Course summary

Jingyue Li (Bill)

Exam

Vurderingsordning: Mappeevaluering

Termin	Statuskode	Vurdering	Vekting	HjelpemidlerDato	Tid	Eksamens-system
Vår	ORD	Arbeider	50/100			INSPERA
Vår	ORD	Hjemme-eksamen (1)	50/100	Utlevering 03.06.2022	09:00	INSPERA
				Innlevering 03.06.2022	13:00	
Sommer	UTS	Arbeider	50/100			INSPERA
Sommer	UTS	Hjemme-eksamen	50/100			INSPERA

A: 89–100 points
B: 77–88 points
C: 65–76 points
D: 53–64 points
E: 41–52 points
F: 0–40 points

Letter grade A-F

Evaluation and grading

- Exercises: 50%
- Exam: 50%
- You have to pass both!
  - You have to hand in **all** the exercises **and** get a passing grade to be allowed to take the exam
  - If you fail the exam, you will fail the course

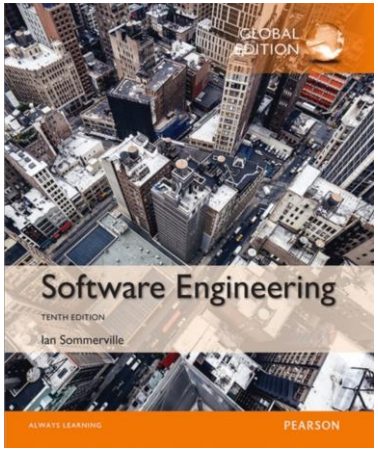
Exercises	Exam	Final grade
Fail	Fail	Fail
Fail	Pass	Fail
Pass	Fail	Fail
Pass	Pass	Combine exercise and exam grades

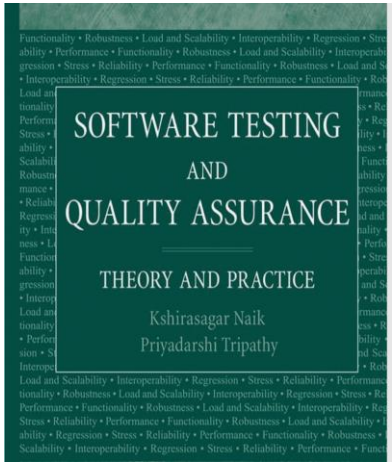
Couse modules

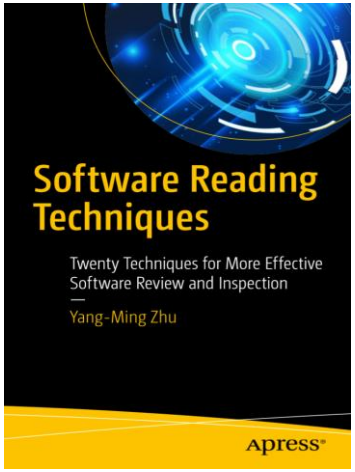
- Requirement module
- Testing module
- Code review and refactoring
- Chosen topics
  - DevOps
  - Technical debt
  - Cost and effort estimation
  - Defect estimation
  - Software ecosystem
  - Software startup
- Exercises 1 - 3

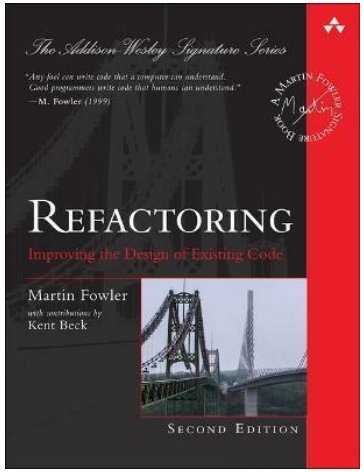
Curriculum

- Journal and conference articles (uploaded to blackboard)
- Some chapters from books (accessible in NTNU library)









(1st/2nd edition)

Requirement module

Description	Students should be able to	To read
Requirement elicitation	• Apply goal oriented RE approach	• Ian Sommerville (2016), Software Engineering (10th ed.), ISBN 978- 0133943030, chapter 4 • Paper 1: Goal-oriented RE A guided tour
Requirement quality	• Identify quality issues of requirements and fix them • Define different function and non-functional requirements	• Paper 1: IEEE Recommended Practice for Software Requirements Specifications Std 830-1998 • Paper 2: Glinz, M. (2007). On Non-Functional Requirements. 15th IEEE International Requirements Engineering Conference (RE 2007), 21–26. • Paper 3: Ontology-Driven Guidance for Requirements Elicitation • Paper 4: Bidirectional requirement traceability

Testing module - 1

Description	Students should be able to	To read
Control flow and data flow testing	• Explain code, decision, path, and output coverages • Create test cases by using different data flow testing strategies • Explain how to use the test coverage information for different purposes	Software testing and quality assurance book, chapter 4 and 5
Domain and function testing	• Apply domain testing approach to generate test cases of single variable and multiple variables in combination • Explain risk-based testing	Software testing and quality assurance book, chapter 6 and 9

