

IFT2251
Introduction au génie logiciel
Hiver 2006 (4 crédits)
Prof. : Julie Vachon

** Début des cours : le lundi 9 janvier 2006 **

Plan de cours

1. Introduction

Les exigences et les attentes à l'égard de la qualité logicielle sont de plus en plus grandes. La taille et la complexité des systèmes informatiques ne cessent d'augmenter. Le développement logiciel ne peut plus être le simple fait du travail "artisanal" d'un développeur aussi ingénieux soit-il. Considérant le rôle critique de certains logiciels, le coût de leur réalisation et les courts échéanciers de production, il est aujourd'hui essentiel que le développement logiciel puisse être soutenu par des mécanismes pertinents de planification, d'analyse, de conception et de vérification. La culture de "bonnes pratiques" et l'application de certains principes de développement contribuent à l'élaboration de systèmes fiables, performants et facilement modifiables.

Description du cours et objectifs généraux

Le cours IFT2251 est une introduction au génie logiciel. Il vise à initier les étudiants aux processus de développement des logiciels, ainsi qu'aux méthodes, techniques et outils utilisés pour développer des logiciels de qualité, c'est à dire corrects, fiables, robustes, facilement maintenables, évolutifs, etc.

L'objectif général de ce cours est de donner à l'étudiant une bonne connaissance du cycle de vie du logiciel et des enjeux liés à chacune des phases du développement.

Nous verrons comment faire l'analyse et la conception de logiciels à partir des concepts, méthodes et techniques proposées par l'approche orientée objet. Les étudiants apprendront à interpréter et à concevoir des modèles décrivant la structure et le comportement de systèmes informatiques tant séquentiels que concurrents. Une partie importante du cours portera sur l'analyse et la conception détaillée de systèmes orientés objets à l'aide d'UML. UML est une notation standard *semi-formelle* utilisée pour spécifier, visualiser, construire et documenter les artefacts des systèmes logiciels. D'autre part, nous verrons également comment faire la spécification *formelle* de certains systèmes plus critiques à l'aide du formalisme des réseaux de Petri.

Pour clore ce cours d'introduction, nous aborderons le sujet de la vérification. Nous survolerons quelques techniques de test (boîte blanche, boîte noire, etc.) et de preuve utilisées pour évaluer la correction des logiciels développés.

Clientèle visée : Étudiants inscrits au DIRO (en 2^e année) ou dans un programme ayant une composante en informatique.

Cours préalables :

IFT1020 - Programmation 2. L'étudiant doit maîtriser les concepts avancés de classes, d'objets, d'héritage, d'interfaces (de classe et utilisateur), de réutilisation, et d'événements. Il doit savoir utiliser les structures de données de base (listes, arbres binaires, fichiers) et les algorithmes de recherche et de tri qui leur sont associés.

Enseignant et démonstrateurs

- **Professeur :**
 - Julie Vachon (vachon@iro.umontreal.ca)
Bureau : P.A.A. 2375
- **Démonstrateurs** (dift2251@iro.umontreal.ca) :

Horaire : Changements possibles.

th		Lun.	10:30	11:30	9 janvier	10 avril	1409	A.-AISENSTADT
th		Mer.	8:30	10:30	11 janvier	12 avril	1409	A.-AISENSTADT
tp		Ven.	10:30	12:30	13 janvier	7 avril	1409	A.-AISENSTADT
<i>intra</i>		<i>Mer.</i>	<i>08:30</i>	<i>10:30</i>	<i>8 mars</i>		<i>1355</i>	<i>A.-AISENSTADT</i>
<i>final</i>		<i>Mer.</i>	<i>08:30</i>	<i>11:30</i>	<i>26 avril</i>		<i>1355</i>	<i>A.-AISENSTADT</i>

2. Objectifs pédagogiques

Introduction

- Définir ce qu'est le génie logiciel, en comprendre les enjeux. Décrire les qualités d'un logiciel et les principes de base qui permettent de les réaliser. Expliquer le cycle de vie d'un logiciel. Décrire et comparer certains processus de développement classiques et plus récents.

Analyse et spécification

- Expliquer les objectifs et les étapes de la phase d'analyse. Décrire les techniques utilisées pour la cueillette d'informations et la spécification.
- Expliquer l'importance de la spécification et distinguer les différents types de notations de spécification existantes.
- Maîtriser les éléments de base de la notation UML :
 - Décrire, interpréter et construire des diagrammes de classes, de cas d'utilisation, d'activités, de séquence et de collaboration.
- À partir d'une étude de cas, réaliser l'analyse d'un système:
 - Compléter la cueillette d'informations et valider l'ensemble des exigences recueillies.
 - Décrire les scénarios des cas d'utilisation du système.
 - Construire le modèle d'analyse du système à l'aide de diagrammes UML.

