Introduction to main course topics
seng 301

Main topics (in order)
Abstraction, Unit and acceptance testing, Version control, Issue management, Unified modeling language, Design, Design patterns, Analysis including use cases and domain models, and Development processes/methodologies.
(We will spend around 1 or 2 weeks on each main topic.)

Other topics
OOP, Python, Java, Django, Git, Subversion, JUnit, PyUnit, ...
Analysis and Design of Large-Scale Software I

Introduction to developing large-scale, quality software, from analysis of requirements, through design, implementation, and testing. Introduction to design for non-functional properties of software. Emphasis on individual skills.

(What difference does large make?)

Abstraction

Abstraction is about considering something independent of some of its details. The process of formulating general concepts by abstracting common properties of instances.

We will discuss: procedural abstraction, data abstraction and object-oriented programming
Unit and acceptance testing

Unit testing is for testing individual pieces of an application.

Acceptance testing is to ensure that the application meets the customer requirements.

(We will discuss testing in general as well.)

Version control

Version control is primarily to allow you to track files or directory trees over time. This is useful for an individual programmer, but it is essential for working in teams.
Issue management

Refers to the process of keeping track of issues with a system and all activities related to resolving that issue.

Generally projects have some tool support for this tracking.
(In some projects both “bugs” and feature requests are handled using an issue tracking system.)

the Unified modeling language (UML)

A visual language for object modeling (i.e., it is for diagramming abstract views of a system).

There are a number of kinds of UML diagrams. We will focus mainly on class diagrams, and somewhat on object diagrams and sequence diagrams.
Design

“Designing object-oriented software is hard, and designing reusable object-oriented software is even harder. You must find pertinent objects, factor them into classes at the right granularity, define class interfaces and inheritance hierarchies, and establish key relationships among them.” [GoF]

Design is about deciding where in a code base various functionality should be implemented.

(As a though exercise, you might want to ask yourself the question: why don’t we implement a large application as one large procedure or method?)

Design patterns

A pattern is a description of communicating objects and classes that are customized to solve a general design problem in a particular context.

Example design problem: how can I design my system so that when the value of my system’s font-size setting changes all interested objects are notified of the change?

(We ideally want a solution that does not introduce unnecessary/unwanted dependencies.)
Analysis

*Dictionary*: A systematic approach to problem solving. Complex problems are made simpler by separating them into more understandable elements.

**In software engineering...**

Analysis is about answering: *what should the system do?*

---

**Analysis/use cases**

Formally captures the functional requirements of a system. This provides a dynamic view of the system (i.e., it is about the *behavior*).
Analysis/domain models
A conceptual model of a system including the key concepts and the vocabulary of the system. Includes main entities (along with attributes of those entities) and relationships giving a structural view of the system.

Development processes and methodologies
Methodology: A systematic way of doing something
Process: A series of actions or steps taken to achieve a particular end.
(Ad hoc: for or concerned with one specific purpose.)

A process or methodology will be about how and when activities such as analysis, design, testing, ... (i.e., the main topics of this course) occur.
A methodologies tells us how to put all of these things together.