CCPS 406 Introduction to Software Engineering

Ilkka Kokkarinen

Chang School of Continuing Education
Ryerson University

Version of December 31, 2022

Course Information

Given at	Chang School of Continuing Education Ryerson University Toronto, Canada
Course code	*CCPS 406, Introduction to Software Engineering*
Instructor	Ilkka Kokkarinen, `ikokkari@ryerson.ca`
Objectives	Upon completion of the course, students will be able to: ● Develop skills that will enable them to construct high quality software that is reliable, and reasonably easy to understand and maintain. ● Understand the importance of sound engineering principles in the process of constructing software systems. ● Apply the theory of software development process in building a software project. ● Analyze system requirements using various analytical tools. ● Learn design strategies to enable efficient software construction. ● Apply testing strategies to ensure quality of software. ● Use CASE tools and software development environments. ● Practice the principles of software project management. ● Develop a team-based software system of reasonable size and complexity with a focus on managing project risks.
Evaluation	The evaluation of this course consists of two components, added together to give the total course grade: ● Project (60%, broken into smaller pieces along the way) ● Take-home final exam (40%)
Programming Project	Students will design, implement, test and release a reasonably sized programming project, from the vision document provided by the instructor who then acts as the customer of this project. Each project must be done in groups of three to four students formed during the first week All project code and documents must be maintained under Git version control and stored in GitHub. The project grade will be calculated from the list of required milestone reports and the final presentation. In the current term, this project will be a small interactive fiction (aka "text adventure") game, consisting of a simple game engine and enough gameworld data to demonstrate the behaviour of the engine. The setting, theme and storyline of this game are to be freely chosen by the group.

Material

The course material consists of the official textbook, augmented by various online resources that expand on the themes and topic of this course, as chosen and curated by your instructor.

SE10	Ian Sommerville: Software Engineering, 10th Edition. The official textbook of this course, along with its collections of lecture slides, lecture videos and supplemental material. We will only cover the first nine chapters of this book during this first course, leaving the topics of the remaining chapters for more specialized courses on software engineering. This textbook can be ordered from Amazon and other places where fine books are sold. The publisher-approved eText version is also available for either one semester or lifetime rental.
SM	Sourcemaking. An excellent collection of tutorials on software design patterns, antipatterns, UML diagrams and software refactoring.
TCC	The Codeless Code. Darkly humorous fables on software engineering for your right brain, illustrating principles of programming philosophy and its best practices. Written as an affectionate parody of zen koans, the reader is expected to connect the implied dots to lift the core message of these fables into the realm of modern software engineering and their own future career in it.
MB	Martin Fowler. Weblog of Martin Fowler's observations on agile software design and related topics.
CCB	Clean Coder Blog. Weblog of Robert Martin's observations on object-oriented design, programming and testing.
YouTube	A collection of good videos, as selected by the instructor, of various colourful characters working in this field providing diverse views, opinions and experiences with various topics of this course. (Even if some videos show code examples in a programming language that you are not familiar with, remember that that code is intended as an illustrative example, and benevolently follow along with your assumption of what everything means.)

List of Modules

Module 1: *Introduction*

YouTube	Ian Sommerville: "10 Questions to Introduce Software Engineering" John Ousterhout: "A Philosophy of Software Engineering" Sandi Metz: "Rules" Kevlin Henney: "Old Is The New New"
SE10	1: Introduction
TCC	Case 162. All according to plan Case 180. Past perfect Case 187. Exit interview Case 207. Parrot Case 229. The recommended approach
MF	Technical Debt Technical Debt Quadrant
CCB	What software craftsmanship is about The obligation of the programmer Where is the foreman?

Module 2: *Software Processes*

YouTube	Ian Sommerville: "Plan-based and Agile software processes" Stormwind Studios: "Agile vs. Waterfall: The 3 Most Impactful Diffefences" Randy Rice: "V-Model" UVA_CS_3240: "Plan-Driven Methodologies", "Agile Methodologies", "The Polar Chart"
SE10	2: Software Processes 3: Agile Development
TCC	Case 10. Pride Case 22. Safety Case 100. Ten thousand mistakes Case 159. Blocks Case 188. Tough love
MF	Waterfall Process Principles of XP Objects and Iteration
CCB	The churn The start-up trap

Module 3: *Requirements*

YouTube	UVA CS 3240: "Requirements Engineering", "Requirements Elicitation", "Requirements Specification", Karl Wiegers: "Karl's Requirement Videos Playlist"
SE10	4: Requirements
TCC	Case 2. Unknown unknowns Case 18. Necessary features Case 111. Labyrinth Case 231. Laziness
MF	Conversational stories Use cases and stories
CCB	Why is estimating so hard?
Other	Liz England: "The Door Problem" John Salvatier: "Reality has a surprising amount of detail"

Module 4: *System Modelling*

YouTube	Eugene O'Loughlin: "Problem solving techniques 11: Use cases" Karl Wiegers: "What is a use case?" Derek Banas: "UML 2.0 Introduction" Derek Banas: "UML 2.0 Activity diagrams"
SE10	5: System modeling
TCC	Case 70. System Case 137. A thousand words Case 154. A bridge to nowhere Case 168. Horizontal, vertical Case 189. The dense forest
MF	Domain Driven Design Software Development Attitude
CCB	Solid relevance Agile is not now, nor was it ever, Waterfall Why won't it...

