiciel reposent sur un ensemble de principes de base qui régissent chaque domaine de la technologie et comprennent des activités de modélisation et d'autres techniques descriptives.

Outils pour générer les produits : les outils de génie logiciel offrent un soutien automatisé ou semi-automatisé pour le processus et les méthodes.

Le processus

Le pilier fondamental pour l'ingénierie du logiciel est la couche de processus. Le processus d'ingénierie du logiciel est la colle qui maintient les couches technologiques ensemble et permet un développement rationnel et rapide des logiciels informatiques.

En général, un processus est une série d'étapes impliquant des activités, des contraintes et des ressources qui produisent une sortie destinée quelconque. Tout processus présente les caractéristiques suivantes:

- Le processus prescrit toutes les principales activités du processus.
- Le processus utilise des ressources, sous réserve d'un ensemble de contraintes (comme un calendrier), et fabrique des produits intermédiaires et finaux.
- Le processus peut être composé de sous-processus qui sont liés d'une certaine façon. Le procédé peut être défini comme une hiérarchie de processus, organisée de sorte que chaque sous-processus a son propre modèle de processus.
- Chaque activité de traitement a des critères d'entrée et de sortie, de sorte que nous savons quand l'activité commence et se termine.
- Les activités sont organisées dans une séquence, de sorte qu'il est clair quand une activité est effectuée par rapport aux autres activités.
- Chaque processus a un ensemble de principes directeurs qui expliquent les objectifs de chaque activité.
- Constantes ou contrôles peuvent demander à une activité, une ressource ou produit.

Processus logiciel

Tout le génie est sur la façon de produire des produits dans un processus discipliné. En général, un processus définit qui fait quoi, quand et comment atteindre un certain objectif. Un procédé pour construire un produit logiciel ou d'améliorer un existant est appelé un processus de développement de logiciels. Un processus de développement de logiciels est souvent décrit en termes d'un ensemble d'activités nécessaires pour transformer les besoins d'un utilisateur dans un système logiciel (Figure 1.3).



Figure 1.3: Processus de développement logiciel (Source: Liu et al, 1998)

Les exigences du client définissent l'objectif du développement de logiciels. Ils sont préparés par le client (parfois avec l'aide d'un ingénieur logiciel) pour définir les services attendu pour

le système, à savoir les exigences fonctionnelles. Les exigences fonctionnelles devraient indiquer ce que le système doit faire plutôt que de la façon dont il est fait. Outre les exigences fonctionnelles, un client peut également avoir des contraintes non-fonctionnelles qu'il / elle souhaite placer sur le système, comme le temps de réponse requis ou l'utilisation d'une norme de langage spécifique.

Il y a quatre activités de processus fondamentaux, qui sont communs à tous les processus logiciels. Ces activités sont les suivantes :

A. *Les spécifications logicielles* : La fonctionnalité du logiciel et les contraintes sur son fonctionnement doit être défini.

B. *Le développement de logiciels* : Le logiciel qui répond aux spécifications doit être produit.

C. *La validation du logiciel :* Le logiciel doit être évalué pour s'assurer qu'il fait ce que le client voulait exactement.

D. *L'évolution du logiciel* : Le logiciel doit évoluer pour répondre aux besoins changeants des clients.

E. Notez que les processus logiciels différents organisent ces activités de différentes manières et sont décrits à différents niveaux de détail. Le calendrier des activités varie, tout comme les résultats de chaque activité.

Évaluation

1. Pourquoi avons-nous besoin d'un processus de développement de logiciels.
2. Expliquer en détail le processus de génie logiciel.
3. Décrivez quatre activités de processus fondamentaux utilisés dans les processus logiciels.

Activité 4: Cycle de vie du développement

Le cycle de vie du développement du logiciel (SDLC) est utilisé pour faciliter le développement d'un produit logiciel d'une manière systématique, bien définie, et rentable. Un système d'information passe par une série de phases de la conception à la mise en œuvre. Ce processus est appelé le cycle de vie du développement du logiciel (SDLC) (en anglais : Software Development Life-Cycle). Les diverses raisons d'utiliser un modèle de cycle de vie comprennent :

- Aide à comprendre l'ensemble du processus
- Garantit une approche structurée pour le développement
- Permet la planification des ressources à l'avance
- Permet des contrôles ultérieurs entre eux
- La gestion des aides à suivre les progrès du système

Le SDLC se compose de plusieurs phases et ces phases doivent être identifiées ainsi que la définition des critères d'entrée et de sortie pour chaque phase. Une phase ne peut commencer que lorsque les critères de phase d'entrée correspondantes sont remplis.

De même, une phase peut être considérée comme complète seulement lorsque les critères de sortie correspondants sont satisfaits. S'il n'y a aucune indication claire de l'entrée et de sortie pour chaque phase, il devient très difficile de suivre l'avancement du projet.

Le SDLC peut être divisé en 5-9 phases, à savoir, il doit avoir un minimum de cinq phases et un maximum de neuf phases. En moyenne, il a sept ou huit phases. Celles-ci sont :

a) Reconnaissance de l'initiation et de la planification / projet de besoin / enquête préliminaire

b) l'identification et sélection de projets / Etude de faisabilité

c) L'analyse des projets

d) La conception du système

e) Codage

f) Essais

g) Mise en œuvre

h) Maintenance

a) Reconnaissance de l'initiation et de la planification / projet de besoin / enquête préliminaire

La reconnaissance du besoin permet la définition du problème. Il est la décision sur les problèmes dans le système existant et l'impulsion pour le changement du système. La première étape de tout projet ou SDLC est appelé l'enquête préliminaire. Il est une brève enquête sur le système considéré. A ce stade, la nécessité de changements dans le système existant est identifié et les lacunes du système actuel sont détectées.

b) Étude de faisabilité

Une étude de faisabilité est une étude préliminaire qui enquête sur les besoins d'information des utilisateurs potentiels et détermine les besoins en ressources, les coûts, les avantages et la faisabilité d'un projet proposé. L'objectif des études de faisabilité est d'évaluer les systèmes alternatifs et de proposer les systèmes les plus réalisables et souhaitables pour le développement.

c) Analyse du projet

L'analyse des projets est une étude détaillée des différentes opérations effectuées par un système et leurs relations à l'intérieur et à l'extérieur du système. Une enquête détaillée devrait être menée avec le personnel étroitement impliqué dans le domaine visé par l'enquête, selon les termes de référence précis découlant des rapports d'études initiales. Les tâches à effectuer doivent être clairement définies, telles que :

- Examiner et documenter les aspects pertinents du système existant, ses lacunes et ses problèmes.
- Analyser les résultats et enregistrer les résultats.
- Définir et documenter un système proposé dans les grandes lignes.
- Tester la conception proposée par rapport à des faits connus.
- Produire un rapport détaillé à l'appui des propositions.
- Estimer les ressources nécessaires pour concevoir et mettre en œuvre le système.

Les objectifs à ce stade sont de fournir des solutions aux problèmes posés, généralement sous la forme de spécifications pour répondre aux besoins des utilisateurs et de faire des recommandations pour un nouveau système informatique. L'analyse est un processus itératif et progressif, l'examen des flux d'information et l'évaluation de diverses solutions de conception alternatives jusqu'à ce qu'une solution préférée soit disponible.

d) la conception du système

La conception du système est la phase la plus créative et stimulante du SDLC. La conception décrit le système final et le processus par lequel il est développé. Cette phase est une phase très importante du cycle de vie. Le processus de conception se traduit par des exigences en une représentation du logiciel qui peut être évaluée pour la qualité avant que le codage ne commence. La conception est documentée et le document produit devient ainsi une partie de la configuration logicielle.

e) Codage

L'objectif de la phase de codage consiste à traduire la conception du système dans le code dans un langage de programmation donnée. Dans cette phase, l'objectif est de mettre en œuvre la conception de la meilleure manière possible. Cette phase affecte les deux phases de test et de maintenance. Un code bien écrit peut réduire l'effort de test et de maintenance. Ainsi, au cours du codage l'accent est mis sur le développement de programmes qui sont faciles à lire et à comprendre, et non seulement sur l'élaboration de programmes qui sont simples à écrire.

f) Essais

Les tests sont la principale mesure de contrôle de qualité utilisée au cours du développement de logiciels. Sa fonction essentielle est de détecter les erreurs dans le logiciel. Le but du test est de découvrir les exigences, la conception et les erreurs de codage dans le programme. Le test est une activité extrêmement critique. Elle exige une bonne planification de l'ensemble du processus de test. Lors de l'essai de l'unité, les cas de test spécifiés sont exécutés et les résultats réels sont comparés avec la sortie attendue. Le résultat final de la phase de test est le rapport d'essai et le rapport d'erreur, ou un ensemble de ces rapports (un pour chaque unité testée).

g) Mise en œuvre

La phase de mise en œuvre est moins créative que la conception du système. Il est principalement concerné par la formation des utilisateurs, le choix du site, et la préparation et la conversion de fichiers. Une fois que le système a été conçu, il est prêt à être exécuté. La mise en œuvre concerne les tâches menant immédiatement à un système pleinement opérationnel. Elle implique des programmeurs, des utilisateurs et la gestion des opérations, mais sa planification et le calendrier est une fonction première d'un analyste de systèmes. Il comprend le test final du système complet à la satisfaction de l'utilisateur, et la supervision de la mise en service du système. La mise en œuvre du système comprend également les mécanismes permettant d'assurer la sécurité du système.

h) Maintenance

L'entretien est une partie importante de la SDLC. S'il y a une erreur à corriger ou à modifier, alors il se fait dans la phase d'entretien. La maintenance du logiciel est également une phase nécessaire du développement de logiciels. L'entretien pourrait consommer plus de temps que le temps consommé dans le développement. En outre, le coût d'entretien varie de 50% à 80% du coût total du développement. L'entretien peut être classé comme :

IMaintenance corrective : signifie la réparation des erreurs de traitement ou de performance ou le fait d'apporter des modifications en raison de problèmes précédemment non corrigées.

Maintenance Adaptative : changer la fonction du programme. Ceci est fait pour adapter au changement de l'environnement externe, telles que les nouvelles réglementations gouvernementales. Par exemple, le système actuel a été conçu de sorte qu'il calcule les impôts sur les bénéfices après déduction du dividende sur les actions de participation. Le gouvernement a émis des ordres maintenant pour inclure le dividende dans le bénéfice de l'entreprise pour le calcul de l'impôt. Cette fonction doit être changée pour adapter au nouveau système.

Maintenance Perfective : amélioration de la performance ou modification des programmes pour répondre aux besoins supplémentaires ou au changement d'utilisateur. Il implique des changements que le client pense, permettra d'améliorer l'efficacité du produit, telles que des fonctionnalités supplémentaires ou la diminution du temps de réponse. Comme l'entretien est très coûteux et très essentiel, des efforts ont été faits pour réduire ses coûts. Une façon de réduire les coûts est par la gestion de la maintenance et des audits de modification du logiciel.

Maintenance préventive : La maintenance préventive est le processus par lequel nous empêchons notre système d'être obsolète. La maintenance préventive implique le concept de re-engineering et de l'ingénierie dans lequel un ancien système avec une technologie

Évaluation

1. Discuter le concept de SDLC en bref.
2. Donner les phases de base dans le cycle de vie de développement de logiciels.
3. Quelles sont les différentes étapes du cycle de vie du développement du logiciel ? Quels sont les produits finaux à chaque étape ?
4. Quelles sont les activités importantes qui sont effectuées au cours de la phase d'étude de faisabilité ?
5. Expliquer les différentes catégories de maintenance dans le cycle de vie de développement de logiciels.

Activité 5: Approches de développement de logiciel

Dans les organisations professionnelles, les approches de développement de systèmes suivent généralement l'un des deux modèles de base : le modèle de cascade ou le modèle évolutif.

Le modèle de cascade (Waterfall model

Le modèle de cascade est un modèle très commun dans le processus de développement de logiciels. Le modèle Waterfall est la base de la plupart des méthodes de développement structurées qui sont entrées en usage depuis les années 1970. Il est également connu comme «le modèle linéaire séquentiel». Le modèle linéaire séquentiel suggère une approche systématique, séquentielle pour le développement de logiciels qui commence au niveau du système et progresse à travers l'analyse, la conception, la mise en œuvre, les tests et le support ou l'entretien comme le montre la figure 1.4.

Le modèle Waterfall fournit un cadre pour la planification et le développement de systèmes de haut vers le bas. Les étapes du modèle de cascade se chevaucheent de et se nourrissent des informations les uns aux autres. Lors de la conception, des problèmes avec les exigences sont identifiées; pendant le codage, les problèmes de conception sont trouvés et ainsi de suite. Le processus de développement n'est pas un modèle linéaire simple, mais implique une séquence d'itérations des activités de développement.

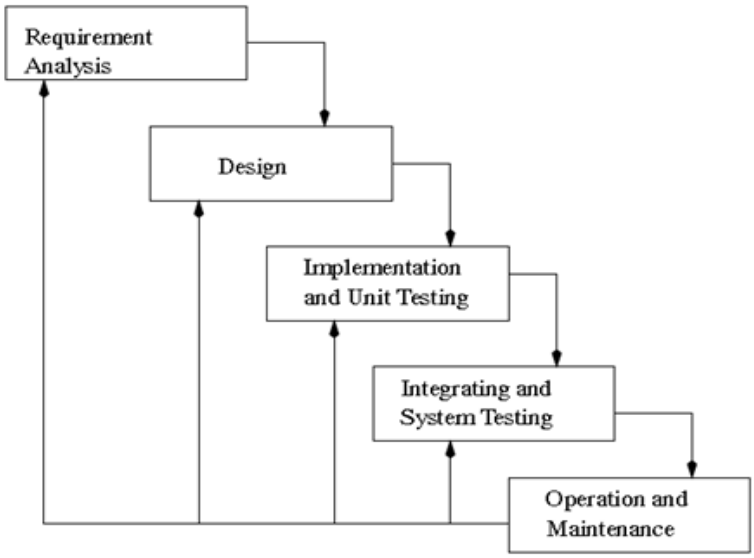


Figure 1.4: Le modèle Waterfall

Avantages du Modèle de cascade

- Les différents avantages du modèle en cascade comprennent :
- Il est un modèle linéaire.
- Il est un modèle segmentaire.
- Il est systématique et séquentielle.
- Il est simple.
- Il dispose d'une documentation appropriée.

Inconvénients du modèle Waterfall

Les divers inconvénients du modèle en cascade comprennent :

- Il est difficile de définir toutes les exigences au début d'un projet.
- Ce modèle ne convient pas pour loger tout changement.
- Une version de travail du système ne se voit pas jusqu'à la fin de la vie du projet.
- Il ne cadre pas bien avec les grands projets.
- Il implique la documentation lourde.
- Nous ne pouvons pas revenir en arrière dans le SDLC.
- Il n'y a pas de modèle d'échantillon pour réaliser clairement les besoins du client.
- Il n'y a pas d'analyse des risques.
- En cas de faute ou une erreur dans une phase alors nous ne pouvons pas faire un bon logiciel.
- Il est un processus axé sur le document qui exige des documents officiels à la fin de chaque phase.

Problèmes avec le modèle Waterfall

Le modèle de cascade est le plus ancien et le paradigme le plus largement utilisé pour le génie logiciel. Parmi les problèmes qui sont parfois rencontrés lorsque le modèle de cascade est appliqué, on peut citer :

I.Les projets réels suivent rarement le flux séquentiel que le modèle propose.

II.Il est souvent difficile pour le client de déclarer toutes les exigences explicitement. Le modèle de cascade exige cela et a de la difficulté à recevoir l'incertitude naturelle qui existe au début de nombreux projets.

III.Le client doit avoir de la patience. Une version de travail du programme (s) ne sera pas disponible avant la fin du projet. Une erreur majeure, inaperçue jusqu'à ce que le programme est mis en production, peut être terrible.

Chacun de ces problèmes est réel. Cependant, le modèle de cascade a une importance capitale dans les travaux de génie logiciel. Ce modèle ne convient que si les exigences sont bien comprises. Il fournit un modèle dans lequel les méthodes d'analyse, la conception, la mise en œuvre, les tests et la maintenance peuvent être incluses. Le modèle de cascade reste un modèle procédural largement utilisé pour le génie logiciel.

Modèles évolutifs de processus logiciel

On reconnaît de plus que le logiciel, comme tous les systèmes complexes, évolue sur une période de temps. Les exigences opérationnelles et de produits changent souvent au cours du développement, ce qui rend un chemin droit à un produit final irréaliste. Le modèle séquentiel linéaire est conçu pour le développement linéaire.

En substance, cette approche en cascade suppose qu'un système complet sera livré après que la séquence linéaire est terminée. Les modèles évolutifs sont itératifs. Ils sont caractérisés d'une manière qui permet aux ingénieurs de logiciel de créer des versions de plus en plus complètes.

Les techniques utilisées dans un développement évolutif comprennent :

Le développement exploratoire où l'objectif du processus est de travailler avec le client pour explorer leurs besoins et de fournir un système final. Le développement commence par les parties du système qui sont compris. Le système évolue en ajoutant de nouvelles fonctionnalités à mesure qu'elles sont proposées par le client.

Prototypage où l'objectif du développement est de comprendre les besoins du client et donc de développer une meilleure définition des besoins pour le système. Le prototype se concentre sur des expériences avec les parties où les exigences du client qui sont mal comprises.

Le développement évolutif est basé sur l'idée de développer une mise en œuvre initiale, exposant ce produit à l'utilisateur et d'affiner ensuite à travers de nombreuses versions jusqu'à ce qu'un système adéquat ait été mis au point. Deux approches évolutives :

- Le modèle incrémental
- Le modèle de la spirale

Le Modèle incrémental

Plutôt que de livrer le système en une seule livraison, le développement et la livraison se décompose en incréments avec chaque partie d'incrémentation fournissant les fonctionnalités requises comme le montre la figure 1.5. Les besoins des utilisateurs sont prioritaires et les exigences les plus prioritaires sont incluses dans les premiers incréments. Une fois que le développement d'un incrément est démarré, les exigences sont gelées même si les exigences pour de futurs incréments peuvent continuer à évoluer.

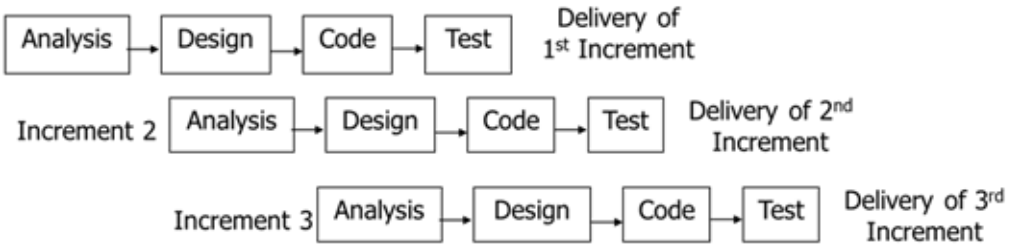


Figure 1.5: Le modèle Incrémental

Le modèle en spirale

Le modèle en spirale est un modèle de processus logiciel évolutif qui couple la nature itérative de prototypage avec les aspects contrôlés et systématiques du modèle séquentiel linéaire. Il fournit le potentiel pour le développement rapide de versions incrémentales du logiciel. En utilisant le modèle en spirale, le logiciel est développé dans une série de livrables supplémentaires.

Le projet est exécuté dans une série de courts cycles de vie, chacun se terminant par une version du logiciel exécutable. Au cours des premières itérations, la libération progressive pourrait être un modèle ou un prototype papier. Au cours des itérations ultérieures, les versions de plus en plus complètes du système d'ingénierie sont produites.

Selon la figure 1.6, un modèle en spirale est divisé en un certain nombre d'activités-cadres, les régions sont également des tâches. Typiquement, il y a entre trois et six régions.

I.*Communication client* : tâches nécessaires pour établir une communication efficace entre les développeurs et le client.

II.*Tâches de planification nécessaires* pour définir les ressources, les échéanciers, et d'autres renseignements liés au projet.

III.*L'analyse des risques* : tâches nécessaires pour évaluer les risques techniques et de gestion.

IV.*Tâches techniques nécessaires* pour construire une ou plusieurs représentations de l'application.

V.*La construction et la production de version* : tâches nécessaires pour construire, tester, installer et fournir un appui de l'utilisateur (par exemple, la documentation et la formation).