Conception

- Expliquer les objectifs et les étapes de la phase de conception. Décrire le principe de modularité et savoir l'appliquer. Évaluer les qualités d'une conception logicielle.
- Identifier les différents types d'architectures logicielles fréquemment utilisés, les décrire et les comparer. Justifier le choix d'une architecture particulière pour la conception d'un logiciel donné.
- À partir d'une étude de cas dont on a fait l'analyse, faire la conception détaillée d'un système :
 - Détailler les cas d'utilisation et construire les diagrammes de robustesse correspondants.
 - Raffiner les diagrammes développés pendant la phase d'analyse selon l'analyse de robustesse et l'approche BCE.
 - Savoir appliquer les bonnes pratiques de conception : règles de base et quelques patrons de conception.
- Décrire et distinguer différentes techniques de réutilisation (bibliothèques, framework, patrons de conception, etc.) et justifier leur utilisation.

Méthodes formelles (réseaux de Petri)

- Comprendre la sémantique opérationnelle (i.e. le fonctionnement) des réseaux de Petri.
- Décrire des systèmes séquentiels ou concurrents à l'aide de réseaux de Petri.
- Construire les graphes de marquages et de couverture d'un réseau de Petri.
- Analyser un réseau de Petri pour prouver certaines propriétés du système qu'il décrit.

Vérification

- Identifier, décrire et comparer les différentes approches de vérification logicielle et leurs objectifs.
- Comprendre les objectifs, les avantages et les inconvénients du test logiciel.
- Décrire, comparer et savoir utiliser les techniques de génération de jeux de tests de type « boîte blanche » et « boîte noire » pour le calcul de jeux de test pertinents.

Et plus globalement...

- Développer une approche de développement rigoureuse.
- Apprendre à faire des choix de modélisation et de conception judicieux.

3. Méthode d'enseignement et d'apprentissage

Formule d'enseignement

La matière sera essentiellement présentée sous forme de cours magistraux donnés par le professeur. Des exemples seront développés en cours pour illustrer la matière.

Formule d'apprentissage

Les étudiants pourront développer et confirmer leurs apprentissages en réalisant les exercices proposés par les auxiliaires d'enseignement pendant les séances de travaux dirigés (démonstrations). Les travaux pratiques, en plus de servir à l'évaluation, seront l'occasion de développer d'autres habiletés (rigueur, jugement, organisation, ...) liées à ce cours. Par ailleurs, on s'attend à ce que les étudiants complètent leur apprentissage de façon autonome par des lectures et des exercices tirés des livres mis en réserve à la bibliothèque.

Support de cours

- Transparents
 - Il est recommandé de les compléter par des notes de cours personnelles.
- Site web :
 - www.iro.umontreal.ca/~pift2251
 - On y trouvera les énoncés de travaux pratiques et des démonstrations, les transparents, les annonces importantes, etc.
 - Les informations données en cours prévaudront sur celles affichées sur le site web.

4. Évaluation

L'évaluation comportera deux examens et quatre travaux pratiques. La pondération sera la suivante:

TP #1	5 %
TP #2	10 %
TP #3	15 %
TP #4	10 %
Examen intra	30 %
Examen final	30%

Seuil: Une moyenne de 50% aux examens est exigée pour que les résultats des travaux pratiques soient comptabilisés dans la note finale. (Un étudiant qui obtiendrait 60% à l'intra et 20% au final se verrait attribuer 0 pour les travaux pratiques...)

Retard : À moins d'indications contraires, les travaux remis en retard seront pénalisés selon les taux suivants :

Retard	Pénalité
1 jour	-5%
2 jours	-15%
3 jours	-30%
4 jours	-50%
5-6 jours	-75%
7 jours et plus	-100%

Plagiat et fraude : Sanctionnés conformément aux règlements de l'université de Montréal et du DIRO. Pour plus d'informations :

- http://www.secgen.umontreal.ca/pdf/reglem/francais/sec_30/ens30_3.pdf
- <http://www.iro.umontreal.ca/Codehonneur.pdf>

5. Questions et consultations

Consultation avec le professeur :

Le professeur sera disponible pour répondre à vos questions les lundis de 14h00 à 16h00 ou sur rendez-vous.

Consultation avec les démonstrateurs :

Les démonstrateurs vous communiqueront leurs disponibilités lors de la première séance de travaux pratiques.

Questions par courriel :

Vous pouvez poser vos questions par courriel au professeur (vachon@iro.umontreal.ca) ou aux démonstrateurs (dift2251@iro.umontreal.ca).

6. Ouvrages de référence

Ces livres sont en réserve à la bibliothèque. Vous êtes fortement encouragés à les consulter pour réviser la matière vue en cours, approfondir vos connaissances et y puiser des exercices.

- Carlo Ghezzi, Mehdi Jazayeri et Dino Mandrioli. « Fundamentals of Software Engineering », Prentice-Hall, 2002.
- Leszek A. Maciaszek. « Requirements Analysis and System Design. Developing Information Systems with UML ». Addison Wesley, 2001.
- Shari Lawrence Pfleeger, « Software Engineering: Theory and Practice », 2/E, Prentice-Hall, 2001.
- James Rumbaugh, Ivar Jacobson et Grady Booch, « The Unified Modeling Language Reference Manual ». Addison Wesley, 1999.
- Roger S. Pressman, « Software Engineering, A Practitioner`s Approach ». 5/E, McGraw Hill, 2001.
- Satzinger, Jackson, Burd, Simond et Villeneuve. « Analyse et conception de systèmes d'information ». Les Éditions Reynald Goulet, 2^e édition, 2003.