linfo2252 Software Maintenance and Evolution

2022

5.00 credits	30.0 h + 15.0 h	Q1

Teacher(s)	Mens Kim ;
Language :	English > French-friendly
Place of the course	Louvain-la-Neuve
Main themes	Whereas many software engineering courses focus on building new systems from scratch, in industrial practice software developers are often confronted with already existing software systems that need to be maintained, reused or evolved. This requires specific skills to understand the design and implementation of an existing system and which parts need to be modified, to build software systems that are easier to maintain, and to design systems with reuse and evolution in mind from the very start. This course will thus study a variety of techniques, tools and methodologies to help building software systems that are easier to understand, maintain, reuse and evolve, such as: Preliminaries and definitions : - need for and problems of software maintenance and evolution - definitions, differences between and types of software maintenance and evolution - technical debt - laws of software evolution Domain modelling : - software product lines - domain analysis - feature modelling - commonalities and variabilities - feature diagrams Software reuse : - definition of and needs for software reuse - reuse techniques and design for reuse - object-oriented techniques for reuse and maintainability - object-oriented application frameworks Software maintenance and evolution : - Best programming practices and coding standards - Code refactoring and reengineering - Bad code smells - Software and design patterns - Design principles and heuristics An industrial case study
Learning outcomes	**At the end of this learning unit, the student is able to :** Given the learning outcomes of the "Master in Computer Science and Engineering" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: • INFO1.1 , INFO1.3 • INFO2.5 • INFO5.5 Given the learning outcomes of the "Master [120] in Computer Science" program, this course contributes to the development, acquisition and evaluation of the following learning outcomes: 1 • SINF1.M3 • SINF2.5 • SINF5.5 Students completing successfully this course will be able to • Understand the difficulties of developing code in a change context as opposed to 'green field' development • Assess the impact of a change request to an existing product of medium size. • Describe techniques, coding idioms and other mechanisms for implementing designs that are more maintainable.

	• Understand how design patterns can improve the design of a software system. • Refactor an existing software implementation to improve some aspect of its design. • Identify the principal issues associated with software evolution and explain their impact on the software lifecycle.' • Discuss the advantages and disadvantages of different types of software reuse.
Evaluation methods	For this course, students will be evaluated by: • a certifying continuous assessment of the project, which is based on 3 different missions, carried out by pairs, to be delivered during the course semester; • a presentation by pairs, carried out during the exam session, where the missions completed during the semester are put into perspective with the concepts seen in the course; • an individual examination on the topics seen in the theory course , carried out during the exam session. To constitute the final grade, the weight given to the continuous assessment is: • 10% of the final grade for active participation during the practical sessions • 50% for the 3 missions carried out in pairs during the course semester and the weight of the exam is: • 20% for the written exam (individual) • 20% for the presentation (by pairs) The grade obtained for the continuous assessment can be individualized according to the student's involvement within his group during the course semester (presence in activities, active participation in the missions and in work carried out and presented). The work giving rise to the continuous assessment grade cannot be redone in the second session; the continuous assessment grade acquired in the first session will be retained in the event of a second session. Only the individual written exam and the presentation in pairs putting the missions into perspective with the concepts seen in the course can be taken in the second session. As for the course istelf, all course assessments will be in English.
Teaching methods	• **Theory sessions** covering the different course topics • **Practical sessions** to apply the concepts in practice • **Missions** to experience the problems related to and solutions for developing and evolving a maintainable and reusable software system. • **Invited presentation by a company** to illustrate some of the course topics applied in industrial practice. • Optionally some sessions on a **theme related to research** in the area of software maintenance, reuse and evolution.
Content	The course will cover a variety of techniques, tools and methodologies to help building software systems that are easier to understand, maintain, reuse and evolve. Preliminaries: • Definitions and difference between software maintenance, software evolution and software reuse • Different types of software maintenance and evolution • Causes for software maintenance and change • Technical debt • Laws of software evolution Domain modelling: • Domain modelling and domain analysis • Software product lines • Feature-oriented domain analysis • Feature modelling, commonalities and variabilities • Feature relationships, dependencies and cross-tree constraints • Semantics of feature models and feature model anomalies Software reuse: • Definitions of reusability, software reuse and reusable components • How object-oriented programming promotes modularity, maintainability and reuse • Encapsulation, information hiding, polymorphism and code sharing • Key object-oriented concepts: object, classes, methods, messages, inheritance • Polymorphism and dynamic binding • Method overriding, self and super calls • Abstract classes and methods • Different kinds of inheritance: single, multiple, interfaces, mixins Bad code smells: • Bad smells • Bad smells vs. refactorings • Bad smell categories and examples • Coupling and cohesion

	Code refactoring: • Refactoring (definitions, motivations, when should you refactor) • Refactoring categories and examples • Refactoring vs. code quality • Merge conflicts due to refactoring Software patterns: • Christopher Alexander's building architectural patterns • (Software) design patterns (definitions, motivations, structure) • Abstract Factory design pattern • Factory Method design pattern • Strategy and Decorator design patterns • Antipattern (definition, purpose, example: The Blob) • The 7 deadly sins Design heuristics: • Design heuristics (definition, purpose, examples) • Design heuristics related to inheritance and polymorphism • Design heuristics related to cohesion • Design heuristics related to coupling Application frameworks: • Object-oriented application frameworks (definition, purpose, examples) • How frameworks can achieve software reuse • Inversion of Control (the “Hollywood” principle) • Software frameworks vs. libraries • Hotspots and hook methods • Commonality and variability • White vs. grey vs. black box frameworks • Template method design pattern • Design patterns vs. frameworks • Refactoring to a framework • Using template methods to evolve an application into a framework • Refactoring to specialise or generalise class hierarchies Industrial case study (invited speech by a selected company) Additional research-related sessions (if time remains) on selected topics such as: • Context-Oriented Software • Reflection and metaprogramming
Inline resources	Moodle course website The course slides as well as other relevant and practical information related to the course will be accessible on Moodle. The same platform will also be the means of communication between the teacher(s) and the students.
Bibliography	**French** Compte tenu de la variété des sujets abordés, ce cours ne suivra pas un seul livre de référence, mais sera basé sur du matériel provenant de nombreuses sources différentes. Les slides de cours seront le matériel de référence principale pour ce cours et des pointeurs vers des lectures supplémentaires seront fournis par la plate-forme de cours en ligne. **English** Given the variety of topics covered, this course will not follow a single textbook but is based on material from many different sources. As such, the course slides will be the main reference material for this course and pointers to additional reading material will be provided through the online course platform.
Other infos	Even though good quality software may be easier to maintain and evolve, software quality assurance techniques will not be addressed explicitly in this course as they are the topic of a separate course on Software Quality Assurance [LINFO2251] Expected background: • Having a good knowledge of and experience with object-oriented programming concepts, algorithms and data structures. • Having prior or simultaneous experience with the development of a medium- to large-scale software system.
Faculty or entity in charge	INFO

Programmes containing this learning unit (UE)

Program title	Acronym	Credits	Prerequisite	Learning outcomes
Master [120] in Computer Science and Engineering	INFO2M	5		🔍
Master [120] in Computer Science	SINF2M	5		🔍