VI.*Evaluation client* : tâches nécessaires pour obtenir les commentaires des clients sur la base de l'évaluation des représentations logicielles créées au cours de la phase d'ingénierie et mis en œuvre au cours de la phase d'installation.

Chacune des régions est peuplée par un ensemble de tâches de travail, appelé un ensemble de tâches, qui sont adaptées aux caractéristiques du projet à entreprendre.

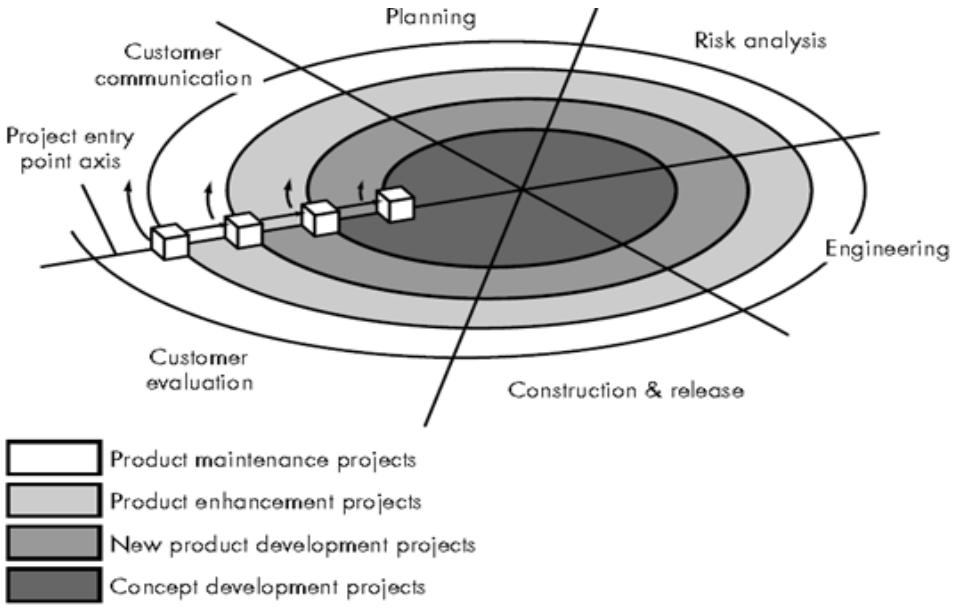


Figure 1.6: Le Modèle Spiral

Problèmes avec le modèle en spiral

Il souffre des problèmes suivants :

- Le processus n'est pas visible. Il est difficile et coûteux de produire des documents qui reflètent toutes les versions du système.
- Les changements continuels
- Les systèmes sont mal structurés ce qui tend à détruire la structure du logiciel.
- Il est toujours possible pour les grands systèmes, mais les changements dans les versions ultérieures sont très difficiles et parfois impossibles. Une nouvelle compréhension et de nouvelles exigences obligent parfois le développeur à recommencer à nouveau l'ensemble du projet. l'évolution du logiciel est donc susceptible d'être difficile et coûteux. Le prototypage fréquent est également très coûteux.

Ces problèmes conduisent directement à d'autres problèmes, comme le fait que le système est difficile à comprendre et à maintenir. Il est suggéré que ce modèle devrait être utilisé dans les circonstances suivantes:

- Le développement de systèmes relativement petits.
- Le développement de systèmes avec une courte durée de vie. Ici, le système est conçu pour soutenir une activité limitée dans le temps, et par conséquent le problème d'entretien n'est pas une question importante.
- Le développement de systèmes ou parties de grands systèmes où il est impossible d'exprimer la spécification détaillée à l'avance

Les idées, les principes et les techniques du processus de développement de l'évolution sont toujours utiles et doivent être utilisés dans les différentes étapes d'un processus de développement plus large, comme la compréhension des exigences et de la validation du processus de cascade.

Évaluation

1. Dessinez le schéma du modèle de cascade du développement de logiciels. Discutez également ses phases en bref.
2. Qu'est-ce qu'un prototype ? Dessinez le schéma du modèle de prototypage du développement de logiciels. Discutez également ses phases en bref.
3. Comment les modèles linéaires et processus itératif diffèrent ?
4. Comparer le modèle en spirale avec le modèle de prototypage en donnant leurs avantages et leurs inconvénients.
5. Y a-t-il jamais un cas où les phases génériques du processus de génie logiciel ne sont pas applicables ? Si oui, décrivez.

Résumé de l'unité

Le Logiciel est devenu l'élément clé dans l'évolution des systèmes et produits informatiques. Les problèmes rencontrés en termes de crise du logiciel et les mythes ont fait que les développeurs de logiciels se sont investis pour appliquer des techniques d'ingénierie dans le développement de logiciels, d'où la discipline du génie logiciel. Le logiciel est composé de programmes, données et documents. Chacun de ces éléments comprend une configuration qui est créée dans le cadre du processus de génie logiciel. Le but du génie logiciel est de fournir un cadre pour la construction de logiciels avec une qualité supérieure.

Cette unité a présenté l'introduction du génie logiciel. Il a expliqué pourquoi la discipline d'ingénierie a été utilisée dans le développement de logiciels après avoir analysé les problèmes rencontrés dans le développement de logiciels. L'unité est allée plus loin et a présenté le processus de développement de logiciels sous la forme d'un cycle de vie de développement des logiciels. Les modèles de processus génériques : le modèle en cascade et les modèles d'évolution ont été décrits.

Évaluation de l'unité

Vérifiez votre compréhension!

1. Qu'entendez-vous par le logiciel
2. Qu'est-ce et des logiciels d'ingénierie?
3. Quels sont les différents mythes et la réalité sur le logiciel?
4. D'un détail expliquer le processus de génie logiciel

Lectures et autres ressources

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783
- Zhiming Liu, (2001), "Object-Otiented Software Development Using UML", The United Nations University, UNU-IIST International Institute for Software Technology, Tech Report 229.

Unité 2. Planification d'un projet logiciel et Analyse et spécifications des besoins

Introduction à l'unité

Cette unité présente les bases pour la production du cahier des charges en utilisant des mécanismes d'ingénierie appliqués au recueil des exigences. Il explique comment comprendre ce que les clients veulent, comment analyser et valider leurs besoins et enfin produire un cahier des charges. L'unité élabore également un ensemble d'activités liées à la planification de projet logiciel. La planification du projet logiciel touche également l'estimation du coût et le calendrier du projet.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de :

- Définir les termes suivants: ingénierie des exigences, Exigence, planification des logiciels
- Décrire les processus adoptés en ingénierie des exigences
- Démontrer le processus de détermination des exigences
- Expliquer le concept de gestion de projet et de ses quatre éléments importants (personnes, produits, processus et projet)
- Décrire le concept de gestion de projet basé sur la recherche de l'estimation des coûts du projet
- Utiliser les étapes guidées de l'estimation pour obtenir le coût du projet.

Termes clés

Exigence: Une exigence est une caractéristique du système ou une description de quelque chose que le système est capable de faire pour remplir l'objectif du système.

L'ingénierie des exigences: L'ingénierie des exigences est l'utilisation systématique des principes éprouvés, des techniques et des outils linguistiques pour l'analyse, la documentation rentable, et l'évaluation des besoins de l'utilisateur et les spécifications du comportement externe d'un système en cours pour satisfaire les besoins des utilisateurs.

Estimation du projet logiciel: L'estimation du projet logiciel est le processus d'estimation des différentes ressources nécessaires à la réalisation d'un projet.

Activités d'apprentissage

Activité 1: Ingénierie des exigences

Introduction

Dans cette activité, la définition de « Besoin » et « ingénierie des exigences » est présentée. Il élabore le processus adopté en ingénierie des exigences qui conduira à l'obtention du cahier des charges du système / logiciel.

Exigence

Une exigence est une caractéristique du système ou une description de quelque chose que le système est capable de faire pour remplir l'objectif du système.

Les exigences décrivent le «quoi» d'un système, et non pas le «comment». L'ingénierie des exigences produit un document volumineux, écrit dans un langage naturel, et contient une description de ce que le système fera sans décrire comment il va le faire. Il fournit le mécanisme approprié pour comprendre ce que le client veut, analyser le besoin, l'évaluation de la faisabilité, la négociation d'une solution raisonnable, en précisant la solution sans ambiguïté, la validation de la spécification et la gestion des exigences.

L'ingénierie des exigences est l'utilisation systématique des principes éprouvés, des techniques et des outils linguistiques pour l'analyse, la documentation rentable, et en cours d'évaluation des besoins de l'utilisateur et les spécifications du comportement externe d'un système pour satisfaire les besoins des utilisateurs. Elle peut être définie comme une discipline, qui répond à des exigences d'objets tout au long d'un processus système de développement.

Comme dans la figure 2.1, l'entrée à l'ingénierie des exigences est l'énoncé du problème préparé par le client. La sortie du processus de l'Ingénierie des Exigences (RE) est une spécification des exigences du système appelée la définition et la description de l'exigence (RDD). Le cahier des charges du système constitue donc la base pour la conception de solutions logicielles.

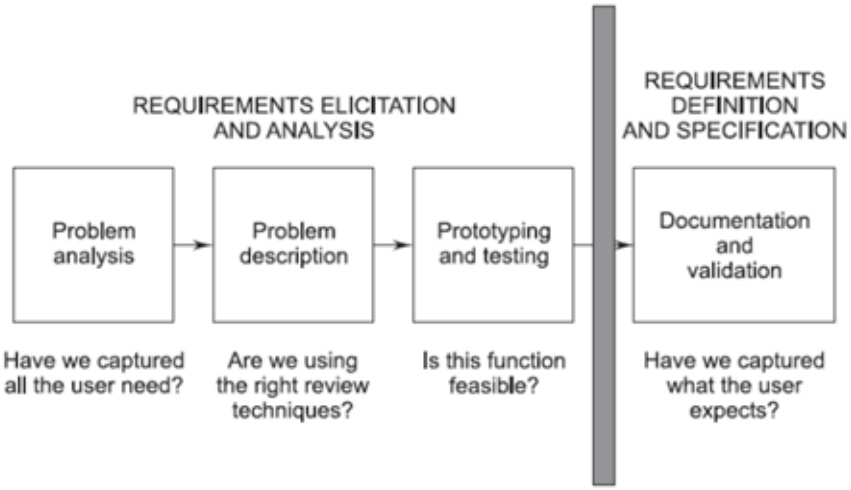


Figure 2.1: Le Processus de recueil des exigences (Source: Agarwal, Tayal and Gupta, 2010)

Types d'exigences

Les exigences sont classées en deux types :

a) Les exigences fonctionnelles :

- Ils définissent des facteurs, tels que les formats E / S, de la structure de stockage, les capacités de calcul, le calendrier et la synchronisation.
- Sont celles qui se rapportent directement au fonctionnement du système.
- Ce sont les aspects du système que le client est le plus susceptible de reconnaître.

b) les exigences non-fonctionnelles :

- Ils définissent les propriétés ou les qualités d'un produit, ainsi que les contraintes / restrictions imposées sur le système.
- Ils comprennent la facilité d'utilisation, l'efficacité, la performance, l'espace, la fiabilité, la portabilité, etc.

Processus d'ingénierie des exigences

La figure 2.2 illustre les étapes du processus d'ingénierie des exigences. L'ingénierie des exigences consiste en les procédés suivants:

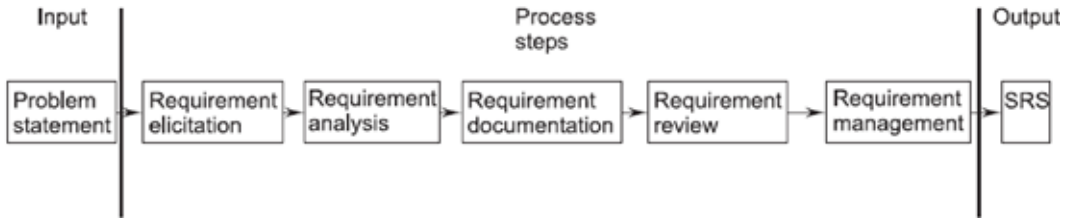


Figure 2.2: Etapes du processus d'ingénierie des exigences(Source: Agarwal, Tayal and Gupta, 2010)

Collecte des exigences

C'est un processus de communication entre les parties concernées et affectées dans la situation du problème. Les sources des informations incluent les clients, les utilisateurs finaux, les utilisateurs primaires, secondaires et les intervenants.

Demandez au client, aux utilisateurs, et à d'autres ce que les objectifs du système ou d'un produit sont, ce qui est à accomplir, comment le système ou le produit s'inscrit dans les besoins de l'entreprise, et enfin, comment le système ou le produit doit être utilisé quotidiennement. Les outils collecte sont des réunions, des interviews, la vidéoconférence, e-mails, et les résultats de l'étude des documents et des faits existants.

Plus de 90% à 95% de la collecte devrait être terminée au stade de l'ouverture, tandis que les 5% restants peuvent être collectés au cours du cycle de vie du développement.

La collecte des exigences est normalement difficile :

Problèmes de portée

La limite du système n'est pas bien définie. Les clients / utilisateurs peuvent spécifier des détails techniques inutiles qui peuvent créer la confusion, plutôt que de clarifier les objectifs généraux du système.

Les problèmes de compréhension

Les clients / utilisateurs ne sont pas complètement sûr de ce qui est nécessaire, une mauvaise compréhension des capacités et des limites de leur environnement informatique, n'ont pas une compréhension complète du domaine du problème, ont du mal à communiquer les besoins de l'ingénieur système, omettent de l'information que l'on croit être «évident», oublient de préciser les exigences qui entrent en conflit avec les besoins des autres clients / utilisateurs, ou spécifient des exigences qui sont ambiguës ou invérifiables.

Les problèmes de volatilité

Les exigences changent au fil du temps

Analyse des exigences et modélisation

Une fois que les exigences ont été recueillies, les produits de travail mentionnés précédemment constituent la base de l'analyse des besoins. L'analyse catégorise les exigences et les organise en sous-ensembles liés; explore chaque exigence en relation avec les autres; examine les exigences de validité, de cohérence, d'omissions, d'ambiguïté; exigences de faisabilité et le rang en fonction des besoins des clients / utilisateurs.

- La validité confirme sa pertinence par rapport aux buts et objectifs et confirme toujours que ce ne soit pas incompatible avec d'autres exigences, mais prend en charge d'autres si nécessaire.
- La faisabilité assure que les intrants nécessaires sont disponibles sans biais ni erreurs, et le support de la technologie est possible pour faire exécuter l'exigence comme une solution

Dans un autre point, l'analyse tente de trouver pour chaque exigence, sa fonctionnalité, les caractéristiques, les installations et la nécessité pour ceux-ci dans des conditions et des contraintes différentes.

- Etats de fonctionnalité "comment atteindre l'objectif de l'exigence."
- Caractéristiques décrivent les attributs de la fonctionnalité, et
- Equipements fournir sa prestation, l'administration et la communication avec d'autres systèmes.

Modèle de processus d'explicitation et de l'analyse

Un modèle du processus générique d'explicitation et de l'analyse est illustré à la figure 2.3. Les activités de processus sont les suivantes :

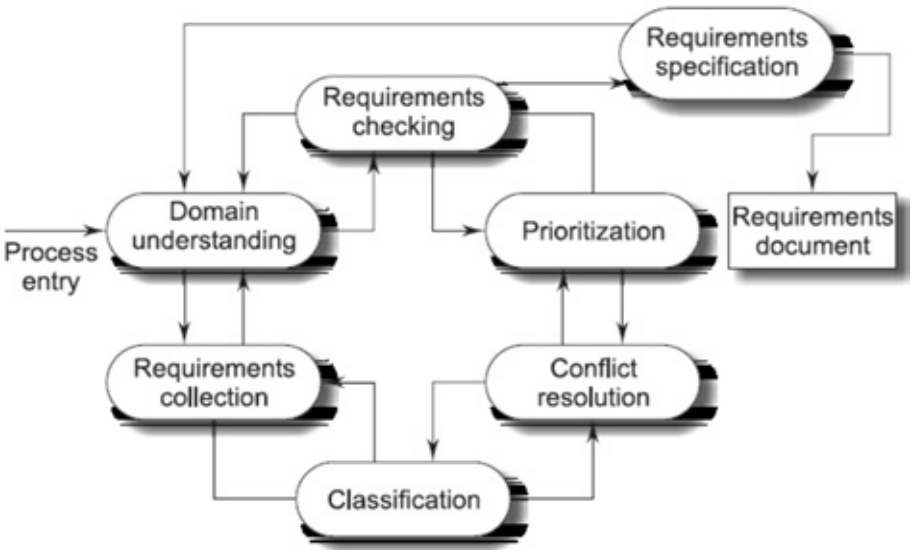


Figure 2.3: Modèle de processus d'explicitation et de l'analyse (Source: Agarwal, Tayal and Gupta, 2010)

Compréhension du domaine : Les analystes doivent développer leur compréhension du domaine d'application.

Collection des exigences : Ceci est le processus d'interaction avec les parties prenantes dans le système pour découvrir leurs besoins. La compréhension de domaine se développe davantage au cours de cette activité.

Classification : Cette activité prend la collecte non structurée des besoins et les organise en grappes cohérentes.

Résolution des conflits : Lorsque plusieurs acteurs sont impliqués, les exigences seront en conflit. Cette activité porte sur les résultats et la résolution de ces conflits.

Priorisation : certaines exigences seront plus importantes que d'autres en termes de priorités. Cette étape implique une interaction avec les parties prenantes pour découvrir les exigences les plus importantes.

Exigences Vérification : Les exigences sont vérifiées pour découvrir si elles sont complètes, cohérentes, et conformément à ce que les intervenants veulent vraiment du système.

Documentation des exigences (Spécification)

Une documentation des exigences est écrite après la collecte des exigences et l'analyse. Ceci est la façon de représenter les exigences dans un format cohérent. Le document constitué est le cahier des charges (qui est appelé Software Requirement Specifications (SRS)).

Le SRS est une spécification pour un produit logiciel particulier, un programme ou un ensemble de programmes qui remplissent certaines fonctions dans un environnement spécifique.

Les exigences doivent être écrites de sorte qu'ils sont significatifs, non seulement pour les clients, mais aussi pour les concepteurs de l'équipe de développement. Le document de définition des besoins décrit ce que le client aimerait avoir :

- Le contour de l'objectif général du système. Les références à d'autres systèmes connexes, des termes et abréviations qui peuvent être utiles sont inclus
- La description de l'arrière-plan et les objectifs de développement du système
- La nouvelle approche proposée pour résoudre le problème par le client, le cas échéant. Décrire une description de l'approche.
- Les caractéristiques détaillées du système proposé, la définition des limites du système et des interfaces à travers elle. Les fonctions du système sont également expliquées. En outre, nous incluons une liste complète des éléments et des classes données et leurs caractéristiques sont incluses.
- L'environnement dans lequel le système fonctionnera. Inclure les exigences pour le soutien, la sécurité et la vie privée et des contraintes matérielles ou logicielles spéciales doivent être précisées.

Un cahier des charges peut être un document écrit, un modèle graphique, un modèle mathématique formel, une collection de scénarios d'utilisation, un prototype, ou toute combinaison de ceux-ci. Certains suggèrent qu'un «modèle standard» devrait être développé et utilisé pour une spécification du système, en faisant valoir que cela conduit à des exigences qui sont présentées d'une manière cohérente et donc plus compréhensible

Validation des exigences

Ceci est un processus manuel qui implique plusieurs lecteurs à la fois : client et personnel pour vérifier le document d'exigences afin de détecter les anomalies et les omissions. La validation des exigences examine la spécification pour s'assurer que toutes les exigences du système ont été déclarées sans ambiguïté; que des incohérences, des omissions et des erreurs ont été détectées et corrigées; et que les produits de travail sont conformes aux normes établies pour le processus, le projet, et le produit.

Un examen ou validation des exigences peut être formel ou informel

- Des examens informels impliquent simplement des discussions au sujet des exigences avec autant d'intervenants du système que possible. De nombreux problèmes peuvent être détectés simplement en parlant du système pour les parties prenantes avant de prendre un engagement à un examen formel.
- Dans un examen formel des exigences, l'équipe de développement doit travailler avec le client. L'équipe d'examen doit vérifier la cohérence de chaque exigence et doit vérifier les exigences dans leur ensemble. L'examen technique formel est le mécanisme de validation des exigences primaires

Gestion des exigences

Les exigences définissent la capacité que la solution logicielle doit fournir et les résultats escomptés qui doivent résulter de son application à des problèmes d'affaires. Afin de générer de telles exigences, une approche systématique est nécessaire, par le biais d'un processus formel de gestion appelé la gestion des exigences.

