# CCPS 406 Introduction to Software Engineering

Ilkka Kokkarinen

Chang School of Continuing Education
Toronto Metropolitan University

Version of April 24, 2024

# Course Information

| | |
|---|---|
| **Given at** | Chang School of Continuing Education<br>Ryerson University<br>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:<br>• Develop skills that will enable them to construct high quality software that is reliable, and reasonably easy to understand and maintain.<br>• Understand the importance of sound engineering principles in the process of constructing software systems.<br>• Apply the theory of software development process in building a software project.<br>• Analyze system requirements using various analytical tools.<br>• Learn design strategies to enable efficient software construction.<br>• Apply testing strategies to ensure quality of software.<br>• Use CASE tools and software development environments.<br>• Practice the principles of software project management.<br>• 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:<br>• Project (**60%**, broken into smaller pieces along the way)<br>• In-person final exam (**40%**)<br>**To pass the course, the student must get at least half of the marks in the final exam, regardless of the project marks.** |
| **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.<br><br>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.<br><br>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 the core of this course and the exam material. This is 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.<br><br>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 colourful characters 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 follow along benevolently with your assumption of what everything means.) |
| **C2** | The site wiki.c2.com, the original Wiki about various software engineering topic. Delightfully nineties in its spirit, content and approach, with each Wiki page containing Talmudic debates between experts in the field. Start with Road Maps, or you could get lost in the Random Pages in spirit of TV Tropes and similar. |
| **SE** | Software Engineering Stack Exchange, to learn the lingo and thinking and culture of the field for students planning to embark in it. |

# List of Modules

## Module 1: *Introduction*

| | |
|---|---|
| **YouTube** | Ian Sommerville: "10 Questions to Introduce Software Engineering"<br>John Ousterhout: "A Philosophy of Software Engineering"<br>Sandi Metz: "Rules"<br>Kevlin Henney: "Old Is The New New"<br>Continuous Delivery: "Developer Jobs Aren't What You Think They Are" |
| **SE10** | 1: Introduction |
| **TCC** | Case 162. All according to plan<br>Case 180. Past perfect<br>Case 187. Exit interview<br>Case 207. Parrot<br>Case 229. The recommended approach |
| **MF** | Technical Debt<br>Technical Debt Quadrant |
| **CCB** | What software craftsmanship is about<br>The obligation of the programmer<br>Where is the foreman? |
| **Other** | Paul McMahon: "All code is technical debt"<br>Henrik Warne: "What Makes a Good Programmer?"<br>Brian Kihoon Lee: "How To Be a Good Codemate"<br>Jamie Brandon: "Reflections on a Decade of Coding" |

# Module 2: *Software Processes*

| | |
|---|---|
| **YouTube** | Ian Sommerville: "Plan-based and Agile software processes"<br>Stormwind Studios: "Agile vs. Waterfall: The 3 Most Impactful Diffefences"<br>Randy Rice: "V-Model"<br>UVA CS 3240: "Plan-Driven Methodologies", "Agile Methodologies", "The Polar Chart"<br>Erick Minick: "What is Continuous Integration?", "Continuous Delivery", "Continuous Deployment vs. Continuous Delivery"<br>Continuous Delivery: "Continuous Delivery Simply Explained" |
| **SE10** | 2: Software Processes<br>3: Agile Development |
| **TCC** | Case 10. Pride<br>Case 22. Safety<br>Case 100. Ten thousand mistakes<br>Case 159. Blocks<br>Case 188. Tough love |
| **MF** | Waterfall Process<br>Principles of XP<br>Objects and Iteration |
| **CCB** | The churn<br>The start-up trap |
| **Other** | Joel on Software: "The Joel Test: 12 Steps to Better Code"<br>Scott Berkun: "Why Software Sucks" (part of best articles)<br>Wikipedia: "The Mythical Man-Month" |

