COMP1730/COMP6730
Programming for Scientists

Software Design.

Announcements

* Major Assignment Released (Materials on Wattle).
* Group Sign-up open on Wattle until Friday.
* Assignment due 9:00am Monday 24 May.
* Please follow the submission instructions.
* Please have a look at the marking rubric.

Overview

* Disclaimer
* Development Methodologies
* Design Principles
* Design Patterns
* Standards and Conventions

Disclaimer

* Software Development is a huge topic.
* You could take a degree in Software Engineering at ANU.
* This lecture isn't going to cover everything, or even most things.
* If you are interested, look at COMP2100 or COMP2120 for more information.

Why Good Design is Important

* Y2K problem
  - Design failure: software using two digits for years (e.g., 99 for 1999)
  - Estimated US$500 billion worldwide to fix
* Knight Capital Flash Crash
  - Poorly designed (and tested) code switched buying and selling in stock market
  - Company lost $440 million in about 30 minutes

From Idea to Implementation

1. Cool Idea - High Level Vision
2. Who? What? How? - People, things and interactions
3. Entities, Structures and Processes - Modeling
4. Types, Classes, Functions - Defining the code structure
5. Code - Implementation

Two Development Methodologies

* Waterfall Method
  - Development proceeds in strict sequence:
  - Requirements gathering - Design - Implementation - Verification - Maintenance
  - Introduced as a non-working model
  - Useful for comparison; still used
* Agile Methodologies
  - Many variants (e.g., eXtreme Programming, Scrum) with similar philosophy
  - Design as an iterative process
  - Adaptive planning, early delivery, continuous improvement

Design and Development

* User cases and user stories
  - Requirements captured as short stories about how certain features or services might be used
  - Useful for making sure stakeholders and developers agree
* Test/Behaviour Driven Development
  - Required functionality specified through unit tests and other forms of automated testing
  - Tools exist for systematically turning high-level behavioural specification into executable tests (e.g., Python behave!)

Design and Development

* Release early, release often
  - Get a minimum viable product working as early as possible.
  - Get the product into the hands of the users and then refine.
  - A project that is incomplete may still deliver something useful.
* Failing fast and continuous development
  - The earlier you can detect problems the better.
  - Continuous automated testing and user feedback will help detect problems quickly.

Some Design Principles

* Keep it Simple
  - Make everything as simple as possible, but no simpler
  - Keep breaking the problem down until it can be explained in a sentence or two.
* Separation of Concerns (Modularity)
  - Reasoning about multiple interactions is difficult
  - Organise your project so you can focus on solving one aspect at a time.
  - Having clearly defined interfaces helps.

More Design Principles

* Principle of least surprise
  - Design functions, etc. so that naming, behaviour, arguments, etc. are consistent
  - Stick with familiar conventions
* Don’t repeat yourself
  - Every piece of knowledge must have a single, unambiguous, authoritative representation.
* You ain’t gonna need it
  - Only implement things when you need them, since requirements may (will) change frequently.

Design Patterns

* A 'template' solution to a common problem
* Examples
  - Adaptor Pattern
  - Observer Pattern
  - Factory Pattern
  - Command Pattern
* Many more - see: en.wikipedia.org/wiki/Software_design_pattern

Standard and Conventions

* “The good thing about standards is that there are so many of them to choose from”— Grace Hopper
* Take home message: It doesn’t matter which standard you choose as long as you choose, and stick to, one.

Why Use Standards?

* Help improve readability and maintenance of software
* Allow the creation of tools to aid in development
* Eliminate the need to make decisions
* Lowers the barrier of entry
* We have already seen and been using a number of standards in the guise of good programming practice

Naming Conventions

* Python restricts names to letters, numbers and underscores, and must start with a letter or underscore - the rest is up to the programmer.
  - `RADIUS_OF_EARTH` - constants
  - `MyCoolClass` - class names
  - `sum_if_negative` - function and variable names
  - `_my_secret_business` - private members

Code Layout and Organisation

* Indentation - tabs or spaces and how many
* Maximum line length - used to be 80 characters - now more commonly 120.
* How much goes in a file - often a single class per file
* Wrap scripts in `if __name__ == '__main__'`

Documentation Conventions

* Use docstrings for functions and classes.
* Keep a consistent style - several different ways to specify parameters and return types, etc.
* Use comments where appropriate but keep them up-to-date and don’t overdo. Well written code needs surprisingly few comments.

Testing Conventions

* Use libraries such as `unittest` or `pytest`
* Have one test for each function or class being tested
* Name tests based on what functionality is being tested
* Do not mix test and production code

* Many Python coding conventions are described in PEP (Python Enhancement Proposal) 0008 — Style Guide for Python Code based on the insight that code is read more often that it is written https://www.python.org/dev/peps/pep-0008/
* Not everyone will like all the conventions used in any given project, but the benefit of consistency that standards bring to a project will outweigh the individual tastes of a single team member