La gestion des exigences est définie comme une approche systématique pour susciter, organiser et documenter les exigences du système, et un processus qui établit et maintient un accord entre le client et l'équipe de projet sur l'évolution des exigences du système. Il est un ensemble d'activités qui aident l'équipe de projet pour identifier, contrôler et suivre les exigences et les modifications apportées aux exigences à tout moment que le projet avance.

La gestion des exigences commence par l'identification. A chaque exigence est attribué un identifiant unique qui pourrait prendre la forme :

<Type de condition> <exigence #>

Où le type de condition prend des valeurs telles que la condition F = fonctionnel, condition D = données, B = exigence de comportement, I = exigence d'interface, et P = puissance requise.

Conclusion

La partie importante et difficile dans la construction d'un logiciel est de décider précisément ce qu'il faut construire et d'établir les exigences techniques détaillées. Ces exigences sont identifiées par l'obtention de l'information du client. Les exigences sont analysées pour évaluer leur clarté, l'exhaustivité et la cohérence. Enfin, les exigences du système sont gérées pour assurer que les changements sont correctement contrôlés.

Évaluation

Vérifiez votre compréhension!

1. Qu'est-ce que les exigences à long terme ?
2. Expliquer le processus de détermination des exigences pour un système basé sur un logiciel.
3. Discutez de la signification et l'utilisation de l'ingénierie des exigences. Quels sont les problèmes dans la formulation des exigences ?
4. Quelles sont les étapes cruciales de l'ingénierie des exigences ? Discutez avec l'aide d'un diagramme.
5. Expliquer l'importance des besoins. Combien de types d'exigences sont possibles et pourquoi ?
6. Qu'est-ce que la collecte des exigences ? Discutez des deux techniques en détail.

Activité 2: Planification de projet logiciel

Introduction

Cette activité explique brièvement le concept de «gestion de projet» en mettant l'accent plus sur l'estimation des coûts du projet. Il présente quatre de la plupart des composants des besoins en matière de gestion de projet qui sont des personnes, des produits, processus et projet. L'activité décrit également les principales étapes à suivre dans l'estimation des logiciels.

Gestion de projet

La gestion de projet implique la planification, le suivi et le contrôle des personnes, des processus et des événements qui se produisent en tant que logiciel qui évolue d'un concept préliminaire à une mise en œuvre opérationnelle. Construire un logiciel informatique est une entreprise complexe, en particulier si elle implique de nombreuses personnes travaillant sur une période relativement longue. Voilà pourquoi les projets de logiciels doivent être gérés.

Dans la gestion de projet, il est nécessaire de comprendre les quatre P's : personnes, produit, processus et projet.

- Les gens doivent être organisés pour effectuer des travaux de logiciel efficacement.
- La communication avec le client doit se produire de telle sorte que la portée du produit et les exigences sont compris.

Avant qu'un projet ne soit être prévu, les objectifs et la portée devraient être établis, des solutions alternatives doivent être envisagées, et les contraintes techniques et de gestion doivent être identifiées. Sans cette information, il est impossible de définir des estimations raisonnables (et précises) du coût, une évaluation efficace du risque, une ventilation réaliste des tâches du projet, ou un calendrier gérable de projet qui fournit une indication significative du progrès.

- Un processus doit être sélectionné s'il est approprié pour les personnes et le produit.

Un processus logiciel fournit le cadre à partir duquel un plan global pour le développement de logiciels peut être établi. Un petit nombre d'activités du cadre sont applicables à tous les projets de logiciels, quelle que soit leur taille ou leur complexité. Un certain nombre de tâches, les jalons, les produits de travail, et l'assurance de la qualité des points doivent permettre d'adapter les activités cadres aux caractéristiques du projet et aux exigences de l'équipe du projet.

- Le projet doit être planifié en estimant l'effort et le temps du calendrier pour accomplir des tâches de travail: la définition des produits de travail, en établissant des points de contrôle de qualité,

Un projet de logiciel planifié et contrôlé est la façon de gérer la complexité. Afin d'éviter l'échec du projet, un gestionnaire de projet et les ingénieurs logiciels qui construisent le produit doivent éviter un ensemble de signes avant-coureurs communs, comprendre les facteurs critiques de succès qui conduisent à une bonne gestion de projet, et développer une approche de bon sens pour la planification, le suivi et le contrôle du projet.

La gestion de projet logiciel commence par un ensemble d'activités qui sont collectivement appelés planification du projet. Il est nécessaire d'estimer le travail à faire, les ressources qui seront nécessaires, et le temps qui s'écoulera du début à la fin avant de commencer le projet. Chaque fois que les estimations sont faites, nous regardons vers l'avenir en gardant un certain degré d'incertitude.

Objectifs de planification du projet

L'objectif de la planification de projet logiciel est de fournir un cadre qui permet au gestionnaire de faire des estimations raisonnables des ressources, les coûts et le calendrier.

L'objectif de la planification est réalisé par un processus de découverte de l'information qui conduit à des estimations raisonnables. Les activités liées à la planification de projet logiciel comprennent :

a) Détermination de la portée du logiciel

La fonction et la performance attribuées au logiciel lors de l'ingénierie des exigences doivent être évaluées pour établir un périmètre de projet qui est sans ambiguïté et qui est compréhensible à tous les niveaux techniques et de management. Le champ d'application du logiciel décrit les données et le contrôle à traiter, la fonction, la performance, les contraintes, les interfaces, et la fiabilité. Les fonctions décrites dans la déclaration du champ sont évaluées et, dans certains cas affinées pour fournir plus de détails avant le début de l'estimation.

Parce que les deux estimations de coûts et de calendrier sont orientées fonctionnellement, un certain degré de décomposition est souvent utile. Les considérations de performance comprennent des exigences de temps de traitement et de réponse. Les contraintes permettent d'identifier les limites placées sur le logiciel par le matériel externe, la mémoire disponible, ou d'autres systèmes existants.

b) Estimation de la ressource

La Figure 2.4 illustre les ressources de développement comme une pyramide. L'environnement de développement -outils matériels et logiciels- est à la base de la pyramide des ressources et fournit l'infrastructure pour soutenir l'effort de développement. À un niveau supérieur, nous rencontrons des blocs réutilisables de construction de composants logiciels, ce qui peut considérablement réduire les coûts de développement et accélérer la livraison. Au sommet de la pyramide est le principal personnes-ressources.

Chaque ressource est spécifiée avec quatre caractéristiques: description de la ressource, une déclaration de disponibilité, de temps lorsque la ressource sera nécessaire; la durée pendant laquelle la ressource sera appliquée. Les deux dernières caractéristiques peuvent être considérées comme une fenêtre de temps. La disponibilité de la ressource pour une fenêtre spécifiée doit être établie le plus tôt possible.

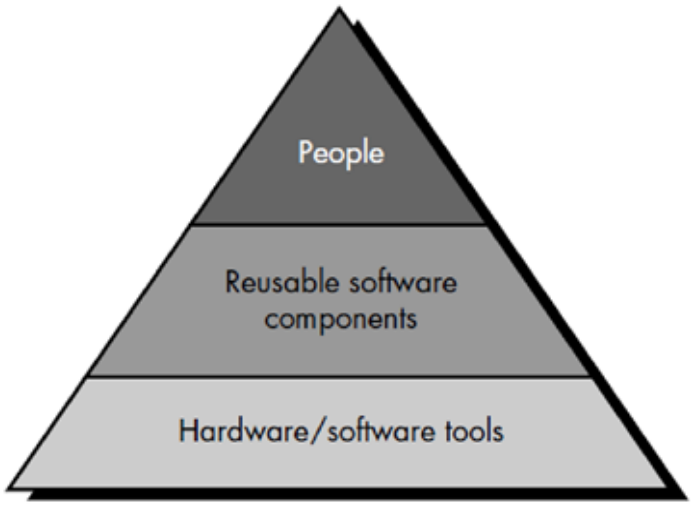


Figure 2.4: Ressources du projet (Source: Pressman, 2001)

c) Estimation du projet logiciel

L'estimation efficace du projet logiciel est l'une des activités les plus difficiles et les plus importantes dans le développement de logiciels. La planification du projet et le contrôle adéquat est impossible sans une estimation fiable.

Estimation des coûts du Projet

L'estimation établit une base pour toutes les activités de planification du projet et fournit donc la feuille de route pour l'ingénierie logicielle. La sous-estimation des projets logiciels et l'insuffisance des effectifs, conduisent souvent à des livrables de faible qualité, hors délais, conduisant ainsi à l'insatisfaction des clients et la perte de crédibilité des développeurs.

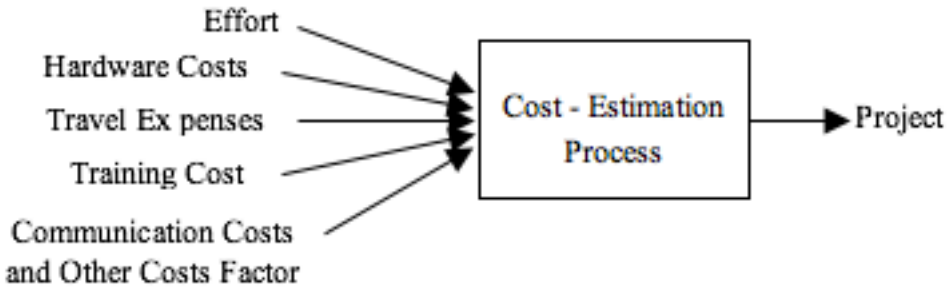


Figure 2.6: Processus d'estimation des coûts
(Source : Agarwal Tayal and Gupta, 2010)

L'estimation comprend principalement les étapes suivantes :

a) Estimation de la taille du projet

Il existe de nombreuses procédures disponibles pour estimer la taille d'un projet, qui sont basées sur des approches quantitatives, telles que l'estimation de lignes de code ou des exigences de fonctionnalité du projet.

b) Estimation des efforts basée sur personne/mois ou personne/heures

Personne/mois est une estimation des ressources personnelles nécessaires pour le projet.

c) L'estimation de l'annexe au calendrier Jours / Mois / Année basée sur le total Personnes/mois nécessaires et la main-d'œuvre affectée au projet

La durée dans le calendrier mois = Total personnel affecté / Total des personnes/mois.

d) L'estimation du coût total de projet

Dans un environnement commercial et concurrentiel, l'estimation des projets logiciels est cruciale pour la prise de décisions de gestion.

Le tableau 2.1 présente la relation entre les différentes fonctions de gestion et des mesures de logiciels / indicateurs.

Table 2.1: Software-Project Estimation (Source: Agarwal, Tayal and Gupta, 2010)

Activity	Tasks Involved
Planning	Cost estimation, planning for training of manpower, project scheduling, and budgeting the project.
Controlling	Size metrics and schedule metrics help the manager to keep control of the project during execution.
Monitoring/Improving	Metrics are used to monitor progress of the project and wherever possible sufficient resources are allocated to improve it.

Estimation de la taille

L'estimation de la taille du logiciel à développer est la première étape pour faire une estimation efficace du projet. Les exigences des clients et les spécifications du système forment une base pour estimer la taille du logiciel. A un stade ultérieur du projet, les documents de conception de systèmes peuvent fournir des détails supplémentaires pour estimer la taille globale du logiciel.

Estimation de l'effort

Une fois que la taille du logiciel est estimée, l'étape suivante consiste à estimer l'effort en fonction de la taille. L'estimation de l'effort peut être faite à partir des spécificités organisationnelles du cycle de vie de développement des logiciels. En fonction des besoins livrables, l'estimation de l'effort pour un projet variera.

Estimation du calendrier

La prochaine étape dans le processus d'estimation est l'estimation du calendrier de l'effort estimé du projet. Le calendrier d'un projet dépendra généralement des ressources humaines impliquées dans un processus. Les efforts en mois/personnes sont convertis en mois calendaires.

Horaire estimation en mois de calendrier peut être calculée selon le modèle suivant [McConnell] :

Horaire en mois de calendrier = 3,0 * (mois/personnes) 1/3

Lorsque le paramètre 3.0 est variable, utilisés en fonction de la situation qui fonctionne le mieux pour l'organisation.

Estimation du coût

L'estimation des coûts est la prochaine étape pour les projets. Le coût d'un projet provient non seulement des estimations de l'effort et la taille, mais d'autres paramètres, tels que le matériel, les frais de déplacement, frais de télécommunication, les coûts de formation, etc. Figure 2.6 et Figure 2.7 illustre le processus coût-estimation.

Raisons de mauvaise estimation

Voici quelques-unes des raisons de la mauvaise estimation :

- Les exigences sont imprécises. En outre, les exigences changent fréquemment.
- Le projet est nouveau et est différent des projets antérieurs traités.
- La non-disponibilité de suffisamment d'informations sur les projets passés.
- Les estimations sont obligées de se fonder sur les ressources disponibles.
- Compromis de coûts et de temps.

Les problèmes associés aux estimations :

- Taille de l'estimation est souvent ignorée et un calendrier est estimé.
- Taille de l'estimation est peut-être l'étape la plus difficile, ce qui a une incidence sur toutes les autres estimations.
- De bonnes estimations ne sont que des projections et sont assujetties à divers risques.
- Les organisations donnent souvent moins d'importance à la collecte et à l'analyse des données historiques de projets de développement passés. Les données historiques sont la meilleure entrée pour estimer un nouveau projet.
- Les gestionnaires de projet sous-estiment souvent le calendrier parce que la direction et les clients hésitent souvent à accepter un calendrier réaliste.

Lignes directrices de l'estimation du projet

Quelques lignes directrices pour l'estimation du projet sont les suivants :

- Préserver et les données du document se rapportant à des projets antérieurs.
- Prévoyez suffisamment de temps pour l'estimation du projet, en particulier pour les grands projets.
- Préparer des estimations réalistes. Les gens associés qui travailleront sur le projet pour atteindre une estimation réaliste et plus précise.
- Utiliser des outils logiciels d'estimation.
- Re-estimer le projet au cours du cycle de vie du processus de développement.
- Analyser les erreurs du passé dans l'estimation des projets.

Conclusion

L'un des problèmes affiliés dans le développement de logiciels est de sous-estimer les ressources nécessaires pour un projet. L'élaboration d'un plan de projet pratique est essentielle pour mieux comprendre les ressources nécessaires, et comment celles-ci doivent être appliquées.

Évaluation

Vérifiez votre compréhension!

1. Pourquoi l'estimation des coûts joue un rôle important dans le processus de développement de logiciels ?
2. Décrire clairement les processus pour l'estimation du projet.
3. Expliquez le terme « Estimation de l'Effort » en vue de l'estimation des coûts du projet logiciel
4. Quelles sont les différentes raisons de la mauvaise estimation ?

Résumé de l'unité

Le planificateur de projet logiciel doit estimer trois choses avant le début d'un projet : combien de temps il faudra, combien d'efforts seront nécessaires, et combien de personnes seront impliquées. L'unité a présenté l'ingénierie des exigences et son processus pour obtenir le cahier des charges. Elle a également présenté la planification de projet logiciel et plus particulièrement l'estimation des coûts du projet.

Évaluation de l'unité

Vérifiez votre compréhension!

1. Qu'entendez-vous par le logiciel
2. Qu'est-ce et des logiciels d'ingénierie?
3. Quels sont les différents mythes et la réalité sur le logiciel?
4. D'un détail expliquer le processus de génie logiciel

Lectures et autres ressources

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Zhiming Liu, (2001), "Object-Otiented Software Development Using UML", The United Nations University, UNU-IIST International Institute for Software Technology, Tech Report 229.

Unité 3. Conception de logiciels

Introduction à l'unité

Le but de la conception du logiciel est de produire un modèle ou une représentation d'une entité qui sera construit plus tard. Il peut être attribué à des exigences d'un client et en même temps évalué pour la qualité d'un ensemble de critères prédéfinis pour « bon pour design ». Cette unité définit ce qu'est une conception de logiciels. Il décrit les objectifs de conception et les principes de conception. Il précise en outre des processus logiciels de conception qui comprennent : conception architecturale, conception abstraite, conception User-Interface, conception bas niveau. La conception orientée-fonction, la conception orientée objet, la notation de la conception, la spécification et la vérification de la conception vont être présentées.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de :

- Décrire ce qui est la conception de logiciels
- Expliquer les objectifs et les principes de conception de logiciels
- Distinguer et décrire les phases / processus de conception de logiciels
- Décrire l'importance de chaque phase / processus de conception de logiciels
- Distinguer les approches "orientées objet" et "Functional-Oriented"
- Élaborer des spécifications de conception et de vérifier la conception.

Termes clés

Conception de logiciels: La conception de logiciels est la pratique de prendre une spécification du comportement observable de l'extérieur et en ajoutant les détails nécessaires à la mise en œuvre effective du système informatique, y compris l'interaction humaine, la gestion des tâches, et les détails de gestion des données.

Principes de conception: La conception de logiciels est à la fois un processus et un modèle. Le processus de conception est une séquence d'étapes qui permettent au concepteur de décrire tous les aspects du logiciel à construire.

Activités d'apprentissage

Activité 1: Conception de logiciel ou Software Design

Introduction

La conception de logiciels se trouve au noyau technique du génie logiciel et est appliqué quel que soit le modèle de processus de logiciel qui est utilisé. Commençant une fois que les exigences de logiciels ont été analysées et précisées, la conception de logiciels est la première des trois techniques des activités à la conception, la génération de code et de test qui sont nécessaires pour construire et vérifier le logiciel. Chaque activité transforme l'information d'une manière qui aboutit finalement à un logiciel informatique validé. ahier des charges du système / logiciel.

Définition de la conception des logiciels

Agarwal et al., (2010) se réfère à la définition de la conception de logiciels comme indiqué par Coad et Yourdon qui se veut être la pratique de prendre une spécification du comportement observable de l'extérieur et en ajoutant les détails nécessaires à la réelle mise en œuvre du système informatique, y compris l'interaction humaine, la gestion des tâches, et les détails de gestion des données.

L'entrée à la conception du logiciel comprend une compréhension des exigences, des contraintes environnementales et des critères de conception, alors que la sortie de l'effort de conception se compose des éléments suivants :

- la conception de l'architecture, qui montre comment les pièces sont interdépendantes
- Spécifications pour toutes les nouvelles pièces
- Définitions pour les nouvelles données

Lors de la conception de logiciels, la spécification des exigences logicielles permet de nourrir la tâche de conception.

La conception de données transforme le modèle de domaine de l'information créée lors de l'analyse en structures de données qui seront nécessaires pour mettre en œuvre le logiciel.

La conception architecturale définit la relation entre les principaux éléments structurels du logiciel, les « modèles de conception » qui peuvent être utilisés pour atteindre les exigences qui ont été définis pour le système, et les contraintes qui affectent la façon dont les modèles de conception architecturale peuvent être appliqués [SHA96].

La représentation, la conception architecturale d'un cadre sur ordinateur du système peut être dérivée de la spécification du système, le modèle d'analyse, et l'interaction des sous-systèmes définis dans le modèle d'analyse.

La conception d'interface décrit comment le logiciel communique en lui-même, avec des systèmes qui interagissent avec elle, et avec les humains qui l'utilisent.

Une interface implique un flux d'informations (par exemple, des données et / ou le contrôle) et un type de comportement.

La conception de niveau composant transforme les éléments structuraux de l'architecture logicielle dans une description de la procédure de composants logiciels.

Lors de la conception, nous prenons des décisions qui pourront finalement affecter le succès de la construction de logiciels et, tout aussi important, la facilité avec laquelle le logiciel peut être maintenu. L'importance de la conception de logiciels peut s'affirmer avec un seul mot : qualité. Le design est le lieu où la qualité est favorisée en génie logiciel. La conception nous donne des représentations de logiciels qui peuvent être évaluées pour la qualité. Le design est la seule façon dont nous pouvons traduire fidèlement les exigences d'un client dans un produit logiciel fini. La conception de logiciels sert de base pour tous les ingénieurs de logiciels. Sans la conception, nous risquons de construire un système instable :

- Un système qui va échouer lorsque de petites modifications seront apportées;
- Un système qui peut être difficile à tester;
- Un système dont la qualité ne peut pas être évaluée avant la fin du processus logiciel, lorsque le temps est court et que beaucoup d'argent a déjà été dépensé.