# Module 3: *Requirements*

| | |
|---|---|
| **YouTube** | UVA CS 3240: "Requirements Engineering", "Requirements Elicitation", "Requirements Specification"<br>Karl Wiegers: "Karl's Requirement Videos Playlist"<br>Continuous Delivery: "5 Common Mistakes in User Stories", "Non-Functional Requirements Are Stupid" |
| **SE10** | 4: Requirements |
| **TCC** | Case 2. Unknown unknowns<br>Case 18. Necessary features<br>Case 111. Labyrinth<br>Case 231. Laziness |
| **MF** | Conversational stories<br>Use cases and stories |
| **CCB** | Why is estimating so hard? |
| **Other** | Liz England: "The Door Problem"<br>John Salvatier: "Reality has a surprising amount of detail"<br>Shamus Young: "Crash Dot Com" (the parts 1–5 are relevant for this lecture, but the entire story is a fun retro tale for all you Gen Z kids) |

# Module 4: *System Modelling*

| | |
|---|---|
| **YouTube** | Eugene O'Loughlin: "Problem solving techniques 11: Use cases"<br>LucidCharts: "UML class diagram tutorial", "UML use case diagram tutorial"<br>Derek Banas: "UML 2.0 Introduction", "UML 2.0 Activity diagrams", "UML 2.0 Sequence Diagrams" |
| **SE10** | 5: System modeling |
| **TCC** | Case 70. System<br>Case 137. A thousand words<br>Case 154. A bridge to nowhere<br>Case 168. Horizontal, vertical<br>Case 189. The dense forest |
| **MF** | Domain Driven Design<br>Software Development Attitude |
| **CCB** | Solid relevance<br>Agile is not now, nor was it ever, Waterfall<br>Why won't it... |
| **Other** | Wikipedia: "Unified Modeling Language"<br>Joel on Software: “Don't Let Architecture Astronauts Scare You” |

# Module 5: *Software Design*

| | |
|---|---|
| **YouTube** | UVA CS 3240: “Software Architecture Introduction”, “Modularity”, "Functional Decomposition", "Object-Oriented Decomposition" |
| **SE10** | 6: Architectural design |
| **TCC** | Case 1. The small stuff<br>Case 64. Three beggars<br>Case 86. Consequences<br>Case 212. Seasons<br>Case 230. All together now |
| **MF** | When To Make a Type?<br>Function Length<br>Beck's Design Rules |
| **CCB** | Screaming architecture<br>Make the magic go away<br>The frenzied panic of rushing |
| **Other** | The Art of Unix Programming: "Basics of Unix Philosophy"<br>Wikipedia: "Worse is Better"<br>Programming Is Terrible: “Write code that is easy to delete, not easy to extend” |

# Module 6: *Design and Implementation*

| | |
|---|---|
| **YouTube** | Sandi Metz: "SOLID Object Oriented Design"<br>Saasbook: "Patterns, Antipatterns and SOLID Class Architecture"<br>Christopher Okhravi's Code Walks: "Single Responsibility Principle", "Liskov Substitution Principle", "Interface Segregation Principle", "Dependency Inversion"<br>AmigosCode: "Dependency Injection"<br>Frederick Vandrabant: "The Law of Demeter: Tell, Don't Ask"<br>Christopher Okhravi's Code Walks: "Law of Demeter" |
| **SE10** | 7: Design and implementation |
| **TCC** | Case 46. Jinyu's Tack<br>Case 104. Guardrails<br>Case 175. Flat shark<br>Case 214. Streams of consciousness |
| **MF** | Yagni<br>Continuous Integration<br>On Pair Programming |
| **CCB** | Future proof<br>Code hoarders<br>Classes vs. data structures<br>Tools are not the answer |
| **Other** | Wikipedia: "GRASP (Object-Oriented Design)", "Cohesion (computer science)", "Coupling (computer programming)", "Law of Demeter"<br>DZone: "The Genius of the Law of Demeter"<br>Donny Wals: "Loose Coupling and the Law of Demeter" |