Testing module - 2

Description	Students should be able to	To read
Integration and system testing	• Explain different approaches for creating integration test cases and their pros and cons • Create different system test cases • Explain different categories of acceptance test cases • Explain different test prioritization approaches	Software testing and quality assurance book, chapter 7, 8, and 14
Regression testing	• Apply graph-walk safe regression test selection approach • Explain firewall regression test selection approach • Explain different regression test minimization and prioritization strategies	Paper 1: Regression testing minimization, selection, and prioritization: a survey

Testing module - 3

Description	Students should be able to	To read
Test in practice	• Explain concepts and strategies related to test planning and execution. • Create a test plan.	Software testing and quality assurance book, chapter 12 and 13

Code review and refactoring

Description	Students should be able to	To read
Code review and code refactoring	• Explain why code inspection and testing complement each other • Explain unsystematic vs. systematic reading techniques • Apply check-list based, defect-based, and perspective-based reading techniques • Explain the purpose and steps of code refactoring • Apply code refactoring methods to identify bad code smells in code (Python) and refactor them	Software reading techniques, chapter 3 and 4 Refactoring book: Improving the Design of Existing Code, chapter 3 (Bad smells in code) Paper 1: Modern code review a case study at google Paper 2: Expectations, Outcomes, and Challenges of Modern Code Review

Chosen topics - 1

Description	Students should be able to	To read
DevOps	• Explain the motivation for using DevOps • Explain the basic concepts of DevOps	• Slides only
Technical debt	• Explain the motivation and approaches to measure technical debt	• Slides only
Cost&Effort and defect estimation	• Explain importance and challenges in estimating cost and software defects • Describe different estimation techniques, i.e., expert judgement, estimation by analogy, Lines of code, and COCOMO • Describe machine learning techniques for software defect predictions	• Slides only

Chosen topics - 2

Description	Students should be able to	To read
Software ecosystem	• Explain definitions, key elements and examples of software ecosystems • Describe characteristics of software ecosystems: complexity, productivity, robustness, healthiness and coopetition	• Slides only
Software startup	• Explain definitions, key elements and examples of software startups • Describe concept of Technical Debt in Software startups • Explain logic of effectuation and causation • Describe concept and usages of Minimum Viable Product in Software startups	• Slides only

Structure of the exam

- Two case studies (10 points each, 20 points in total)
- Five open-ended questions (3 points each, 15 points in total)
- 15 close-ended (Multiple-choice) questions (1 point each, 15 points in total)

A case study example

- Company A is going to pay Company B for developing Autonomous Truck Platooning. Truck platooning (as shown in the above picture) is the linking of two or more trucks in convoy, using connectivity technology and automated driving support systems. These vehicles automatically maintain a set, close distance between each other when they are connected for certain parts of a journey, for instance, on motorways.

- The requirements Company B got from Company A are as follows.

We are going to develop Autonomous Truck Platooning. We want some numbers of trucks can drive autonomously, and other trucks are driven by humans. The autonomous trucks should drive at a default speed. We hope the truck platooning can drive along the route we set in GPS before the journey and follow the human drivers' command to change the route. We hope the autonomous trucks should have three chairs for the drivers. We want the truck to drive safely. So the trucks should have sensors to avoid collisions with obstacles because there are always other vehicles or obstacles in the motorway. When there are obstacles on the road, the trucks should stop. We also want the trucks to drive efficiently, which means that the trucks should minimize the distance between them and maximize the speed of the whole Platooning. The trucks should use sensors to detect the distances between trucks and the speed of the trucks.

A case study example (cont')

- Task: Identify quality issues in these requirements according to the requirements quality metrics: *ambiguity*, *inconsistency*, *forward referencing*, and *opacity*. If one requirement has several quality issues, list all of them. Then, try to fix the requirements quality issues of each requirement and write down the improved requirements.
- Example answers

- Ambiguity (1 point): So the trucks should have sensors to avoid collisions with obstacles because there are always other vehicles or obstacles in the motorway (Not explain obstacles).
  Your fix: Only need to make a short list of obstacles. It does not need to be a complete list.

- Inconsistency (1 point): E.g., We hope the truck platooning can drive along the route we set in GPS before the journey and follow the human drivers' command to change the route (Conflict on who can make the control decision).
  Your fix: Only need to eliminate the inconsistency.

An open-ended question example

Question: Explain what static backward slicing is and how to create backward slicing using data flow information.

Answers:

- The purpose of code slicing is to choose only a subset of code that is relevant to a particular variable at a certain point to minimize the code for more cost-efficient testing and debugging.

- Static backward slicing: A slice with respect to a variable *v* at a certain point *p* in the program is the set of statements that contributes to the value of the variable v at p.

- To establish static backward slicing, all Defs and All P-uses of a variable v before a certain point p are chosen.

Summary

- Use the slides as the table of content to read books and papers
- The content of the course, especially the chosen topics, varies each year. Not all questions from previous years are relevant to this year
- The case studies and open-ended questions in previous years' exams can be used for practice
- We will create a discussion channel in Blackboard for Q&A exam related questions