Conception Objectifs / Propriétés

Voici quelques-unes des propriétés ou des objectifs de la conception du logiciel :

a) Exactitude

La conception d'un système est correcte si le système construit précisément selon la conception satisfait aux exigences de ce système. De toute évidence, l'objectif lors de la phase de conception est de produire des dessins corrects. L'objectif devrait se concentrer à trouver la meilleure conception possible dans les limites imposées par les exigences et l'environnement physique et social dans lequel le système fonctionnera.

b) Vérifiabilité

La conception doit être correcte et doit être vérifiée pour l'exactitude. La Vérifiabilité concerne la facilité avec laquelle la justesse de la conception peut être vérifiée. Diverses techniques de vérification doivent être facilement appliquées à la conception.

c) Exhaustivité

L'Exhaustivité exige que toutes les différentes composantes de la conception doivent être vérifiées, à savoir, toutes les structures de données pertinentes, des modules, des interfaces externes, et les interconnexions de modules sont spécifiées.

d) Traçabilité

La traçabilité est une propriété importante qui permet d'obtenir la vérification de la conception. Elle exige que l'ensemble de l'élément de conception soit conforme à ces exigences.

e) Efficacité

L'efficacité de tout système est concernée par l'utilisation de ressources limitées par le système. Le besoin d'efficacité se pose en raison de considérations de coût. Si certaines ressources sont rares et coûteuses, il est souhaitable que ces ressources soient utilisées efficacement. Dans les systèmes informatiques, les ressources qui sont le plus souvent prises en considération pour l'efficacité sont le temps de traitement et la mémoire. Un système efficace consomme moins de temps processeur et de la mémoire.

f) Simplicité

La simplicité est une des critères de qualité les plus importantes pour les systèmes logiciels. L'entretien d'un système de logiciel est en général assez coûteux. La conception du système est l'un des facteurs les plus importants qui affectent la maintenabilité du système.

Principes de conception

La conception de logiciels est à la fois un processus et un modèle. Le processus de conception est une séquence d'étapes qui permettent au concepteur de décrire tous les aspects du logiciel à construire. Il est important de noter, cette compétence créative, l'expérience passée, un sens de ce qui fait «bon» logiciel, et un engagement global de la qualité sont des facteurs critiques de succès pour une conception compétente.

Comme les plans d'un architecte pour une maison, le modèle de conception commence par représenter la totalité de la chose à construire (par exemple, un rendu en trois dimensions de la maison) et lentement affine la chose à fournir des orientations pour la construction de chaque détail (par exemple, la plomberie mise en page). De même, le modèle de conception qui est créé pour le logiciel offre une variété de points de vue différents du logiciel informatique.

Les principes de conception de logiciels tels que stipulés par Agarwal et al, 2010, comprennent :

(A) Le processus de conception ne doit pas souffrir de la « vision du tunnel ».

Un bon concepteur devrait envisager d'autres approches, à en juger chacun sur la base des exigences du problème, les ressources disponibles pour faire le travail, et les concepts de conception.

(B) La conception doit être traçable au modèle d'analyse

Parce qu'un seul élément du modèle de conception retrace souvent à de multiples exigences, il est nécessaire d'avoir un moyen de suivre la façon dont les exigences ont été satisfaites par le modèle de conception.

(C) La conception ne doit pas réinventer la roue

Les systèmes sont construits en utilisant un ensemble de modèles de conception, dont beaucoup ont probablement été rencontrés auparavant. Ces modèles devraient toujours être choisis comme une alternative à la réinvention. Le temps est court et les ressources sont limitées! Le temps de conception devrait être investi dans la représentation vraiment de nouvelles idées et de l'intégration de ces modèles dans ceux qui existent déjà.

(D) La conception doit « réduire au minimum la distance intellectuelle » entre le logiciel et le problème tel qu'il existe dans le monde réel.

Autrement dit, la structure de la conception du logiciel devrait (si possible) imiter la structure du domaine du problème.

(E) La conception doit présenter l'uniformité et l'intégration

Une conception est uniforme si elle apparaît qu'une personne a développé la chose entière. Règles de style et le format doivent être définis pour une équipe de conception avant le début des travaux de conception. Une conception est intégrée si l'on prend soin de définir les interfaces entre les composants de conception.

(F) La conception doit être structurée de manière à tenir compte du changement

Les concepts abordés dans la section suivante permettent une conception pour réaliser ce principe.

(G) La conception doit être structurée de manière à se dégrader légèrement, même lorsque les données aberrantes, des événements ou des conditions de fonctionnement sont rencontrées

Eh bien un logiciel bien conçu ne devrait jamais être une "bombe". Il devrait être conçu pour tenir compte des circonstances inhabituelles, et si elle doit mettre fin à la transformation, le faire d'une manière gracieuse.

(H) La conception n'est pas codée, le codage est pas conçu

Même lorsque les conceptions de procédure détaillées sont créées pour les composantes du programme, le niveau d'abstraction du modèle de conception est plus élevé que le code source. Les seules décisions de conception au niveau de l'adresse de codage les détails de mise en œuvre de petits qui permettent la conception de la procédure à coder.

(I) La conception doit être évaluée pour la qualité car il est en cours de création, et non après le fait

Une variété de concepts de conception et des mesures de conception sont disponibles pour aider le concepteur dans l'évaluation de la qualité.

(J) La conception doit être revue pour minimiser les erreurs conceptuelles (sémantiques)

Il y a parfois une tendance à se concentrer sur des détails lorsque la conception est examinée, à côté de la forêt pour les arbres. Une équipe de conception doit veiller à ce que les éléments conceptuels majeurs de la conception (omissions, d'ambiguïté et d'incohérence) soient traités avant de se soucier de la syntaxe du modèle de conception.

Lorsque ces principes de conception sont correctement appliqués, l'ingénieur logiciel crée un design qui présente des facteurs externes et internes de qualité [MEY88]. Les facteurs de qualité externes sont les propriétés du logiciel qui peut être facilement observée par les utilisateurs (par exemple, la vitesse, la fiabilité, l'exactitude, l'utilisabilité).

Les facteurs de qualité internes sont importants pour les ingénieurs logiciels. Ils conduisent à une conception de haute qualité du point de vue technique. Pour atteindre des facteurs de qualité internes, le concepteur doit comprendre les concepts de conception de base.

Selon Pressman, (2001), il y a trois principes de conception :

- Problème partitionnement
- Abstraction
- La conception Top-down et la conception Bottom-up

Problème partitionnement

Lors de la résolution d'un petit problème, tout le problème peut être abordé à la fois. Mais pour la résolution de problèmes plus importants, le principe de base est le principe de test « diviser pour régner ». Ce principe suggère de diviser le problème en petits morceaux, de sorte que chaque morceau peut être conquis séparément. Pour la conception de logiciels, par conséquent, l'objectif est de diviser le problème en petits morceaux qui peuvent être résolus séparément.

Abstraction

Une abstraction d'un composant décrit le comportement externe de ce composant, sans se soucier des détails internes qui produisent le comportement. L'abstraction est un élément crucial du processus de conception et il est essentiel pour le problème de partitionnement. Le cloisonnement est essentiellement l'exercice pour déterminer les composants d'un système. Toutefois, ces composants ne sont pas isolés les uns des autres, mais interagissent avec d'autres composants. Afin de permettre au concepteur de se concentrer sur un seul composant à la fois, l'abstraction d'autres composants est utilisée.

L'abstraction est utilisée pour les composants existants, ainsi que pour les composants qui sont nouvellement conçus. L'abstraction des composants existants joue un rôle important dans la phase de maintenance.

L'objectif de base de la conception du système est de spécifier les modules dans un système et leurs abstractions. Une fois les différents modules sont spécifiés, au cours de la conception détaillée le concepteur peut se concentrer sur un seul module à la fois. La tâche dans la conception et la mise en œuvre détaillée consiste essentiellement à mettre en œuvre les modules de sorte que les caractéristiques abstraites de chaque module sont réunies.

Design Top-down et Design Bottom-up

Le système est constitué de composants qui ont des composants qui leur sont propres, par conséquent, un système est une hiérarchie de composants. Les composants de niveau le plus élevé correspond à l'ensemble du système. Pour concevoir de telles hiérarchies, il y a deux approches possibles : top-down et bottom-up.

L'approche top-down commence à partir de la composante de plus haut niveau de la hiérarchie et procède par à des niveaux inférieurs. Par contre,

Une approche bottom-up commence avec le composant de niveau le plus bas de la hiérarchie et produit grâce à des niveaux progressivement plus élevés au composant de niveau supérieur.

a) Approche Top-Down

Une approche de conception descendante commence par identifier les principaux composants du système, les décomposer en leurs composants de niveau inférieur et itérer jusqu'à ce que le niveau de détail souhaité est atteint (voir la figure 3.1).

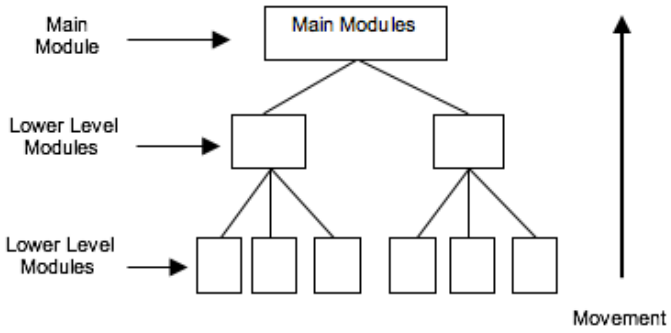


Figure 3.2: Approche Down-Up

Les méthodes de conception Top-down se traduisent souvent par une certaine forme de raffinement par étapes. A partir d'une conception abstraite, chaque étape de la conception est raffinée à un niveau plus concret, jusqu'à ce qu'on atteigne un niveau où il n'y a plus de nécessité de raffinement et où la conception peut être implémentée directement. L'approche top-down a été promulguée par de nombreux chercheurs et a été trouvé pour être extrêmement utile pour la conception. La plupart des méthodes de conception sont basées sur l'approche top-down.

b) L'approche Bottom-Up

Une approche de conception bottom-up comme le montre la Figure 3.2 commence avec la conception des composants et des produits les plus élémentaires ou primitives à des composants de plus haut niveau qui utilisent ces composants de niveau inférieur. Les méthodes

ascendantes fonctionnent avec des couches d'abstraction. En partant du bas, les opérations qui fournissent une couche d'abstraction sont mises en œuvre. Les opérations de cette couche sont ensuite utilisées pour mettre en œuvre des opérations plus puissantes et encore une couche d'abstraction plus élevée, jusqu'à ce que le stade est atteint lorsque les opérations soutenues par la couche sont celles souhaitées par le système.

Remarque :

Une approche top-down ne convient que si les spécifications du système sont clairement connues et le développement du système est à partir de zéro.

Cependant, l'approche ascendante est utilisée si un système à construire est à partir d'un système existant. En effet, il commence à partir de certains composants existants. Ainsi, par exemple, si un type d'amélioration itérative du processus est suivi, dans les itérations ultérieures, l'approche bottom-up pourrait être plus appropriée (dans la première itération une approche top-down peut être utilisée).

Évaluation

Vérifiez votre compréhension!

1. Quelle est la conception du système?
2. Expliquer, en détail, les trois principes de conception dans la conception du système.
3. Quelle est l'abstraction ? Quelles sont les mesures de vérification pour la conception du système?
4. Définir :

    - Partitionnement de problème
    - Abstraction
    - Conception Top-down et la conception de bas en haut

Activité 2: Le processus de conception

Introduction

La conception de logiciels est un processus itératif par lequel les exigences sont converties en un «modèle» pour la construction du logiciel. Initialement, le plan représente une vue holistique du logiciel. Autrement dit, la conception est représentée à un haut niveau d'abstraction d'un niveau qui peut être directement tracée au système spécifique des données objectives et plus détaillées, fonctionnelle, et les exigences de comportement.

Comme les itérations de conception se produisent, le raffinement ultérieur conduit à concevoir des représentations à des niveaux beaucoup plus faibles de l'abstraction. Les moyens que les morceaux d'un problème sont résoluble séparément; le coût de résoudre le problème dans son ensemble est supérieure à la somme du coût de résoudre tous les morceaux. Cependant, les différentes pièces ne peuvent être totalement indépendants les uns des autres qu'ils forment ensemble le système. Les différentes pièces doivent coopérer et à communiquer pour résoudre le problème plus vaste.

La figure 3.3 montre le modèle général du processus de conception de logiciels.

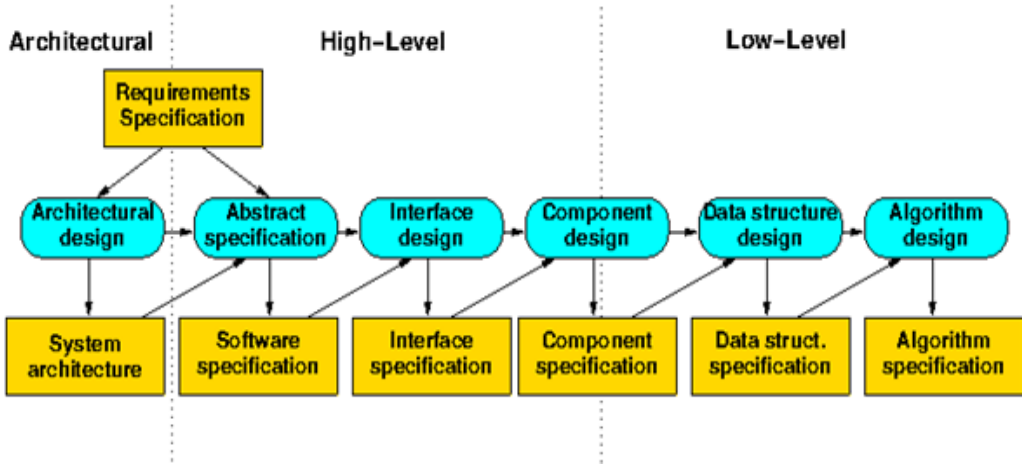


Figure 3.3: Modèle général du processus de conception de logiciels

Conception architecturale

Les grands systèmes sont toujours décomposés en sous-systèmes qui fournissent un certain ensemble connexe de services. Le processus de conception initiale de l'identification de ces sous-systèmes et d'établir un cadre pour le contrôle du sous-système et la communication est appelée conception architecturale.

La conception architecturale représente la structure de données et de programmes composants qui sont nécessaires pour construire un système informatique. Il considère le style architectural que le système prendra, la structure et les propriétés des composants qui constituent le système, et les interrelations qui se produisent entre toutes les composantes architecturales d'un système.

L'architecture n'est pas le logiciel opérationnel. Au contraire, il est une représentation qui permet à un ingénieur logiciel à :

- Analyser l'efficacité de la conception pour répondre à ses besoins déclarés,
- Envisager des alternatives architecturales à un stade où des changements de conception est encore relativement facile,
- Réduire les risques associés à la construction du logiciel.

Les méthodes de conception d'architecture ont un coup d'œil dans différents styles architecturaux pour la conception d'un système. Ceux-ci sont :

Architecture centrée sur les données

L'architecture centrée sur les données implique l'utilisation d'une opération de base de données centrale d'insertion et de mise à jour sous la forme d'une table. Un magasin de données (par exemple, un fichier ou base de données) réside au centre de cette architecture et est accessible fréquemment par d'autres composants que la mise à jour, ajouter, supprimer ou modifier des données dans le magasin.

La figure 3.4 illustre un style typique de données centrée. Le logiciel client accède à un référentiel central. Dans certains cas, le référentiel de données est passif. Autrement dit, le logiciel client accède à des données indépendantes de toute modification des données ou les actions d'un autre logiciel client.

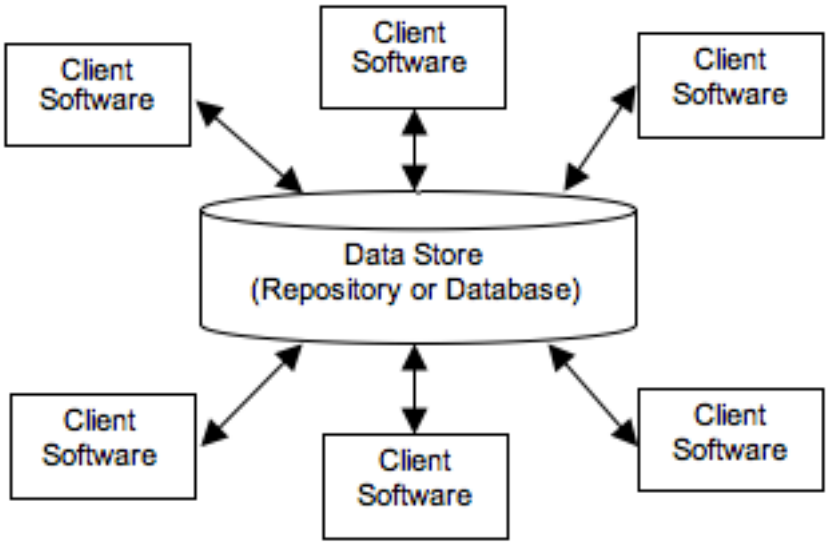


Figure 3.4: Style typique de données centrée

Architecture de flux de données (Data-flow)

L'architecture de flux de données est un élément central dans le mécanisme de tuyau et le filtre. Cette architecture est appliquée lorsque des données d'entrée prennent la forme de sortie après passage à travers les différentes phases de transformation. Ces transformations peuvent se faire par des manipulations ou différents calculs effectués sur les données.

Architecture orientée objet

Dans l'architecture orientée objet la conception du logiciel se déplace autour des clauses et des objets du système. La classe encapsule les données et les méthodes. Les composants d'un système d'encapsuler des données et des opérations qui doivent être appliquées pour manipuler les données. La communication et la coordination entre les composants est accomplie par l'intermédiaire d'un message qui passe.

Architecture en couche

L'architecture en couches définit un certain nombre de couches et chaque couche exécute des tâches. La couche la plus extérieure gère toutes les fonctionnalités de l'interface utilisateur et la couche la plus interne occupe principalement l'interaction avec le matériel.

La structure de base d'une architecture en couches est illustrée à la figure 3.5. Un certain nombre de différentes couches sont définies, chacune des opérations accomplissant qui deviennent progressivement plus près du jeu d'instructions de la machine. A la couche externe, le fonctionnement de l'interface utilisateur de service des composants. À la couche interne, les composants exécutent le système d'exploitation d'interface (couche de base). Les couches intermédiaires fournissent des services d'utilité (couche utilitaire) et le logiciel d'application (couche d'application) fonctions.

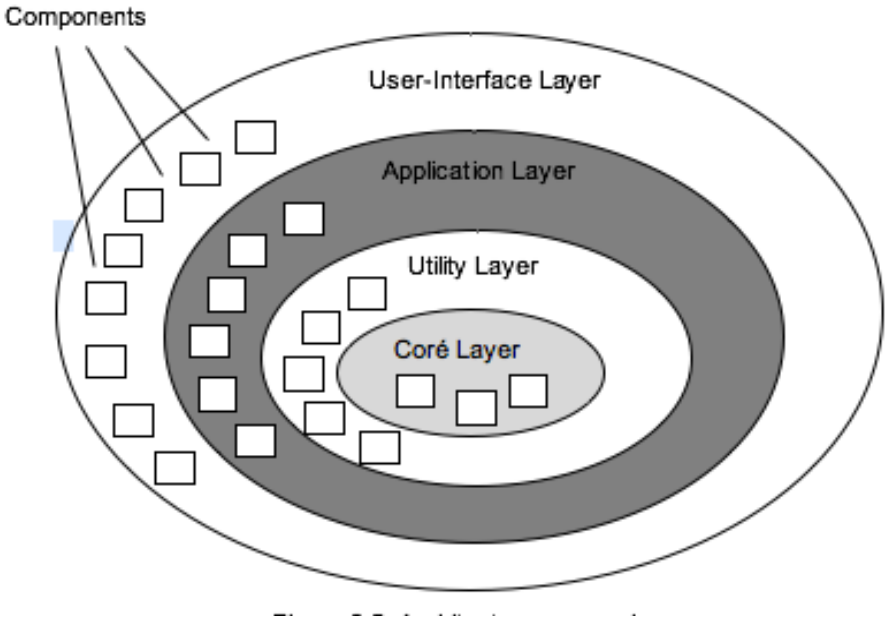


Figure 3.5: Architecture en couches

Objectifs de la conception architecturale

Pour développer un modèle d'architecture logicielle, ce qui donne une organisation globale du module de programme dans le produit logiciel. L'architecture logicielle comprend deux aspects de la structure des données et des structures hiérarchiques des composants logiciels. La conception architecturale définit l'organisation des composantes du programme. Il ne fournit pas les détails de chaque composant et sa mise en œuvre.