# Module 7: *Design Patterns and Anti-Patterns*

| | |
|---|---|
| **YouTube** | UVA CS 3240: "Design Patterns Introduction", "Creational Patterns", "Behavioural Patterns", "Structural Patterns"<br>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) |
| **SM** | Design Patterns |
| **TCC** | Case 15. Immutable<br>Case 98. Anti matter<br>Case 155. Don't help<br>Case 156. The garden path<br>Case 195. The magician's code |
| **MF** | Is Design Dead?<br>Getter Eradicator<br>Uniform Access Principle |
| **CCB** | Pattern pushers<br>The single responsibility principle<br>The open-closed principle<br>Thorns around the gold<br>The dark path<br>Types and tests |
| **Other** | Henrik Warne: "Lessons Learned in Software Development" |

# Module 8: *Software Validation*

| | |
|---|---|
| **YouTube** | Let's Build That App: "What is Unit Testing, Why We Use It, and Sample Test Cases"<br>Kevlin Henney: "Structure and Interpretation of Test Cases"<br>AmigosCode: "Software Testing Tutorial - Learn Unit Testing and Integration Testing"<br>UVA CS 3240: "Verification and Validation", "Testing Strategies and Types", "Continuous Integration"<br>Erick Minick: "What is Continuous Testing?"<br>Continuous Delivery: "Unit Testing Is The Bare Minimum" |
| **SE10** | 8: Software testing |
| **TCC** | Case 103. The black enamel box<br>Case 135. Ass-backwards compatibility<br>Case 142. The blind leading the blind<br>Case 182. Mousetrap<br>Case 219. Nothing really matters |
| **MF** | Test Categories<br>Unit Test<br>Test Cancer<br>Mocks Aren't Stubs<br>Object Mother |
| **CCB** | TDD harms architecture<br>When TDD doesn't work<br>Giving up on TDD |
| **Other** | Joel on Software: "Top Five (Wrong) Reasons You Don't Have Testers" |

# Module 9: *Refactoring*

| | |
|---|---|
| **YouTube** | Conor Hoekstra: "Beautiful Python Refactoring"<br>Andy Sterkowitz: "Five Programming Antipatterns for Beginners"<br>Sandi Metz: "Learn Code Smells and Level Up Your Game!"<br>Continuous Delivery: "TDD Is The Best Design Technique" |
| **SM** | Code Smells<br>Refactoring Techniques |
| **TCC** | Case 94. Conventions<br>Case 118. Clay<br>Case 178. Unity<br>Case 210. Hygiene<br>Case 213. The imperfect mirror |
| **MF** | Refactoring<br>AntiPattern<br>Workflows of Refactoring |
| **Other** | Arlo Belshee: "The Core 6 Refactorings" |

# Module 10: *Software Evolution*

| | |
|---|---|
| **YouTube** | UVA CS 3240: "Software Maintenance"<br>Christina Hill: "The 4 Types of Software Maintenance"<br>Loop Conf: "Zen and the Art of Software Maintenance"<br>Christopher Okhravi: "E-Type, P-Type, S-Type Systems"<br>Continuous Delivery: "14 Step Continuous Delivery Checklist" |
| **SE10** | 9: Software Evolution |
| **TCC** | Case 35. Sparrow breaks free<br>Case 78. Crashing the third gate<br>Case 128. The prison of infinite pleasures<br>Case 133. Dead language<br>Case 134. Thin ice<br>Case 228. The trembling giant |
| **MF** | Canary Release<br>Frequency Reduces Difficulty |
| **CCB** | A little structure<br>Stabilization phases<br>Is Doctor Calvin in the room? |
| **Other** | Epigrams in Programming<br>Wikipedia: "Lehman's Laws of Software Evolution"<br>Wikipedia: "No Silver Bullet" |

## 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 take place in person during the last session of the course, in the same TMU classroom as the course lectures. The exam will last two hours, and is closed book without notes or aids. The exam will consist of twenty true/false questions followed by twenty multiple choice questions.

**As per requirement by TMU School of Computer Science, students must score at least half the marks of the final exam to pass the course, regardless of their other marks.**