Module 5: *Software Design*

YouTube	UVA CS 3240: "Functional Decomposition", "Object-Oriented Decomposition"
	LucidCharts: "UML class diagram tutorial"
	LucidCharts: "UML use case diagram tutorial"
SE10	6: Architectural design
TCC	Case 1. The small stuff
	Case 64. Three beggars
	Case 86. Consequences
	Case 212. Seasons
	Case 230. All together now
MF	When To Make a Type?
	Function Length
	Beck's Design Rules
CCB	Screaming architecture
	Make the magic go away
	The frenzied panic of rushing

Module 6: *Design and Implementation*

YouTube	Sandi Metz: "SOLID Object Oriented Design" Saasbook: "Patterns, Antipatterns and SOLID Class Architecture" Christopher Okhravi's Code Walks: "Single Responsibility Principle", "Liskov Substitution Principle", "Interface Segregation Principle", "Dependency Inversion" AmigosCode: "Dependency Injection"
SE10	7: Design and implementation
TCC	Case 46. Jinyu's Tack Case 104. Guardrails Case 175. Flat shark Case 214. Streams of consciousness
MF	Yagni Continuous Integration On Pair Programming
CCB	Future proof Code hoarders Classes vs. data structures Tools are not the answer

Module 7: *Design Patterns and Anti-Patterns*

YouTube	UVA CS 3240: "Design Patterns Introduction", "Creational Patterns", "Behavioural Patterns", "Structural Patterns" Derek Banas: "Design Patterns Video Tutorial" (you don't need to watch this entire playlist, just the introduction followed by the individual videos about the particular design patterns that you would like to learn more about) Sandi Metz: "Learn Code Smells and Level Up Your Game!" Andy Sterkowitz: "Five Programming Antipatterns for Beginners"
SM	Creational: Builder, Factory Method, Object Pool Structural: Decorator, Adapter, Facade Behavioural: Iterator, Observer Code smells
TCC	Case 15. Immutable Case 98. Anti matter Case 155. Don't help Case 156. The garden path Case 195. The magician's code
MF	Is Design Dead? Getter Eradicator Uniform Access Principle AntiPattern Workflows of Refactoring
CCB	Pattern pushers The single responsibility principle The open-closed principle Thorns around the gold The dark path Types and tests

Module 8: *Software Validation*

YouTube	Let's Build That App: "What is Unit Testing, Why We Use It, and Sample Test Cases" Kevlin Henney: "Structure and Interpretation of Test Cases" AmigosCode: "Software Testing Tutorial - Learn Unit Testing and Integration Testing" UVA_CS_3240: "Verification and Validation", "Testing Strategies and Types", "Continuous Integration"
SE10	8: Software testing
TCC	Case 103. The black enamel box Case 135. Ass-backwards compatibility Case 142. The blind leading the blind Case 182. Mousetrap Case 219. Nothing really matters
MF	Test Categories Unit Test Test Cancer Mocks Aren't Stubs Object Mother
CCB	TDD harms architecture When TDD doesn't work Giving up on TDD

Module 9: *Refactoring*

YouTube	Frederick Vandrabant: "The Law of Demeter: Tell, Don't Ask" Christopher Okhravi's Code Walks: Law of Demeter Conor Hoekstra: "Beautiful Python Refactoring" ThoughtWorks: "Continuous Delivery 101, Part 1", "Part 2"
TCC	Case 94. Conventions Case 118. Clay Case 178. Unity Case 210. Hygiene Case 213. The imperfect mirror
MF	Refactoring
Others	Things of Interest: It is probably time to stop recommending Clean Code DZone: The Genius of the Law of Demeter

Module 10: *Software Evolution*

YouTube	UVA CS 3240: "Software Maintenance" Christina Hill: "The 4 Types of Software Maintenance" Loop Conf: "Zen and the Art of Software Maintenance" Christopher Okhravi: "E-Type, P-Type, S-Type Systems"
SE10	9: Software Evolution
TCC	Case 35. Sparrow breaks free Case 78. Crashing the third gate Case 128. The prison of infinite pleasures Case 133. Dead language Case 134. Thin ice Case 228. The trembling giant
MF	Canary Release Frequency Reduces Difficulty
CCB	A little structure Stabilization phases Is Doctor Calvin in the room?
Other	Epigrams in Programming

Module 11: Project Presentations

During this week, student groups present their projects to the instructor/customer in front of their peers. Discussion of the successfully implemented features of these projects, as seen from the user's point of view.

Module 12: Project Postmortems

During this week, student groups report and present their learning experiences on software engineering. Discussion of ways that the engineering issues and topics discussed during this course came up during the group project, and what human, communications or other outside issues affected the productivity of the group during the project.

Module 13: Final Exam

The final exam will be done in person during the last session of the course, in a TMU classroom to be announced later in the semester. The exam will last three hours.