Pour contrôler la relation entre les modules. Un module peut commander un autre module ou peut être commandé par un autre module. L'organisation d'un module peut être représentée par une structure en forme d'arbre.

Pourquoi l'architecture logicielle est importante ?

L'architecture logicielle est importante parce que :

1. Représentations de l'architecture logicielle sont un catalyseur pour la communication entre toutes les parties (parties prenantes) intéressés par le développement d'un système informatique.
2. L'architecture met en lumière les décisions précoces de conception qui auront un impact profond sur tous les logiciels travaux d'ingénierie qui suit et, tout aussi important, sur le succès ultime du système comme une entité opérationnelle.
3. Architecture "constitue un relativement petit modèle, intellectuellement saisissable de la façon dont le système est structuré et comment ses composantes travaillent ensemble".

Spécification Résumé

Pour chaque sous-système, une spécification abstraite des services qu'elle fournit et les contraintes dans lesquelles il doit fonctionner est produite. L'objectif de base de la conception du système est de spécifier les modules dans un système et leurs abstractions. Une fois les différents modules sont spécifiés, au cours de la conception détaillée du concepteur peut se concentrer sur un seul module à la fois. La tâche dans la conception et la mise en œuvre détaillée consiste essentiellement à mettre en œuvre les modules de sorte que les caractéristiques abstraites de chaque module sont réunies.

Il existe deux mécanismes d'abstraction commune pour les systèmes de logiciels :

- Abstraction fonctionnelle et
- Abstraction de données.

Dans l'abstraction fonctionnelle, un module est spécifié par la fonction qu'il remplit. Par exemple, un module pour trier un tableau d'entrée peut être représenté par la spécification de tri. L'abstraction fonctionnelle est la base de partitionnement dans les approches axées sur la fonction. Autrement dit, lorsque le problème est partagé, la fonction de transformation globale du système est divisée en petites fonctions qui composent la fonction système.

La deuxième unité d'abstraction est l'abstraction de données. Il y a certaines opérations nécessaires à partir d'un objet de données, en fonction de l'objet et l'environnement dans lequel il est utilisé. L'abstraction des données soutient ce point de vue. Les données ne sont pas traités comme de simples objets, mais est traité comme des objets avec certaines opérations prédéfinies sur eux. Les opérations définies sur un objet de données sont les seules opérations qui peuvent être effectuées sur ces objets. De l'extérieur d'un objet, le fonctionnement interne de l'objet est caché; seules les opérations sur l'objet sont visibles.

Conception Interface Utilisateu

La conception de l'interface se concentre sur trois domaines de préoccupation:

1. La conception des interfaces entre les composants logiciels
2. La conception des interfaces entre les logiciels et les autres producteurs et consommateurs d'informations (à savoir d'autres entités externes) non humains, et
3. La conception de l'interface entre un être humain (à savoir, l'utilisateur) et l'ordinateur.

Ce processus de conception de logiciels est concerné par la troisième conception de l'interface de conception d'interface de catégorie d'utilisateur.

La conception de l'interface utilisateur crée un moyen de communication efficace entre un humain et un ordinateur. Après une série de principes de conception d'interface, la conception identifie les objets et les actions d'interface et crée ensuite une mise en page d'écran qui constitue la base d'un prototype d'interface utilisateur.

La conception de l'interface utilisateur commence par l'identification des exigences environnementales utilisateur, la tâche, et. Une fois les tâches utilisateur ont été identifiés, les scénarios d'utilisateurs sont créés et analysés pour définir un ensemble d'objets et d'actions interface. Ceux-ci forment la base pour la création de la mise en page de l'écran qui représente la conception graphique et le placement des icônes, la définition du texte descriptif de l'écran, la spécification et le titrage pour les fenêtres, et la spécification des grands et petits éléments de menu. Les outils sont utilisés pour le prototype et, finalement, mettre en œuvre le modèle de conception, et le résultat est évalué pour la qualité.

La conception de l'interface utilisateur a autant à voir avec l'étude des personnes. Quelques questions qui doivent être posées et des réponses dans le cadre de la conception de l'interface utilisateur comprennent :

1.Qui est l'utilisateur?

2.Comment l'utilisateur va apprendre à interagir avec un nouveau système informatique?

3.Comment l'utilisateur interpréter l'information produite par le système?

4.Qu'est-ce que l'utilisateur attend du système?

Règles d'or dans la conception de l'interface utilisateur

Il y a trois règles d'or qui forment la base d'un ensemble de principes de conception de l'interface utilisateur

- Placer l'utilisateur dans le contrôle.
- Réduire la charge de la mémoire de l'utilisateur.
- Faire l'interface cohérente.

a) Placer l'utilisateur dans le contrôle

Les suivants sont les principes de conception qui permettent à l'utilisateur de garder le contrôle:

Définir les modes d'interaction d'une manière qui ne force pas un utilisateur en actions inutiles ou indésirables. Un mode d'interaction de l'état actuel de l'interface.

Fournir une interaction flexible. Parce que les différents utilisateurs ont des préférences différentes d'interaction, les choix devraient être fournis.

Permettre l'interaction utilisateur soit interruptible et infaisable. Même lorsqu'ils sont impliqués dans une séquence d'actions, l'utilisateur doit être en mesure d'interrompre la séquence à faire autre chose (sans perdre le travail qui avait été fait). L'utilisateur doit également être en mesure de "annuler" toute action.

Simplifier l'interaction que les niveaux de compétence avance et permettre l'interaction à personnaliser. Les utilisateurs trouvent souvent qu'ils effectuent la même séquence d'interactions à plusieurs reprises. Il est intéressant de concevoir un mécanisme de «macro» qui permet à un utilisateur avancé pour personnaliser l'interface pour faciliter l'interaction.

Cacher internes techniques de l'utilisateur occasionnel. L'interface utilisateur doit déplacer l'utilisateur dans le monde virtuel de l'application. L'utilisateur ne doit pas être au courant du système d'exploitation, déposer les fonctions de gestion, ou d'autres technologies de calcul des arcanes.

Conception pour une interaction directe avec les objets qui apparaissent sur l'écran. L'utilisateur se sent un sentiment de contrôle quand capable de manipuler les objets qui sont nécessaires pour effectuer une tâche d'une manière similaire à ce qui se produirait si l'objet était une chose physique.

b) Réduire la charge de la mémoire de l'utilisateur

Plus un utilisateur doit se rappeler, plus d'erreurs sera l'interaction avec le système. Il est pour cette raison que l'interface utilisateur bien conçue ne taxe la mémoire de l'utilisateur. Chaque fois que possible, le système doit «se souvenir» des informations pertinentes et d'aider l'utilisateur à un scénario d'interaction qui aide le rappel.

c) Faire l'interface cohérente.

L'interface doit présenter et obtenir de l'information d'une manière cohérente. Cela implique que :

Toutes les informations visuelles soient organisées selon une norme de conception qui est maintenue dans tous les écrans,

Mécanismes d'entrée sont limitées à un ensemble limité qui sont utilisés régulièrement dans toute l'application,

Les mécanismes de navigation d'une tâche sont bien définies et mises en œuvre.

Processus de conception d'interface utilisateur

L'ensemble du processus de conception d'une interface utilisateur :

- Commence par la création de différents modèles de fonctionnement du système (tel que perçu de l'extérieur).
- Les tâches orientées humains-ordinateurs qui sont nécessaires pour atteindre la fonction du système sont ensuite décrites;
- Les problèmes de conception applicables à tous les modèles d'interface sont considérés; outils sont utilisés pour le prototypage
- Mettre en œuvre définitive, le modèle de conception; et le résultat est évalué pour la qualité.

a) Interface de conception Modèles

Un modèle de conception de l'ensemble du système intègre les données, l'architecture, l'interface, et les représentations de procédure du logiciel. Le cahier des charges peut établir certaines contraintes qui aident à définir l'utilisateur du système, mais la conception de l'interface est souvent accessoire au modèle de conception.

Le modèle de l'utilisateur établit le profil des utilisateurs finaux du système. Pour construire une interface utilisateur efficace, "toute la conception doit commencer par une compréhension des utilisateurs visés, y compris les profils de leur âge, le sexe, les capacités physiques, l'éducation, culturelle ou ethnique, la motivation, les objectifs et la personnalité"

b) Le processus de conception d'interface utilisateur

Le processus de conception pour les interfaces utilisateur est itératif et peut être représenté en utilisant un modèle en spirale comme représenté à la figure 3.6. La spirale implique que chacune de ces tâches vont apparaître plus d'une fois, à chaque passage autour de la spirale représentant élaboration supplémentaire des besoins et la conception résultante. Dans la plupart des cas, l'activité de mise en œuvre implique le prototypage, le seul moyen pratique de valider ce qui a été conçu. Le processus de conception d'interface utilisateur comprend quatre activités-cadres distincts :

I.Utilisateur, la tâche, et l'analyse de l'environnement et de la modélisation

L'activité initiale d'analyse se concentre sur le profil des utilisateurs interagissent avec le système. Niveau de compétence, la compréhension des affaires, et la réceptivité générale du nouveau système sont enregistrées; et les différentes catégories d'utilisateurs sont définies. Pour chaque catégorie d'utilisateur, les exigences sont provoquées.

Une fois que les exigences générales ont été définies, une analyse des tâches plus détaillée est menée. Ces tâches que l'utilisateur effectue pour atteindre les objectifs du système sont identifiées, décrites et élaborées (sur un certain nombre de passes itératives à travers la spirale).

II.Design d'interface

Les informations recueillies dans le cadre de l'activité d'analyse est utilisée pour créer un modèle d'analyse de l'interface. En utilisant ce modèle de base, l'activité de conception commence. Le but de la conception de l'interface est de définir un ensemble d'objets

d'interface et des actions (et leurs représentations à l'écran) qui permettent à un utilisateur d'effectuer toutes les tâches définies d'une manière qui répond à tous les objectifs de la facilité d'utilisation défini pour le système.

III.Construction d'interface

L'activité de mise en œuvre commence normalement par la création d'un prototype qui permet de scénarios d'utilisation à évaluer. Comme le processus itératif de conception continue, une trousse d'outils de l'interface utilisateur peut être utilisée pour compléter la construction de l'interface.

IV.La validation de l'interface se concentrate sur:

- La capacité de l'interface pour mettre en œuvre toutes les tâches de l'utilisateur correctement, pour accueillir toutes les variations de la tâche, et de parvenir à toutes les exigences générales de l'utilisateur;
- La mesure dans laquelle l'interface est facile à utiliser et facile à apprendre; et
- L'acceptation par les utilisateurs de l'interface comme un outil utile dans leur travail.

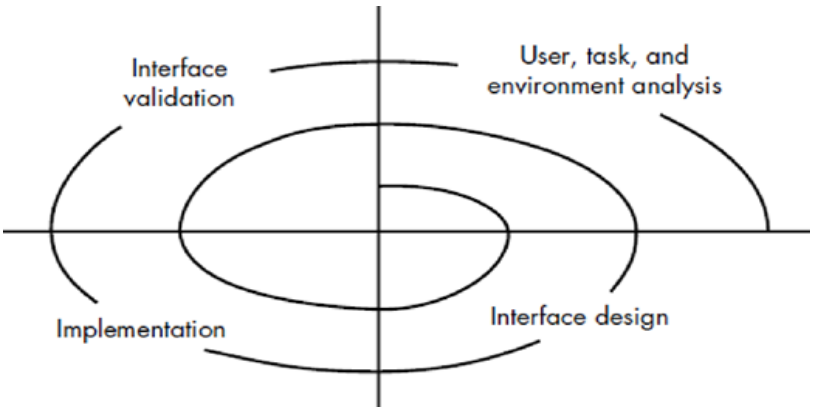


Figure 3.6: Le processus de conception de l'interface utilisateur (Source: Pressman, 2001)

Conception de composants

La conception au niveau du composant, appelé aussi la conception de la procédure, se produit après que les données, l'architecture, et des conceptions d'interface ont été établis. Voilà des données, l'architecture et la conception de l'interface doivent être traduits dans le logiciel opérationnel. Pour ce faire, la conception doit être représentée à un niveau d'abstraction qui est proche de code. La conception au niveau des composants établit le détail algorithmique nécessaire pour manipuler des structures de données, la communication de l'effet entre les composants logiciels via leurs interfaces, et mettre en œuvre les algorithmes de traitement attribués à chaque composant.

La conception au niveau des composants est représentée en utilisant un langage de programmation. Essentiellement, le programme est créé en utilisant le modèle de conception comme un guide. Une approche alternative est de représenter la conception procédurale en utilisant certains intermédiaires (par exemple, graphique, tabulaire, ou texte) représentation qui peut être traduit facilement dans le code source.

Programmation structurée

Pressman 2001 indiquent que dans les années 1960 un ensemble de constructions logiques contraintes a été proposé à partir de laquelle tout programme pourrait être formé. Les constructions ont souligné "l'entretien de domaine fonctionnel." Autrement dit, chaque construction avait une structure logique prévisible, a été entré dans la partie supérieure et est sorti au fond, ce qui permet au lecteur de suivre le flux de procédure plus facilement. Les constructions sont séquence, l'état, et la répétition.

- Séquence met en œuvre les étapes de traitement qui sont essentiels dans la description d'un algorithme.
- État offre la possibilité pour le traitement choisi sur la base de quelque événement logique et
- La répétition permet d'itérer.

Ces trois constructions sont fondamentales à une programmation technique de conception structurée importante au niveau des composants.

Low Level-Design

Modularisation

Un système est considéré comme modulaire si elle est constituée de composants discrets de sorte que chaque composant peut être mis en œuvre séparément, et un changement d'un composant a un impact minimal sur les autres composants.

Les systèmes modulaires intègrent des collections d'abstractions dans lequel chaque abstraction fonctionnelle, chaque abstraction de données, et chaque abstraction de contrôle gère un aspect local du problème étant résolu.

Système modulaire se compose de bien définies, des unités gérables avec des interfaces bien définies entre les unités. Les propriétés souhaitables d'un système modulaire comprennent :

- Chaque fonction dans chaque abstraction a un seul but bien défini.
- Chaque fonction manipule pas plus d'une importante structure de données.
- Fonctions de partager des données globales sélectivement. Il est facile d'identifier toutes les routines qui partagent une structure principale de données.
- Les fonctions qui manipulent des instances de types de données abstraites sont encapsulées avec la structure de données manipulé.

Modularité améliore la clarté de la conception, qui à son tour facilite la mise en œuvre, le débogage, les tests, la documentation, et la maintenance du logiciel.

Structure Graphique

Le tableau de la structure est l'une des méthodes les plus couramment utilisés pour la conception du système. Les diagrammes de structure sont utilisés lors de la conception architecturale pour documenter les structures hiérarchiques, des paramètres et des interconnexions dans un système.

Il partitionne un système dans des boîtes noires. Une boîte noire signifie que la fonctionnalité est connue à l'utilisateur sans la connaissance de la conception interne. Les entrées sont donnés à une boîte noire et les sorties appropriées sont générés par la boîte noire. Ce concept permet de réduire la complexité parce que les détails sont cachés de ceux qui n'ont pas le besoin ou le désir de savoir. Ainsi, les systèmes sont faciles à construire et facile à entretenir.

Pseudo-Code

"Pseudo" signifie imitation ou faux et «code» se réfère aux instructions écrites dans un langage de programmation. La notation pseudo-code peut être utilisée dans les deux phases de conception préliminaire et détaillée. Utilisation de pseudo-code, le concepteur décrit les caractéristiques du système à l'aide de courtes, concises phrases de langue anglaise qui sont structurés par des mots-clés, tels que If-Then-Else, While-Do. Mots-clés et indentation décrivent le flux de contrôle, tandis que les expressions anglaises décrivent les actions de traitement. Le Pseudo-code est également connu comme langue de conception de programme.

Flowcharts

Un organigramme est une technique commode de représenter le flux de contrôle dans un programme. Un organigramme est une représentation graphique d'un algorithme qui utilise des symboles pour montrer les opérations et les décisions à suivre par un ordinateur pour résoudre un problème. Les instructions réelles sont écrites dans les symboles / boîtes en utilisant des déclarations claires. Ces boîtes sont reliées par des lignes pleines ayant des flèches pour indiquer l'écoulement de l'opération dans une séquence.

Organigrammes sont le plan à suivre lorsque le programme est écrit. Les programmeurs experts peuvent écrire des programmes sans tirer les organigrammes. Mais pour un débutant, il est recommandé qu'un organigramme devrait être établi avant d'écrire un programme, qui à son tour permettra de réduire le nombre d'erreurs et d'omissions dans le programme. Les Flowcharts aident également au cours des essais et des modifications dans les programmes.

Différence entre Flowcharts et structure Charts

Un diagramme de la structure diffère d'un organigramme de la manière suivante :

- Il est généralement difficile d'identifier les différents modules du logiciel à partir de sa représentation d'organigramme.
- L'échange de données entre les différents modules n'est pas représentée dans un organigramme.
- La commande séquentielle des tâches inhérentes à un organigramme est supprimée dans un diagramme de structure.
- Un organigramme n'a pas de boîtes de décision.

Contrairement à des organigrammes, diagrammes de structure montrent comment les différents modules au sein d'une interaction de programme et les données qui est passé entre eux.

Évaluation

1. Définir la conception architecturale.
2. Quels sont les objectifs de la conception architecturale?
3. Expliquer les différentes techniques de conception qui relèvent de la catégorie de la conception de bas niveau.
4. Définir:
   a) modularisation
   b) les cartes de structure
   c) Pseudo-Code
   d) Flowcharts
5. Distinguer entre «Structure Charts » et "Flowcharts" en donnant des exemples
6. Qu'est-ce qu'un organigramme? Expliquer certains de ses symboles. Aussi donner un exemple approprié.
7. Donnez le format hiérarchique d'un diagramme de structure. En outre, donner les éléments de base d'un diagramme de structure.
8. Élaborer deux principes de conception supplémentaires qui "placent l'utilisateur dans le contrôle."
9. Élaborer deux principes de conception supplémentaires qui "réduisent la charge de la mémoire de l'utilisateur."

10. Élaborer deux principes de conception supplémentaires qui «font l'interface cohérente."

11. Tous les langages de programmation modernes mettent en œuvre les constructions de programmation structurés. Fournir des exemples de trois langages de programmation.

Activité 3: Autres aspects de la conception de logiciels

Les approches orientées fonctionnelles Versus les approches orientées objet

Le tableau 3.1 présente la différence entre l'approche orientée fonctionnelle Versus l'approche orientée objet dans la conception de logiciels

Table 3.1: Functional-Oriented Versus Object-Oriented Approaches

S/N	Functional-oriented Approach	Object-oriented Approach
1.	The basic abstractions, which are given to the user, are real-world functions, such as sort, merge, track, display, etc.	The basic abstractions are not the real-world functions, but are the data abstraction where the real-world entities are represented, such as picture, machine, radar system, customer, student, employee, etc.
2.	Functions are grouped together by which a higher-level function is obtained. An example of this technique is software analysis/ Software Design (SA/SD).	The functions are grouped together on the basis of the data they operate on, such as in class person, function displays are made member functions to operate on its data members such as the person name, age, etc.
3.	The state information is often represented in a centralized shared memory.	The state information is not represented in a centralized shared memory but is implemented/distributed among the objects of the system.

Les spécifications de conception

Les spécifications de conception abordent différents aspects du modèle de conception et sont complétés en tant que concepteur affine sa représentation du logiciel.

- Premièrement, la portée globale de l'effort de conception est décrite, qui est dérivé de la spécification du système et le modèle d'analyse (spécification des exigences logicielles).
- Ensuite, la conception de données est spécifiée, qui comprend des structures de données, des structures de fichiers externes, des structures de données internes, et une référence croisée qui relie des objets de données à des fichiers spécifiques.
- Ensuite, la conception architecturale indique comment l'architecture du programme a été dérivée du modèle d'analyse. diagrammes de structure sont utilisés pour représenter la hiérarchie du module.
- La conception de l'interface indique la conception d'interfaces de programmes externes et internes avec une conception détaillée de l'interface homme / machine. Un prototype détaillé d'une interface graphique peut également être représenté.
- La conception de procédure spécifie les composants séparément-éléments adressables de logiciels tels que des sous-programmes, des fonctions ou des procédures sous la forme de récits de traitement de la langue anglaise. Ce récit explique la fonction procédurale d'un composant (module).

