COMP 3004 Final review – Fall 2019

- Software engineering
    - what is it? why is it necessary?

- Build models
    - what is a model? a virtual representation of what we're going to build
    - why build models? to get a better idea of how to build the real system, clarify details and requirements so that we build the right thing
    - traceability: ability to track requirements through the entire development process
    - why traceability: better for maintenance, tells us which parts of the system are related to each other, and what needs to change if something needs to change

- Software development life cycle activities/phases
    - requirements elicitation
    - analysis
    - high level system design
    - detailed object design
    - implementation
    - testing
    - deployment and maintenance
    - what are the work products of each activities?

- Models in requirements analysis:
    - functional model (use cases, requirements)
    - dynamic model (state machines, sequence diagrams, activity diagrams)
    - object model (class diagrams, data dictionaries)

- Requirements elicitation
    - we need to know what the client wants
    - figure out the functional and non-functional requirements (9 categories)
    - scenarios and use cases
    - use cases:
        - high-level vs detailed use cases
        - actors (end users and external systems), system boundaries
        - relationships between actors and use cases (initiate, participate)
        - relationships between use cases (include, extend, inherit)
        - UML use case diagrams, use case table descriptions

- Analysis
    - dynamic model
        - show the system behaviour from the user's point of view
        - UML sequence diagrams, UML state machine diagrams
    - object model
        - where do find the objects: the use cases
        - entity, boundary, control objects
        - attributes, operations
        - aggregation (shared, composition), inheritance
        - multiplicity, directionality
        - UML class diagrams, data dictionary

- High level system design
    - design goals (based on NFRs)
    - subsystem decomposition
        - coupling and cohesion
        - layers and partitions
        - services (UML component diagrams, ball-and-socket notation)
        - architecture styles (3-tier, MVC, 4-tier, repository, pipe and filter, client-server, peer-to-peer)
    - design patterns
        - categories of patterns (creational, behavioural, structural)
        - purpose and usage of patterns (the ones we saw in class and in the textbook)
    - system design strategies
        - components, runtime components, and nodes (UML deployment diagram)
        - hardware/software mapping
        - persistent data storage
        - access control (static vs dynamic)
        - global control flow (procedural, event-driven, threaded)
        - boundary use cases (startup, shutdown, configuration)

- Detailed object design
    - application vs solution domain
    - types of inheritance (specification, implementation)
    - delegation
    - Liskov substitution principle
    - OCL, contracts (invariants, preconditions, postconditions)

- Implementation
    - model transformations, optimizing the object model
    - mapping associations to collections in a programming language
        - one-to-one, one-to-many, many-to-many
        - qualified associations
        - association classes
    - mapping associations to storage schema (include aggregation and inheritance)
        - buried associations
        - association tables
        - vertical vs horizontal mapping of inheritance relationships to storage

- Testing
    - test cases, test components, test stubs, test drivers
    - blackbox vs whitebox techniques
    - unit testing
        - path testing, equivalence testing, boundary testing, state, polymorphism
    - integration testing
        - big bang, top down, bottom-up, sandwich, modified sandwich
    - system testing
        - functional testing, performance testing, acceptance testing
- Project management
    - classic vs agile
    - risk management

- Software development life cycle models/processes
  - sequential, iterative
  - waterfall, V-model, spiral, USDP
  - agile

- Configuration management
  - version control
  - change management
  - system building
  - release management

- Ethics
  - professionalism, software disasters
  - ACM software engineering code of ethics
  - process for making ethical decisions
    - identify the stakeholders, their risks and benefits, their rights
    - identify possible courses of action (at least 3)
    - classify them into: ethically acceptable, prohibited, obligatory
    - pick one! (not two) and justify

- Omitted material
  - OCL (part of 4.3)
  - project management (7.1)
  - configuration management (7.3)
  - USDP life cycle model (part of 7.2)

- Final exam
  - 3 hours, out of 100 marks
  - concepts: 42 marks (21 mcq, 2 marks each)
  - exercise: 50 marks (1 question, 6 parts)
  - ethics: 8 marks (1 question)

- BRING
  - campus card
  - many pencils, erasers

- ASSIGNED SEATING: on Monday, check Grades section for “Row” and “Seat” numbers

- QUESTIONS
  - write down your question, wait until I get to your row
  - please be reasonable: I can’t give you answers, I can’t explain the material
  - DO NOT ASK QUESTIONS OF THE TAS
    - they are NOT the TAs for this course, they don’t know the material
    - they answer anyway, giving wrong answers