UCLouvain

lingi2252

2020

Software Maintenance and Evolution

Due to the COVID-19 crisis, the information below is subject to change, in particular that concerning the teaching mode (presential, distance or in a comodal or hybrid format).

5 credits	30.0 h + 15.0 h	Q1

Teacher(s)	Mens Kim ;
Language :	English
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: • Software development in the context of an existing code base as opposed to 'green field' development o Software comprehension and concern location o Change impact analysis o Reverse engineering • Software Maintenance o Best programming practices o Coding standards o Design principles and heuristics o Design patterns o Refactoring o Reengineering • Software Reuse and Evolution o The laws of software evolution o Reuse techniques and design for reuse o Libraries vs. application frameworks • Software product lines
Aims	1 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: • 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. - - - - *The contribution of this Teaching Unit to the development and command of the skills and learning outcomes of the programme(s) can be accessed at the end of this sheet, in the section entitled "Programmes/courses offering this Teaching Unit".*
Evaluation methods	**Due to the COVID-19 crisis, the information in this section is particularly likely to change.** COURSE EVALUATION : • [10%] Active participation during practical sessions • [40%] 2 intermediate missions linked to the practical sessions • [10%] demo of developed application at end of semester • [50%] during exam session • [25%] written exam • [25%] presentation of a final mission In case of doubt about the final grade, the teacher reserves the right to ask a student to pass a complementary oral exam.
Teaching methods	**Due to the COVID-19 crisis, the information in this section is particularly likely to change.** COURSE ORGANISATION: **Theory sessions** covering the different course topics **Practical sessions** to apply the concepts in practice • *developing and evolving a maintainable and reusable software system* **Missions** to complete an application developed during the practical sessions
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 • The principle of 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 speaker) Context-Oriented Programming • Traditional vs. context-oriented software • Design heuristics (definition, purpose, examples) • Context-oriented programming for dynamic software adaptation • Implementing techniques for dynamic adaptation of software behaviour to context • Method dispatch and method pre-dispatch • Case studies of context-oriented programming systems Additional sessions (if time remains) • Reflection and metaprogramming • Aspect-oriented programming
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 [LINGI2251] 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	Aims
Master [120] in Computer Science and Engineering	INFO2M	5		
Master [120] in Computer Science	SINF2M	5		