La spécification de conception contient une référence croisée exigences. Le but de cette référence croisée est :

- Pour établir que toutes les conditions sont remplies par la conception du logiciel.
- Pour indiquer quels composants sont essentiels à la mise en œuvre des exigences spécifiques.

La dernière partie de la spécification de conception contient des données supplémentaires, comme la description de l'algorithme, les procédures de remplacement et des données tabulaires.

Vérification de la conception

Comme dans les autres phases du processus de développement, la sortie de la phase de conception du système doit être vérifiée avant de procéder avec les activités de la phase suivante.

Si la conception est exprimée dans une notation formelle pour laquelle les outils d'analyse sont disponibles, grâce à des outils, il est possible de vérifier la cohérence interne (par exemple, les modules utilisés par un autre sont définis, l'interface d'un module est compatible avec la façon dont les autres utilisent elle, l'utilisation des données est conforme à la déclaration, etc.).

Si la conception n'est pas spécifiée dans un langage exécutable formel, il ne peut pas être traitée au moyen d'outils, et d'autres moyens de vérification doivent être utilisés.

Il existe deux approches fondamentales de vérification :

- La première consiste à expérimenter avec le comportement d'un produit pour voir si le produit fonctionne comme prévu (par exemple, tester le produit).
- L'autre consiste à analyser le produit ou toute documentation de conception liée à elle-déduire son bon fonctionnement comme une conséquence logique des décisions de conception. Les deux catégories de vérification

Évaluation

1. Donner des deux différences importantes entre les approches axées sur la fonction et la conception orientée objet.
2. Discuter des principaux avantages de l'approche orientée objet de conception sur l'approche de conception orientée fonction.
3. Expliquer la spécification de conception à long terme.
4. Discuter de la vérification à long terme en référence à la conception du système.
5. Quelle est l'abstraction? Quelles sont les mesures de vérification pour la conception du système?
6. Avez-vous concevoir un logiciel lorsque vous "écrivez" un programme? Ce qui rend la conception de logiciels différents de codage?

Résumé de l'unité

Le design est la partie la plus importante de l'ingénierie logicielle. Lors de la conception, des améliorations progressives de structure de données, l'architecture, les interfaces et les détails de la procédure de composants logiciels sont élaborés, revus et documentés. Résultats de conception dans les représentations de logiciels qui peuvent être évalués pour la qualité.

Un certain nombre de principes et de concepts de conception de logiciels de base ont été présentés. Les principes de conception guident l'ingénieur logiciel que les processus de conception produit. Les concepts de conception fournissent des critères de base pour la qualité du design. Modularité (à la fois le programme et les données) et le concept d'abstraction permet au concepteur de simplifier et de réutiliser des composants logiciels.

Raffinement fournit un mécanisme pour représenter des couches successives de détail fonctionnel. Programme et structure de données contribuent à une vue d'ensemble de l'architecture logicielle, tandis que la procédure fournit les détails nécessaires à la mise en œuvre de l'algorithme. Le masquage d'information et l'indépendance fonctionnelle fournissent des heuristiques pour atteindre une modularité efficace.

Évaluation de l'unité

Vérifiez votre compréhension!

1. Qu'entendez-vous par le logiciel
2. Qu'est-ce et des logiciels d'ingénierie?
3. Quels sont les différents mythes et la réalité sur le logiciel?
4. D'un détail expliquer le processus de génie logiciel

Lectures et autres ressources

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783
- Zhiming Liu, (2001), "Object-Otiented Software Development Using UML", The United Nations University, UNU-IIST International Institute for Software Technology, Tech Report 229.

Unité 4. Implémentation et tests

Introduction à l'unité

Pendant la phase de mise en œuvre, chacun des composants de la conception est réalisé en tant qu'unité de programme. Chaque unité doit ensuite être soit vérifiée ou testée par rapport à sa spécification obtenue grâce à l'étape de conception. Le Test du logiciel est de savoir comment s'assurer qu'il répond aux spécifications de la phase de conception après sa mise en œuvre. Cette unité traitera de manière générale tous les aspects liés à la mise en œuvre et les tests de logiciels. L'unité décrira les techniques utilisées pour la mise en œuvre et les mécanismes utilisés pour effectuer des tests de logiciels.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de :

- Définir ce qu'est la mise en œuvre et les concepts de test
- Décrire les principes de base des objectifs de test et d'essai
- Déterminer si le comportement observé est conforme au comportement attendu
- Elaborer différents niveaux de tests
- Distinguer et effectuer des tests boîte blanche et boîte noire
- Définir les différentes techniques de test et de distinguer leurs différences
- Effectuer les plans de test

Termes clés

Mise en œuvre logicielle: la mise en œuvre de logiciels est un processus de réalisation du cahier des charges de conception comme une unité de programme

Test: Le test est un ensemble d'activités utilisé pour tester le code source afin de découvrir (et corriger) les erreurs avant la livraison du logiciel client.

Cas de test: Un cas de test est un ensemble d'instructions destiné à découvrir un type particulier d'erreur ou d'un défaut dans le système de logiciel en induisant un échec

Essais structuraux: Les essais structuraux est une approche de test où les tests sont tirés de la connaissance de la structure et la mise en œuvre du logiciel.

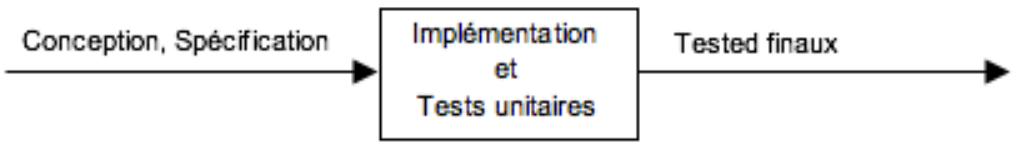
Tests fonctionnels : Les tests fonctionnels se réfère à des tests qui implique que l'observation de la sortie pour certaines valeurs d'entrée, et il n'y a aucune tentative d'analyser le code, qui produit la sortie

Activités d'apprentissage

Activité 1: Implémentation ou Codage du logiciel

Introduction

Lors de l'étape de mise en œuvre, chacun des éléments de la conception est réalisée comme une unité de programme. Chaque unité doit ensuite être soit vérifiée ou testée par rapport à sa spécification obtenue à l'étape de conception. Ce processus est tel que représenté sur la figure 4.1. Ensuite, les unités de programmes individuels représentant les composants du système sont réunies et testées dans son ensemble pour garantir que les exigences de logiciel ont été satisfaites. Lorsque les développeurs sont satisfaits du produit, il est ensuite testé par le (les tests d'acceptation) client. Cette phase se termine lorsque le produit est accepté par le client.



Codage

Au cours de codage l'accent est mis sur l'élaboration de programmes qui sont faciles à lire et à comprendre et non pas simplement sur le développement de programmes qui sont simples à écrire. Le codage peut être soumis à des normes à l'échelle de l'entreprise qui peuvent définir l'ensemble des pages de programmes, tels que les en-têtes pour des commentaires dans toutes les unités, les conventions de nommage pour les variables, les classes et fonctions, le nombre maximum de lignes dans chaque composant, et d'autres aspects de la normalisation.

La programmation structurée permet à l'intelligibilité des programmes. L'objectif de la programmation structurée est de linéariser le flux de contrôle du programme. Une simple sortie et des constructions d'entrée unique doivent être utilisées. Les constructions comprennent la sélection (si-alors-sinon) et les itérations.

Programmation structurée

La programmation structurée fait référence à une méthodologie générale de la rédaction de bons programmes. Un bon programme est celui qui a les propriétés suivantes :

a) Il faut effectuer toutes les actions souhaitées.

b) Il doit être fiable, à savoir, effectuer les actions requises dans les marges d'erreur acceptables.

c) Il doit être clair, à savoir, facile à lire et à comprendre.

d) Il devrait être facile à modifier.

e) Il devrait être mis en œuvre dans le calendrier et le budget spécifiés.

Les programmes structurés ont comme propriété une entrée unique, et une unique sortie. Cette fonctionnalité aide à réduire le nombre de trajets pour l'écoulement de contrôle. S'il y a des chemins arbitraires pour le flux de contrôle, le programme sera difficile à lire, comprendre, déboguer et maintenir.

Un programme est l'un des deux types : structure statique ou structure dynamique.

- La structure statique est la structure du texte du programme, qui est habituellement juste une organisation linéaire des états du programme.
- La structure dynamique du programme est la séquence d'instructions exécutées au cours de l'exécution du programme.

Les deux structures statiques et dynamiques sont la séquence d'instructions. La seule différence est que la séquence d'instructions dans une structure statique est fixe, tandis que, dans une structure dynamique, il n'a pas été fixé. Cela signifie que la séquence dynamique des états peut changer d'exécution en exécution. La structure statique d'un programme peut être facilement comprise. La structure dynamique d'un programme peut être facilement vue au moment de l'exécution.

Objectifs de programmation structurée

L'objectif de la programmation structurée est d'écrire des programmes de sorte que la séquence d'instructions exécutées au cours de l'exécution d'un programme est la même que la séquence d'instructions dans le texte de ce programme.

Comme les déclarations dans un texte de programme sont linéairement organisées, l'objectif de la programmation structurée est de développer des programmes dont le débit commandé pendant l'exécution est linéarisé et suit l'organisation linéaire du texte du programme.

Depuis qu'un programme ne peut pas être écrit comme une séquence d'instructions simples, sans aucune ramification ou de répétition, des constructions structurées sont utilisées. Dans la programmation structurée, une déclaration n'est pas une déclaration d'affectation simple, elle est une déclaration structurée.

Principes de programmation structurée

Toutes les méthodes de conception de programmes structurés reposent sur les deux principes fondamentaux affinement progressif et trois structures de contrôle structurés. L'objectif de la conception du programme est de transformer la fonction souhaitée du programme, comme indiqué dans le cahier des charges du programme, dans un ensemble d'instructions, qui peut facilement être traduit en un langage de programmation choisi. Le processus d'affinement progressif est une approche que la fonction du programme indiqué est décomposé en fonctions subsidiaires en augmentant progressivement le niveau de détail jusqu'à ce que les fonctions de niveau les plus bas sont réalisables dans le langage de programmation.

Le deuxième principe de la conception d'un programme structuré est que tout programme peut être construit en utilisant seulement trois structures de contrôle structurées. La sélection des constructions, des itérations, et la séquence sont indiquées sur la figure 4.2 (a, b, c, d). Tout programme indépendant de la plate-forme technologique peut être écrit en utilisant ces produits de construction, à savoir, la sélection, la répétition, l'ordre. Ces structures sont à la base de la programmation structurée.

Avantages de la programmation structurée

Les avantages de la programmation structurée sont :

- Est-ce que il est très commode de mettre la logique systématiquement dans le programme. En raison de la facilité de manipulation, l'utilisateur, le lecteur, et le programmeur doivent pouvoir comprendre facilement le programme, alors que la logique pourrait être complexe et difficilement compréhensible.
- Il est facile de vérifier, de procéder à des examens, et de tester les programmes structurés de manière ordonnée. Si des erreurs sont détectées, elles sont faciles à repérer et à corriger.

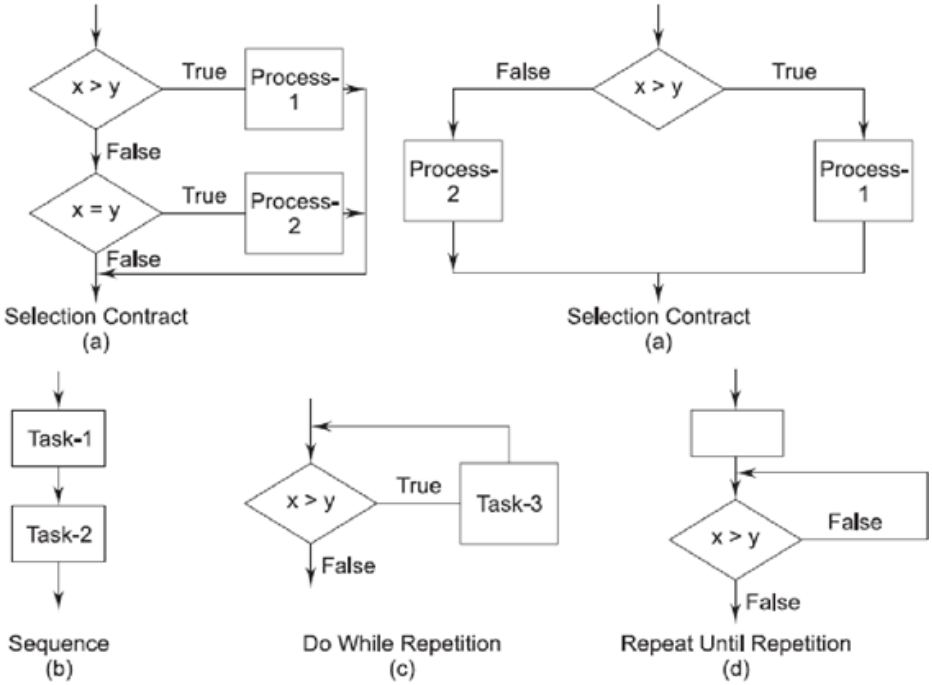


Figure 4.2: Bases de la Programmation Structurée : Sélection, Itérations, et Séquence

Évaluation

Vérifiez votre compréhension!

1. Qu'entendez-vous par le logiciel
2. Qu'est-ce et des logiciels d'ingénierie?
3. Quels sont les différents mythes et la réalité sur le logiciel?
4. D'un détail expliquer le processus de génie logiciel

Activité 2: Fondamentaux du Test de Logiciel

Principes d'essai

Le test est un ensemble d'activités qui peuvent être planifiées à l'avance et menées systématiquement. Un certain nombre de stratégies logicielles tests ont été proposés dans la littérature. Tous fournissent au développeur des logiciels accompagnés de modèle pour les tests. Avant de faire usage de techniques de test, l'ingénieur logiciel doit connaître les principes de base des processus de test. Un modèle de test a le principe de base suivant (Agarwal et al., 2010) :

a) Tous les tests doivent être déterminés en fonction des besoins des clients

Le but est de découvrir d'éventuels défauts ou des défauts qui causent que le système ne fonctionne pas selon les exigences du client.

b) Les tests doivent être planifiées avant même de commencer

Après avoir terminé le processus d'analyse des exigences, la planification de test peut commencer. Les cas de tests détaillés peuvent commencer dès que le modèle de conception se termine.

c) Le principal Pareto est appliqué aux tests de logiciels

Simple test en utilisant le principe de Pareto affirme que 80 pour cent de tous les défauts découverts lors de la phase de test peut affecter 20 pour cent de toutes les composantes du programme. Le problème à ce stade est d'isoler ces composants suspects en les testant.

d) Les essais devraient commencer "dans le petit composant" et aller progressivement vers les "grands."

Les premiers tests planifiés et exécutés généralement se concentrent sur les composants individuels. Comme le test progresse, le focus passe dans une tentative de trouver des erreurs dans les grappes intégrées de composants et, finalement, dans l'ensemble du système.

e) Le Test Exhaustif n'est pas possible

Le nombre de permutations de chemin, même pour un programme de taille modérée est exceptionnellement élevé. Pour cette raison, il est impossible d'exécuter toutes les combinaisons de chemins au cours des essais. Il est possible, cependant, pour couvrir de manière adéquate la logique du programme, de veiller à ce que toutes les conditions de la conception au niveau des composants soient exercées.

f) Pour être plus efficace un essai devrait être effectué par un tiers indépendant

L'ingénieur logiciel qui crée un système n'est pas la meilleure personne pour effectuer tous les tests du programme. Surtout pour les grands projets un groupe de test indépendant est nécessaire.

Une stratégie pour les tests de logiciels doit tenir compte des tests de bas niveau qui sont nécessaires pour vérifier qu'un petit segment de code source a été correctement mis en œuvre ainsi que des tests de haut niveau qui valident les principales fonctions du système par rapport aux exigences des clients. Une stratégie doit fournir des directives pour le praticien et un ensemble de jalons pour le gestionnaire.

Test Oracle

Pour tester un programme, nous avons besoin d'avoir une description de son comportement attendu et un procédé pour déterminer si le comportement observé est conforme au comportement attendu. Pour cela nous avons besoin d'un Test Oracle.

Un Test Oracle est un mécanisme, différent du programme lui-même, qui peut être utilisé pour vérifier l'exactitude de la sortie du programme pour les cas de test.

Conceptuellement, nous pouvons envisager de tester un processus dans lequel les cas de test sont donnés à Test Oracle et le programme en cours de test. Les deux sorties produites sont ensuite comparées afin de déterminer si le programme se comportait bien dans les cas d'essai, comme le montre la figure 4.3.

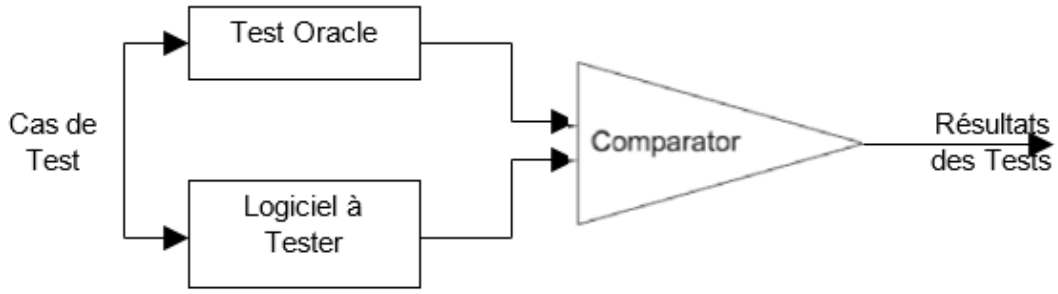


Figure 4.3: Test Oracle (Source: Agarwal et al., 2010)

Les « Test Oracle » sont considérés comme des humains, ils peuvent effectuer des tests quand il y a un écart entre les oracles et les résultats du programme. Vous devez d'abord vérifier les résultats produits par les oracles avant de déclarer qu'il y a une faille dans le programme. Dans le tableau illustré ci-dessus, les données de cas de test sont conformes aussi dans le « Test Oracle » qu'au niveau du programme a testé. Le résultat de chacun des éléments est comparé pour déterminer si le programme se comportait bien en conformité avec le scénario de test.

Les oracles humains utilisent généralement les spécifications du programme afin de décider ce que le comportement «correct» du programme ne devrait pas être. Pour aider l'oracle à déterminer le comportement correct, il est important que le comportement du système soit spécifié sans ambiguïté et que la spécification elle-même soit exempte d'erreurs.

Évaluation

Vérifiez votre compréhension!

1. Que sont les tests ? Expliquer les différents types de tests effectués au cours du développement de logiciels.
2. Définir les différents principes de tests.
3. Quels sont les oracles de test?

Activité 3: Niveaux de Test

Il y a trois niveaux de test :

- Les tests unitaires
- Test d'intégration
- Test du système

Tests Unitaires

Dans la phase de tests unitaires, de composants individuels sont testés pour garantir leur bon fonctionnement. Sur la plus petite unité de conception de logiciels, chaque composant est testé de manière indépendante, sans les autres composants du système. Il y a quelques raisons pour effectuer des essais de conduite au lieu de tester l'ensemble du produit :

- La taille d'un simple module est assez petite pour détecter assez facilement une erreur.
- Le module est assez petit pour pouvoir tenter de le tester de manière exhaustive
- En itérant les tests unitaires, de multiples erreurs dans de très différentes parties du logiciel sont éliminées.

La figure ci-dessous illustre le processus de réalisation des tests unitaires.

