Learning goals for the course CS-C3180 Software design and modelling

Topic	Core concepts	Basic knowledge	Good knowledge	Very good knowledge
Software Design and Modelling	Software design activities, models and methods during design activities.	Can explain the essence of software design activities. Can explain the purpose of modelling. Can explain, how the activities and models relate to problem solving.	Can apply systematic methods to create models that are relevant and understandable by their intended audience. Can analyse strengths and weaknesses of the created models.	Can evaluate the benefits and limitations of modelling methods. Can analyse how the different models support each other. Can relate the methods and models with agile software development.
Requirements engineering (RE)	Cultural change in software engineering. Functional and quality requirements, Use cases, User stories, Quper method	Can explain the essence of RE. Can explain the meaning of the core concepts.	Can apply use case and user story methods for modelling functional requirements. Can apply the Quper method for defining quality requirements. Can assess RE models critically.	Can analyse how RE relates to domain modelling, software architecture design and testing. Can assess RE methods critically.
Domain modelling (DM)	Purpose and properties of models, modelling, Domain Model, glossary	Can explain the essence of DM. Can explain the meaning of the core concepts.	Can create models that represent the key domain knowledge using UML notation. Can assess domain models critically.	Can analyse the contribution of domain models to requirements engineering, testing and software architecture design. Can evaluate the benefits and limitations of domain modeling methods.
Testing	Systematic approach; Test strategy; Coverage and knowledge in testing; Test design techniques: operational variables modelling, combination testing, scenario based testing	Can describe the meaning of a systematic approach and coverage in testing. Can explain the meaning of core concepts.	Can define test strategy for a simple software system. Can systematically design tests using design documentation (RE, DM) and basic software test design techniques. Can assess test coverage and explain the meaning of used coverage criteria.	Can apply domain knowledge to guide and reason the test design decisions. Can critically assess and improve design models (RE, DM, SAD) from the viewpoint of testing. Can analyse the contribution and value of software testing in relation to other development activities.
Software arcitecture design (SAD)	Software architecture (SA). Architecturally significant requirement (ASR). Software architecture design (SAD) and modelling.	Can explain the meaning of the core concepts.	Can design and model a functional architecture that tries to address the ASRs. Can define an ASR and justify why it is architecturally significant.	Can design and model a functional architecture in a consistent manner. Can focus on the architecturally significant parts of the functional architecture. Can analyze the purpose of functional architecture design.
Teamwork	Team communication, Good teamwork practices	Can explain good teamwork practices for software design and modelling. Can compare the differences between working as a team and individual work.	Can analyse own work practices and contribution as a team member.	Can analyse and select appropriate teamwork practices for software design and modelling. Can apply good practices for successful teamwork.

Table version: 21.8.2017