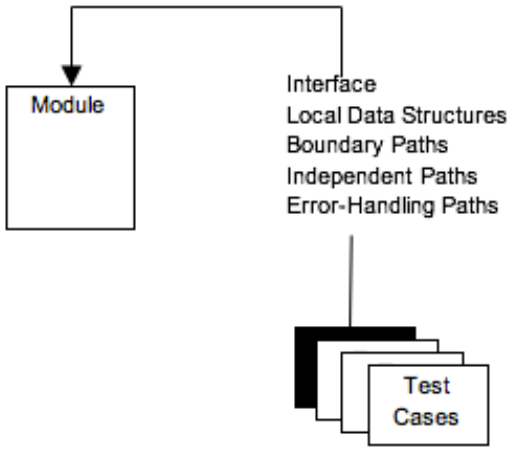


Figure 4.4: Unit Tests (Source: Agarwal et al., 2010)

Dans ce cas, l'interface de module est testée afin de faire en sorte que les flux d'informations circulent correctement à l'intérieur et à l'extérieur de l'unité du programme qui est en cours de test. La structure locale de données est examinée pour s'assurer que les données stockées temporairement conservent leur intégrité pendant toutes les étapes de l'algorithme d'exécution. Tous les chemins indépendants à travers le contrôle de la structure sont exercés pour assurer que toutes les déclarations qui ont été faites dans le module ont exécutées au moins une fois. Et enfin, toutes les erreurs de manipulation des chemins sont testées. Selon Agarwal et al, 2010 il y a quelques erreurs courantes dans le calcul :

- Mélange de codes d'opération
- Initialisation incorrecte
- Priorité incorrecte pour les opérations arithmétiques
- Degré de précision non définie
- la représentation incorrecte de la représentation des expressions

Les cas de test dans les tests unitaires doivent permettre de découvrir des erreurs comme :

- La comparaison de différents types de données
- Les opérateurs logiques erronés
- Comparaison incorrecte de variables
- Terminaison de boucle incorrecte
- Défaut de quitter lorsque l'itération divergente est rencontrée
- Variables de boucle incorrectement modifiées

Test d'intégration

Un autre niveau de test est le test d'intégration. Le Test d'intégration est une technique systématique pour la construction programmes structurés permettant de découvrir les erreurs liées à l'interfaçage. Dans un tel test, les modules unitaires testés sont combinés en sous-systèmes, qui sont ensuite testés. Le but est de voir si les modules peuvent être intégrés correctement.

Voici les différentes approches utilisées pour effectuer des tests d'intégration :

Approche incrémentale

L'approche incrémentale signifie d'abord associer uniquement deux composants ensemble et les tester. Retirez les erreurs si elles sont là, sinon combiner un autre composant, pour ensuite tester à nouveau, et ainsi de suite jusqu'à ce que l'ensemble du système développé est testé.

Tests d'intégration Top-Down

Le test d'intégration Top-down est une approche progressive pour la construction de programmes structurés. Les modules sont intégrés en se déplaçant vers le bas à travers la hiérarchie de commande en commençant par le module de commande principal.

Intégration ascendante

Les tests d'intégration Bottom-up, comme son nom l'indique, commence la construction et les essais avec les composants au niveau le plus bas dans la structure du programme. Une stratégie d'intégration bottom-up peut être mise en œuvre avec les étapes suivantes :

Les composants de bas niveau sont combinés en grappes qui effectuent des sous-fonctions logicielles spécifiques.

- Un programme de contrôle pour le test (Pilote) est écrit pour coordonner les cas d'entrée et de sortie.
- Le cluster est testé.
- Les pilotes sont supprimés et les groupes sont combinés en mouvement vers le haut dans la structure du programme.

Test de régression

Les tests de régression c'est l'activité qui contribue à s'assurer que les changements (en raison de tests ou pour d'autres raisons) n'introduisent pas un comportement involontaire ou d'autres erreurs. La suite de tests de régression contient trois classes différentes de cas de test :

- Les tests supplémentaires qui mettent l'accent sur les fonctions du logiciel.
- Un échantillon représentatif de tests qui exercera toutes les fonctions logicielles.
- Tests qui se concentrent sur les composants logiciels qui ont été modifiés.

Smoke Test

Le Smoke Test est une approche de tests d'intégration qui est couramment utilisé lorsque des produits logiciels "rétractables" sont développés. Il se caractérise par une approche d'intégration de roulement, car le logiciel est reconstruit avec de nouveaux composants et des essais.

Le Smoke Test englobe les activités suivantes :

Les composants logiciels qui ont été traduits en code sont intégrés dans un "build". Une version inclut tous les fichiers de données, des bibliothèques, des modules réutilisables et composants techniques qui sont nécessaires pour mettre en œuvre une ou plusieurs fonctions de produit.

Une série de tests est conçu pour exposer les erreurs qui empêchent l'ensemble du produit de bien remplir ses fonctions.

Le Build est intégrée avec d'autres Build et l'ensemble du produit (dans sa forme actuelle) est testé selon le Smoke Test, tous les jours.

Le Smoke Test fournit un certain nombre d'avantages lorsqu'il est appliqué sur des projets d'ingénierie de logiciels complexes :

- risque d'intégration est réduit au minimum.
- qualité du produit final est améliorée.
- diagnostic et correction des erreurs sont simplifiées.
- Le progrès est plus facile à évaluer.

Test d'intégration Sandwich

Le Test d'intégration Sandwich est la combinaison du haut vers le bas (l'approche bottom-up). Donc, il est aussi appelé test d'intégration mixte. Dans ce cas, l'ensemble du système est divisé en trois couches, comme un sandwich : la cible est au milieu et une couche est au-dessus de la cible et une autre en dessous de la cible. L'approche top-down est utilisée dans la couche qui est au-dessus de la cible et l'approche bottom-up est utilisée dans la couche qui est en dessous de la cible.

Test du système

Dans le test du système, les sous-systèmes sont intégrés pour rendre l'ensemble du système. Le processus de test est préoccupé de trouver des erreurs qui résultent d'interactions imprévues entre les sous-systèmes et composants du système. Il est également préoccupé de valider que le système répond à ses exigences fonctionnelles et non fonctionnelles. Il existe trois principaux types essentiellement de tests du système :

Test Alpha

se réfère à l'essai du système effectué par l'équipe de test au sein de l'organisation de développement.

Le test alpha est effectué sur le site du développeur par le client sous la direction de l'équipe de projet.

Dans ce cas de test, les utilisateurs tentent de tester le logiciel sur la plate-forme de développement et de souligner les erreurs de correction. Cependant, le test alpha a une capacité limitée de détecter les erreurs et de les corriger.

Les tests Alpha sont effectués dans un environnement contrôlé. Il est une simulation de l'utilisation de la vie réelle. Une fois que le test alpha est terminé, le produit logiciel est prêt pour la transition vers le site du client pour la mise en place et l'exploitation.

Test Beta

Le Beta test est le test du système effectué par un groupe de clients sélectionnés.

Si le système est complexe, le logiciel n'est pas mis en œuvre directement. Il est installé et tous les utilisateurs sont invités à utiliser le logiciel en mode de test; Ceci est appelé le bêta-test.

Les tests bêta sont effectués sur le site du client dans un environnement où le logiciel est exposé à un certain nombre d'utilisateurs. Le développeur peut ou non être présent pendant que le logiciel est en cours d'utilisation. Ainsi, le bêta test est une expérience logicielle en situation réelle, sans mise en œuvre effective. Dans ce test, les utilisateurs finaux enregistrent leurs observations, les erreurs, les erreurs, et ainsi de suite et font des rapports périodiques.

Dans un bêta-test, l'utilisateur peut suggérer une modification, un changement majeur, ou une déviation. Le développement doit examiner le changement proposé et le mettre dans le système de gestion du changement pour un changement en douceur du logiciel. Il est une pratique courante de mettre tous ces changements dans des versions ultérieures.

Test d'admission

Le test d'acceptation est le test du système effectué par le client pour décider d'accepter ou de rejeter la livraison du système.

Lorsque le logiciel client est construit pour un client, une série de tests d'acceptation sont effectués pour permettre au client de valider toutes les exigences.

Dirigé par l'utilisateur final plutôt que les ingénieurs logiciels, un test d'acceptation peut varier d'un «test drive» informelle à une série planifiée de tests.

En fait, les tests d'acceptation peuvent être effectués sur une période de plusieurs semaines ou mois, découvrant ainsi les erreurs cumulées qui pourraient dégrader le système au fil du temps.

Évaluation

Vérifiez votre compréhension!

1. Quels sont les différents niveaux de tests? Explique.
2. Logiciel développé Supposons a passé avec succès tous les trois niveaux de tests, à savoir, les tests unitaires, tests d'intégration et les tests du système. Peut-on prétendre que le logiciel est exempt de défauts? Justifie ta réponse.
3. Qu'est-ce que les tests unitaires?
4. Qu'est-ce que les tests d'intégration? Quels types de défauts sont découverts lors de tests d'intégration?
5. Qu'est-ce que les tests de régression? Quand les tests de régression fait? Comment les tests de régression effectuée?
6. Qu'est-ce que les tests du système? Quels sont les différents types de tests de système qui sont habituellement effectuées sur de grands produits logiciels?
7. Définir les tests de sandwich.
8. Pourquoi les tests de régression est important? Quand est-il utilisé?

Activité 4: Test boite blanche, Test boite noire (White-Box, Black Box-Testing)

White-Box Testing / Structural Testing

Une approche complémentaire aux tests fonctionnels (dits de boîte noire), est le test structurel ou boîte blanche. Dans cette approche, les groupes d'essai doivent avoir une connaissance complète de la structure interne du logiciel. Le test boite blanche est une approche de test où les tests sont tirés de la connaissance de la structure après la mise en œuvre du logiciel. Il est généralement appliqué aux unités de programme relativement petites, telles que les sous-programmes, ou les opérations associées à un objet.

Comme son nom l'indique, le testeur peut analyser le code et utiliser les connaissances sur la structure d'un composant pour obtenir des données d'essai comme le montre la figure 4.5. L'analyse du code peut être utilisée pour savoir combien de cas de tests sont nécessaires pour garantir que toutes les déclarations contenues dans le programme sont exécutées au moins une fois au cours du processus de test.

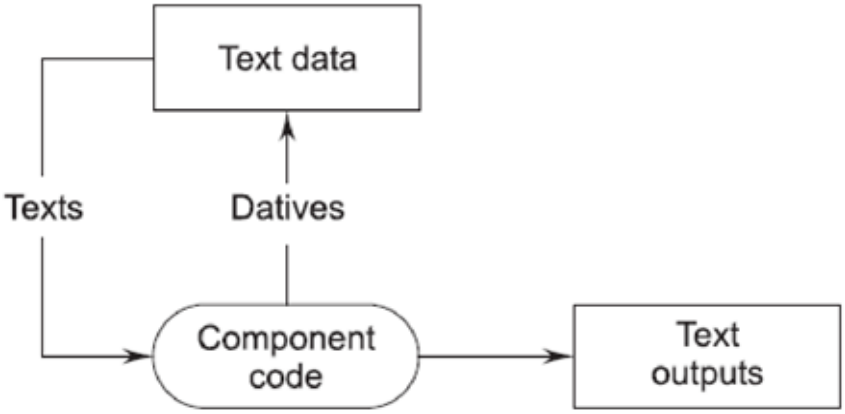


Figure 4.5: Test boite blanche (Source: Agarwal et al., 2010)

Dans les tests boîte blanche, les cas de test sont sélectionnés sur la base de l'examen du code, plutôt que sur la base des spécifications. Le test boîte blanche est illustré à la figure 4.6.

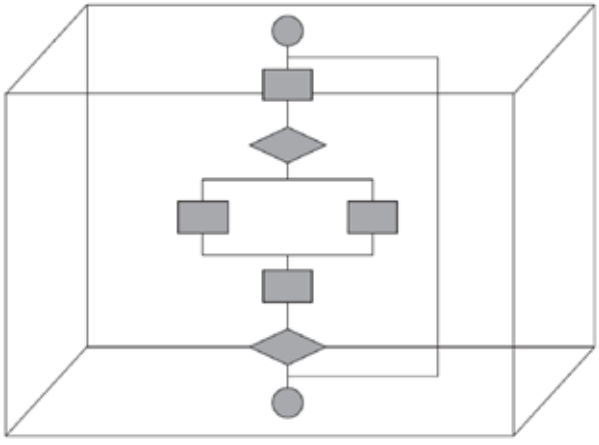


Figure 4.6: White-Box Testing (Source: Agarwal et al., 2010)

En utilisant des méthodes de test boîte blanche l'ingénieur logiciel permet de tester les cas :

- Garantir que tous les chemins indépendants au sein d'un module ont été examinés au moins une fois.
- Exercer toute décision logique sur leurs côtés vrais et faux.
- Exercer toutes les boucles à leurs limites.
- Contrôler les structures de données internes pour assurer leur validité.

La nature des défauts logiciels sont :

- Les erreurs logiques et des hypothèses incorrectes sont inversement proportionnelles à la probabilité qu'un chemin de programme sera exécuté.
- Nous croyons souvent qu'un chemin logique ne doit pas être exécuté lorsque, en fait, il peut être exécuté sur une base régulière.

les erreurs typographiques sont aléatoires. Quand un programme est traduit en code source de langage de programmation, il est probable que certaines erreurs de frappe se produiront.

Raisons pour effectuer les tests white-box

Pour vérifier si:

- Tous les chemins dans un processus sont correctement opérationnels.
- Toutes les décisions logiques sont exécutées avec des conditions vraies et fausses.
- Toutes les boucles sont exécutées avec leurs valeurs limites testées.
- Pour vérifier si les spécifications de structure de données d'entrée sont testées et utilisées pour un autre traitement.

Avantages de construction de Tests boîte blanche

Les divers avantages de tests boîte blanche comprennent :

- Aide le développeur de test à raisonner soigneusement pour la mise en œuvre.
- se rapproche de la répartition effectuée par l'exécution d'équivalence.
- Révèle des erreurs dans le code caché.

Tests fonctionnels boite noire (Black-Box)

Dans les tests fonctionnels, la structure du programme n'est pas considérée. Les cas de test sont décidés sur la base des exigences ou des spécifications du programme ou le module et le fonctionnement interne du module ou du programme ne sont pas considérés pour la sélection des cas de test.

Les tests fonctionnels se réfère à des tests qui implique que l'observation de la sortie pour certaines valeurs d'entrée, et il n'y a aucune tentative d'analyser le code, qui produit la sortie.

La structure interne du programme est ignorée. Pour cette raison, les tests fonctionnels est parfois appelé test boîte noire (également appelé test comportemental) dans lequel le contenu d'une boîte noire est pas connue et la fonction de la boîte noire est parfaitement compris en termes de ses entrées et sorties.

Le test boîte noire, aussi appelé test comportemental, met l'accent sur les exigences fonctionnelles du logiciel. Il permet à l'ingénieur logiciel d'obtenir des ensembles de conditions d'entrée qui s'appliquent pleinement à toutes les exigences fonctionnelles pour un programme.

D'autres noms pour les tests de boîte noire (BBT) comprennent des tests de spécifications, les tests de comportement, les tests pilotés par les données, les tests fonctionnels, et d'entrée / test de sortie entraîné. Dans les tests de boîte noire, le testeur ne connaît que les entrées qui peuvent être donnés au système et ce que la sortie du système devrait donner. En d'autres termes, la base pour décider des cas de test dans les tests fonctionnels sont les exigences ou les spécifications du système ou d'un module. Cette forme de test est aussi appelé test fonctionnel ou comportemental.

Les tests boîte noire ne sont pas une alternative aux techniques de tests boîte blanche; au contraire, elle est une approche complémentaire qui est susceptible de découvrir une autre classe d'erreurs que les méthodes boîte blanche. Le test boîte noire identifie les types d'erreurs suivants:

- Mauvaises ou fonctions manquantes.
- Interface manquante ou erronée.
- Les erreurs dans le modèle de données.
- Les erreurs d'accès à la source de données externe.

Lorsque ces erreurs sont contrôlées alors:

- Fonction(s) sont valides.
- Une classe d'entrées est validée.
- La validité est sensible à certaines valeurs d'entrée.
- Le logiciel est valide et fiable pour un certain volume de données ou de transactions.
- Combinaisons spécifiques rares sont prises en charge.

Le test boîte noire tente de répondre aux questions suivantes :

- Comment est la validité fonctionnelle testée?
- Comment sont le comportement et les performances du système testé?
- Comment sont les limites d'une classe de données isolées?
- Comment les combinaisons de données spécifiques affectent le fonctionnement du système?

- Quels sont les taux de données et volumes de données que le système peut tolérer?
- Le système est particulièrement sensible à certaines valeurs d'entrée?
- Quel effet des combinaisons spécifiques de données auront sur le fonctionnement du système?

En appliquant des techniques de boîte noire, nous obtenons un ensemble de cas de test qui satisfont aux critères suivants:

- Les cas de test qui réduisent par un nombre qui est supérieur à un.
- Les cas de tests qui nous disent quelque chose sur la présence ou l'absence de classes d'erreurs.

Avantages de Test boîte noire

Les avantages de ce type de test comprennent :

- Le test est biaisé parce que le concepteur et le testeur sont indépendants les uns des autres.
- Le testeur n'a pas besoin de la connaissance de tous les langages de programmation spécifiques.
- Le test est effectué à partir du point de vue de l'utilisateur, pas du concepteur.
- Les cas de tests peuvent être conçus dès que les spécifications sont complètes.

Plan de Test

Le plan d'essai est un document qui contient des cas de test différents conçus pour tester différents objets de test et les différents attributs de test. Le plan met l'essai d'un formulaire de commande logique et séquentielle selon la stratégie choisie, de haut en bas ou de bas en haut. Habituellement, le plan d'essai est une matrice de test et la liste des cas de test en conformité avec l'ordre d'exécution de chaque tâche. Le tableau 4.1 illustre la matrice de cas de test et d'essai dans le test.

ID de test, le nom du test, et les cas de tests sont bien conçus avant la phase de développement et ont été conçus pour ceux qui effectuent les tests. Un plan de test états :

- Les éléments à tester.
- A quel niveau ils seront testés à.
- La séquence qu'ils sont à tester dans.
- Comment la stratégie de test sera appliquée à l'essai de chaque élément et l'environnement de test.

Table 4.1: Test Plan (Source: Agarwal et al., 2010)

										Test
Planned Date		N	...	4	3	2	1	Test	Test ID	
Successful	Completed							Name	Tester	ID

Un cas de test est un ensemble d'instructions destinées à découvrir un type particulier d'erreur ou d'un défaut dans le système de logiciel en induisant un échec. L'objectif de cas de test sélectionnés est d'assurer qu'il n'y a pas d'erreur dans le programme et s'il est alors doit être immédiatement représenté.

Une fenêtre de test idéale doit contenir toutes les entrées au programme. Ceci est souvent appelé test exhaustive.

Il y a deux critères de sélection des cas de test :

- Spécification d'un critère pour évaluer un ensemble de cas de test.
- Génération d'un ensemble de cas de test qui satisfont à un critère donné.

Chaque cas de test a besoin de la documentation appropriée, de préférence dans un format fixe. Il existe de nombreux formats; un format est suggéré dans le tableau 4.2:

Table 4.2: Test Case Documentation Format (Source: Agarwal et al., 2010)

Test Case Name	Test Case ID
Purpose of Test	Testing Object (Unit, Application, Module, etc.)
Test Attribute	
Tests focus (function, feature, process, interface, validation, verification, etc.)	
Test type (alpha, beta, unit, intégration, system)	
Test Process	A set of instructions for conducting the test-initial stating Condition -inputs-spécifications-output excepte
Test Results	Expected and actual and comparison, error description, post-process state
Action	Correction, authorization, and feedback through retest
Action to initialisé the pré-test status	

Évaluation

Vérifiez votre compréhension!

1. Qu'est-ce qu'un cas de test? Quelle est la conception de cas de test?
2. Quelle est la différence entre

   a) test boîte noire et tests boîte blanche

   b) Top-down et les approches de test bottom-up

   c)les essais alpha et bêta
3. Quels sont les plans de test et des cas de test? Illustrer chacun avec un exemple.
4. Pourquoi les tests de logiciels a besoin une planification? Explique.
5. Que sont les Smoke tests ?
6. Différencier les tests d'intégration et les tests du système.
7. Définir des tests structurels. Donner les diverses raisons pour lesquelles le test structurel est effectué.
8. Expliquer les deux catégories de tests boîte noire. Indiquez également les avantages des tests de boîte noire.

Résumé de l'unité

Dans cette unité, les aspects liés à la mise en œuvre / phase et les tests de codage ont été discutés. Dans la phase de codage, une programmation structurée a été expliqué plus. L'objectif principal pour la conception de cas de test est d'obtenir un ensemble de tests qui ont la plus forte probabilité pour découvrir des erreurs dans le logiciel. Pour atteindre cet objectif, deux catégories différentes de techniques de conception de cas de test sont utilisés: tests boîte blanche et les tests de boîte noire.

Les tests boîte blanche se concentrent sur la structure de contrôle du programme. Les cas de test sont dérivés pour assurer que toutes les déclarations contenues dans le programme ont été exécutées au moins une fois au cours des essais et que toutes les conditions logiques ont été exercées. Les test black-box sont conçus pour valider les exigences fonctionnelles sans tenir compte du fonctionnement interne d'un programme.

La différence entre les différentes techniques de tests tels que les tests unitaires, l'intégration et le système a été expliqué en détail. La nécessité de mettre en œuvre un plan de test, ou un cycle de vie des tests logiciels et comment les techniques et les outils sont intégrés dans ce qui a été discuté dans ce chapitre.

Évaluation de l'unité

Vérifiez votre compréhension!

1. Qu'entendez-vous par le logiciel
2. Qu'est-ce et des logiciels d'ingénierie?
3. Quels sont les différents mythes et la réalité sur le logiciel?
4. D'un détail expliquer le processus de génie logiciel

Lectures et autres ressources

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783
- Zhiming Liu, (2001), "Object-Otiented Software Development Using UML", The United Nations University, UNU-IIST International Institute for Software Technology, Tech Report 229.

Unité 5. Maintenance et Gestion de projet

Introduction à l'unité

Le Logiciel va certainement subir un changement après la livraison au client. Cette phase de maintenance démarre avec le système en cours d'installation pour une utilisation pratique, après que le produit est livré au client. Elle dure jusqu'au début de la phase de retrait du système. La maintenance n'apporte pas normalement de changements majeurs à l'architecture du système. Les modifications sont mises en œuvre par la modification des composants existants et en ajoutant de nouveaux composants au système. La maintenance comprend toutes les modifications apportées au produit une fois que le client l'a accepté et attesté qu'il satisfait le document de spécification.

Le génie logiciel est une activité difficile. Beaucoup de choses peuvent aller mal. Comprendre les risques et prendre des mesures proactives pour les éviter ou les gérer est un élément clé pour la production d'un bon logiciel.

Objectifs de l'unité

À la fin de cette unité, vous devriez être capable de :

- Expliquez pourquoi la maintenance des logiciels est nécessaire
- Décrire les catégories maintenance
- Décrire les facteurs affectant la maintenance
- Définir l'analyse et la gestion des risques
- Décrire les catégories de risques et gestion des risques
- Effectuer les plans de test

Termes clés

Maintenance logicielle: C'est l'activité associée avec le maintien d'un système informatique opérationnel en permanence en phase avec les besoins des utilisateurs et des opérations de traitement de données.

Analyse et gestion du risque logiciel: L'analyse et la gestion du risque logiciel consistent en une série d'étapes qui aident à comprendre et à gérer l'incertitude.

Risque: Le risque est défini comme une exposition au danger (détérioration, altération ou perte, etc.)

Gestion des risques: La gestion des risques est le domaine qui tente de veiller à ce que l'impact des risques sur le coût, la qualité, et le calendrier soit minime.

Activités d'apprentissage

Activité 1: La phase de maintenance du logiciel

Introduction

La maintenance du logiciel est l'activité associée avec le maintien, en permanence, d'un système informatique en condition opérationnelle en phase avec les besoins des utilisateurs et des opérations de traitement de données. La maintenance du logiciel est une activité très large qui inclut des corrections d'erreurs, des améliorations des capacités, la suppression des fonctionnalités obsolètes et l'optimisation. Le processus de maintenance des logiciels est coûteux et risqué et il est très difficile. Il existe un besoin de maintenance des logiciels pour les raisons suivantes :

- Les besoins des utilisateurs changent avec le temps. Ainsi, le client peut exiger des améliorations fonctionnelles ou de performances
- Les problèmes de programme/système, des erreurs logicielles
- Le logiciel doit être adapté pour tenir compte des changements dans son environnement externe (par exemple, un changement nécessaire en raison d'un nouveau système d'exploitation ou d'un périphérique), - Modification de l'environnement du matériel / logiciel
- Pour améliorer l'efficacité du système
- Pour modifier les composants
- Pour tester le produit résultant et vérifier l'exactitude des modifications
- Pour éliminer les effets indésirables résultant des modifications
- Pour augmenter ou affiner le logiciel
- Pour optimiser le code afin d'accélérer son exécution
- Pour contrôler l'efficacité et le respect des normes
- Pour rendre le code plus facile à comprendre et à utiliser
- Pour éliminer tout écart par rapport aux spécifications

Catégories de Maintenance

La maintenance logicielle peut être classée dans les quatre catégories suivantes :

- Maintenance corrective - modifications apportées pour corriger les problèmes découverts.
- Maintenance adaptative - modifications pour le maintenir utilisable dans un environnement modifié ou changeant.
- Maintenance perfective - améliorer la performance ou la maintenabilité du logiciel.
- Maintenance préventive - modifications pour détecter et corriger des pannes éventuelles.

a) Maintenance corrective

La maintenance corrective signifie la réparation des erreurs de traitement ou de performance ou le fait d'apporter des modifications en raison de problèmes précédemment non corrigés. Elle permet de corriger des erreurs qui ne sont pas découvertes dans les premiers stades du processus de développement, tout en laissant la spécification inchangée.

b) Maintenance Adaptative

La maintenance adaptative permet de changer les fonctions du programme. Ceci est fait pour l'adapter à l'évolution de l'environnement extérieur comme par exemple les nouvelles réglementations gouvernementales. Ce type est connu comme une maintenance d'amélioration.

c) Maintenance Perfective

La maintenance Perfective est l'un des moyens d'amélioration de la performance et permet aussi de modifier les programmes pour répondre aux besoins supplémentaires de l'utilisateur. Elle implique des changements que le client estime qu'ils permettront d'améliorer l'efficacité du produit, par l'ajout de fonctionnalités supplémentaires ou par la diminution du temps de réponse, par exemple. Ce type est aussi une sorte de maintenance d'amélioration.

d) Maintenance préventive

La maintenance préventive est le processus de prévention des systèmes contre l'obsolescence. La maintenance préventive implique le concept de re-engineering et de l'ingénierie dans lequel un ancien système avec une technologie ancienne est repensé en utilisant une nouvelle technologie. Cette maintenance empêche le système de disparaître.

Des études ont montré que, en moyenne, les mainteneurs passent environ 17% de leur temps sur la maintenance corrective, 65% sur la maintenance perfective, et 18% sur la maintenance adaptative comme le montre la figure 5.1 (Sommeville, 2000). On estime qu'entre 40% et 70% des coûts globaux du cycle de vie de développement de logiciels sont consacrés à la maintenance.

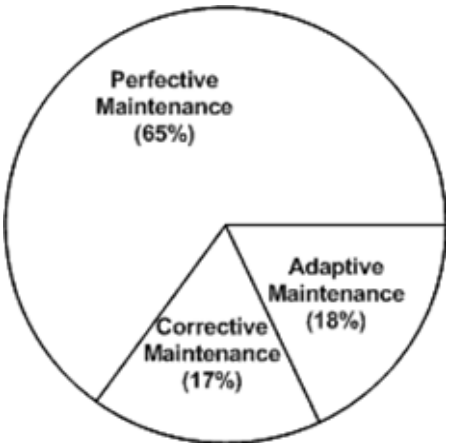


Figure 5.1: Répartition de l'effort de maintenance (Source: Sommeville, (2000))

Coûts de maintenance

Le coût de maintenance est généralement supérieur aux coûts de développement (2 fois à 100 fois en fonction de l'application), il est affecté à la fois par des facteurs techniques et non techniques. Les coûts de maintenance augmentent au fur et à mesure que le logiciel est maintenu. La maintenance modifie la structure logicielle rend encore les futures opérations de maintenance plus difficiles.

Il est conseillé d'investir plus d'efforts dans les premières phases du cycle de vie du logiciel pour réduire les coûts de maintenance. Le ratio augmente fortement avec le défaut de réparation dans une phase (de la phase d'analyse à la phase de mise en œuvre comme indiqué dans le tableau 5.1). Par conséquent, plus d'efforts au cours du développement, pour en réduire les défauts, va certainement réduire le coût de maintenance.

Table 5.1: Ratio / défaut de réparation

Phase	Ratio
Analyse	1
Conception	10
Implémentation (développement)	100

Facteurs influant sur la maintenance

Il y a beaucoup d'autres facteurs qui contribuent à l'effort nécessaire pour maintenir un système. Ces facteurs comprennent les éléments suivants :

Nouvelle application

Comme les utilisateurs acquièrent de l'expérience avec l'utilisation fréquente du logiciel, ils vont commencer à souhaiter une amélioration et de nouvelles caractéristiques potentielles

Mobilité du personnel

Il est toujours plus facile pour les programmeurs d'origine de mettre à jour le code par rapport à quelqu'un d'autre. Lorsque le personnel change, il devient plus difficile de maintenir le code à moins ce dernier soit très bien documenté.

Évaluation

Vérifiez votre compréhension!

1. Qu'est-ce que la maintenance du logiciel ? Décrire les différentes catégories de maintenance. Quelle catégorie consomme le plus d'effort et pourquoi ?
2. Certaines personnes pensent que « la maintenance est gérable ». Quelle est votre opinion sur cette question ?
3. Expliquer les facteurs qui influent sur le coût de maintenance.
4. Expliquer les différents types de maintenance.
5. Pourquoi la maintenance est-elle nécessaire?
6. Quels sont les différents types de maintenance qu'un logiciel pourrait avoir besoin ? Pourquoi un tel entretien est nécessaire?

Activité 2: Analyse et gestion des risques de logiciels

Introduction

L'analyse et la gestion des risques sont une série d'étapes qui aident à comprendre et à gérer l'incertitude. Un risque est un problème potentiel qui pourrait (ou ne pas) se produire. Mais, dans tous les cas, il est important de l'identifier, d'évaluer sa probabilité d'occurrence, d'estimer son impact, et d'établir un plan d'urgence au cas où le problème devrait effectivement se produire.

Le risque est défini comme une exposition sur le risque d'altération ou de perte. Autrement dit, le risque implique qu'il existe une possibilité que quelque chose de négatif puisse se produire. Dans le cadre de projets logiciels, le négatif signifie qu'il y ait un effet négatif sur le coût, la qualité ou la prévision calendaire.

La gestion des risques est domaine d'étude qui tente de veiller à ce que l'impact des risques sur le coût, la qualité, et le calendrier soit minime

Risques logiciels

Le risque implique toujours deux caractéristiques :

- Incertitude : le risque pourrait ou ne pas se produire ; il n'y a donc pas de risque probable à 100%. Sinon cela devient un incident, une réalité.
- Perte : si le risque devient une réalité, les conséquences ou les pertes non désirées se produisent.

Lorsque les risques sont analysés, il est important de quantifier le degré d'incertitude et le degré de perte associé à chaque risque. Pour ce faire, différentes catégories de risques sont prises en compte.

Risques du projet :

Les risques du projet menacent le plan de projet. Autrement dit, si les risques du projet deviennent réalité, il est probable que le calendrier va glisser et que les coûts augmenteront. Les risques du projet identifient le potentiel budgétaire, le calendrier, le personnel (dotation en personnel et de l'organisation), les ressources, les clients et les exigences des problèmes et leur impact sur un projet de logiciel.

Risques techniques :

Les risques techniques menacent la qualité et temps de production du logiciel. Si un risque technique devient une réalité, la mise en œuvre peut devenir difficile, voire impossible. Les risques techniques identifient la conception potentielle, la mise en œuvre, l'interface, la vérification et les problèmes de maintenance. En outre, l'ambiguïté de la spécification, l'incertitude technique, l'obsolescence matérielle et technique, l'innovation "de pointe" sont également des facteurs de risque. Quand les risques techniques se produisent, le problème devient plus difficile à résoudre.

Risques commerciaux

Les risques commerciaux menacent la viabilité du logiciel à construire. Les risques commerciaux compromettent souvent le projet ou le produit. Les cinq principaux risques d'affaires (i.e. commerciaux) sont :

- Construction d'un excellent produit ou système que personne ne veut vraiment (risque de marché),
- Construction d'un produit qui ne correspond plus à la stratégie commerciale globale de l'entreprise (risque stratégique),
- Construction d'un produit que la force de vente ne comprend pas comment vendre,
- La perte du soutien de la haute direction en raison d'un changement d'orientation ou un changement de personnes (risque de gestion),
- La perte (risques budgétaires ou mobilité du personnel).

Les risques prévisibles sont extrapolés à partir de l'expérience des projets passés (par exemple, la rotation du personnel, une mauvaise communication avec le client, la dilution de l'effort personnel que les demandes de maintenance en cours sont desservies). Les risques imprévisibles sont le joker dans le jeu. Ils peuvent se produire, mais ils sont extrêmement difficiles à identifier à l'avance.

Identification des risques

L'identification des risques est une tentative systématique d'identification des menaces pour le plan de projet (estimations, calendrier, disponibilité des ressources, etc.). Un premier pas vers la prévention des risques lorsque cela est possible et de les contrôler si nécessaire en identifiant les risques connus et prévisibles. Il existe deux types distincts de risques pour chacune des catégories qui ont été présentés ci-dessus : les risques génériques et des risques spécifiques aux produits.

- Les risques génériques sont une menace potentielle pour chaque projet de logiciel.
- Les risques spécifiques aux produits ne peuvent être identifiés que par ceux qui ont une compréhension claire de la technologie.

Pour identifier les risques spécifiques aux produits, le plan du projet et la cible du logiciel sont examinés en vue d'apporter une réponse à la question suivante :

"Quelles sont les caractéristiques particulières de ce produit qui pourraient menacer le plan de projet?"

Une méthode d'identification des risques consiste à créer un inventaire des risques. Cet inventaire peut être utilisé pour l'identification des risques et permettre de se concentrer sur un sous-ensemble de risques connus et prévisibles dans les sous-catégories génériques suivantes :

- Taille du produit : risques associés à la taille globale du logiciel à construire ou à modifier.
- Business Impact : risques associés aux contraintes imposées par la direction ou par la réalité du marché.
- Caractéristiques clients : risques associés à la nature du client et à la capacité du développeur à communiquer avec le client en temps opportun.
- Définition du processus : risques associés au niveau dans lequel le processus logiciel a été défini et suivi par l'équipe de développement.
- Environnement de développement : risques liés à la disponibilité et la qualité des outils à utiliser pour développer le produit.
- Technologique : risques intégrés à la complexité du système à construire et à la « nouveauté » de la technologie.
- La taille du personnel et son expérience : risques associés à l'expertise technique et à l'expérience globale des ingénieurs par rapport au projet logiciel.

L'inventaire des risques peut être organisé de différentes manières. Des questions pertinentes à chacun des sujets peuvent être répondues pour chaque projet de logiciel. Les réponses à ces questions permettent au planificateur d'estimer l'impact du risque. Une autre forme d'identification des risques énumère simplement les caractéristiques qui sont pertinentes pour chaque sous-catégorie générique. Enfin, un ensemble de « composants de risque et les pilotes » sont répertoriés avec leur probabilité d'occurrence.

Bien que les risques génériques soient importants à considérer, généralement les risques spécifiques aux produits causent le plus de mal. Assurez-vous d'identifier autant de risques spécifiques que possible.

Catégories de gestion des risques

La gestion des risques joue un rôle important pour s'assurer que le produit logiciel est exempt d'erreurs. Tout d'abord, la gestion des risques veille à ce que le risque soit évité, et s'il ne peut être évité, alors le risque est détecté, contrôlé, et finalement contenu.

Comme il est précisé dans la figure 5.2, la gestion des risques peut être classée comme suit :

a) Prévention des risques

- L'anticipation des risques
- Outils de risque

b) Détection des risques

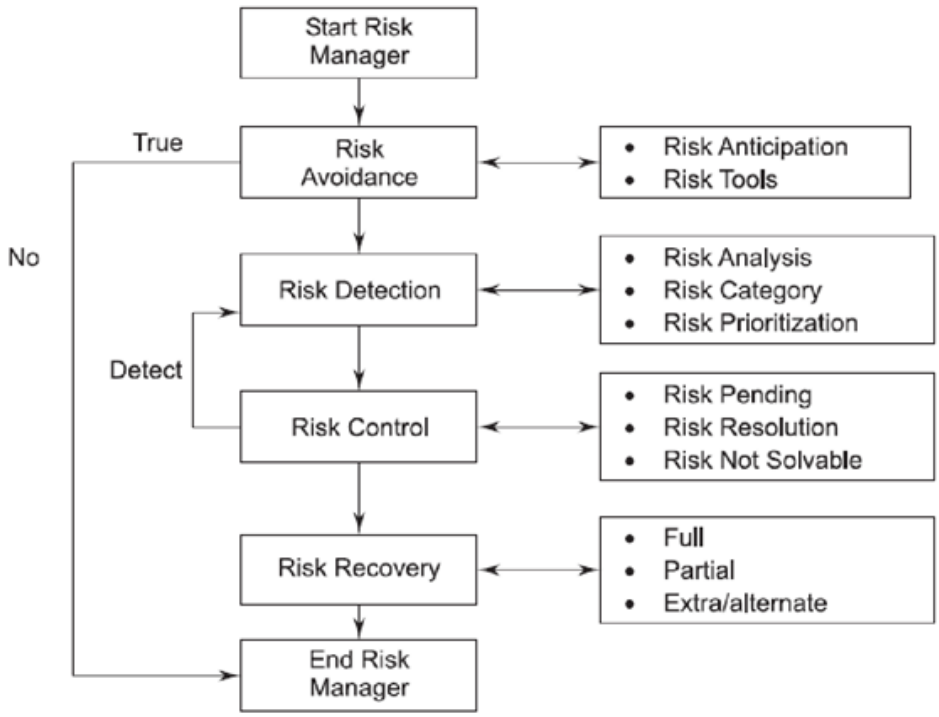
- Analyse de risque
- Catégorie de risque
- Hiérarchisation et priorisation des risques

c) Contrôle des risques

- Risques appréhendés
- Résolution des risques
- Risques résiduels

d) Recouvrement des risques

- Plein
- Partiel
- Fonction Extra / alternatif



Risk-Management Toll

La première phase consiste à éviter les risques en anticipant et en utilisant les outils de l'historique des projets précédents. Dans le cas où il n'y a pas de risque, le gestionnaire des risques arrête. Dans le cas des risques, la détection se fait en utilisant diverses techniques d'analyse des risques. Ensuite, le risque est contrôlé, résolu, et dans le pire des cas (si le risque est pas résolu), en abaissant la priorité. Enfin, la résolution des risques se fait entièrement, partiellement, ou une autre solution est trouvée.

Sources de risques

Il existe deux sources principales de risques :

a) Les risques génériques

Risques communs à tous les projets logiciels. Par exemple, l'exigence malentendu, permettant suffisamment de temps pour les tests, perdre le personnel clé, etc.

b) Risques spécifiques au projet

Un vendeur peut faire la promesse de fournir des logiciels notamment pour une date donnée, alors qu'en réalité l'équipe de développement est incapable de le faire.

Évaluation

Vérifiez votre compréhension!

1. Lister trois types courants de risques qu'un projet de logiciel typique pourrait courir.
2. Quelle est l'activité de gestion des risques ? Est-il rentable de faire la gestion des risques ? Quel est l'effet de cette activité sur le coût global du projet ?
3. Quelles sont les deux principales sources de risques ? Expliquer les trois catégories de risques qu'un projet de logiciel pourrait subir.
4. Fournir cinq exemples, issus d'autres domaines, qui illustrent bien les problèmes associés à une stratégie de réaction au risque.
5. Décrire la différence entre les « risques connus » et « risques prévisibles ».

Résumé de l'unité

Cette unité a décrit les aspects de la maintenance des logiciels et de la gestion de projet en montrant que cette dernière inclut davantage aussi la gestion des risques.

Évaluation de l'unité

Vérifiez votre compréhension!

1. Qu'entendez-vous par le logiciel
2. Qu'est-ce et des logiciels d'ingénierie?
3. Quels sont les différents mythes et la réalité sur le logiciel?
4. D'un détail expliquer le processus de génie logiciel

Lectures et autres ressources

- Agarwal B. B., Tayal S. P. and Gupta M., (2010), "Software Engineering & Testing, an Introduction", Jones and Bartlett Publishers, ISBN: 978-1-934015-55-1
- Pressman Roger S., (2001), "Software Engineering, A Practitioner' S Approach" Fifth Edition, McGraw-Hill Higher Education, ISBN 0073655783
- Zhiming Liu, (2001), "Object-Otiented Software Development Using UML", The United Nations University, UNU-IIST International Institute for Software Technology, Tech Report 229.

Siège de l'Université Virtuelle Africaine

The African Virtual University
Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

Bureau Régional de l'Université Virtuelle Africaine à Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org

2017 UVA