— COMP(2041|9044) 20T2 —

Software Construction: Techniques and Tools

'UNIX Magic', by Gary Overacre

Convenor/Lecturer **Andrew Taylor** Course Admin **Jashank Jeremy**

COMP(2041|9044) Staff

Convenor, Lecturer Andrew Taylor

Admin Jashank Jeremy

Tutors Samuel O'Brien, Anson Lee, Matthew Turner, Sage Barreda-Pitcairn, Tom Nguyen, Nathan Ellis, Vivian Dang, Maxwell Goodbury, Luka Kerr, Dylan Brotherston, Asher Silvers, Coen Townson, Sabrina Yan, James Treloar, Sabine Lim, Esther Wong, Connor O'Shea, Bridget McCarthy

Course Goals

Overview: to expand your knowledge of programming.
First programming courses deals with ...

- one language (C or Python at CSE)
- some aspects of programming (e.g. basics, correctness)
- on small tightly-specified examples

COMP(2041|9044) deals with ...

- other languages (Shell, Perl)
- other aspects of programming (e.g. testing, performance)
- on larger (less-small) less-specified examples

Course Goals

Introduce you to:

- building software systems from components
- treating software as an object of experimental study

Develop skills in:

- using software development tools (e.g. git)
- building reliable, efficient, maintainable, portable software

Ultimately: get you to the point where you could build some software, put it on github, have people use it and have it rated well.

Inputs

At the start of this course you should be able to:

- produce a correct procedural program from a spec
- understand fundamental data structures + algorithms
  (char, int, float, array, struct, pointers, sorting, searching)
- appreciate the use of abstraction in computing

Outputs

At the end of this course you should be able to:

- understand the capabilities of many programming tools
- choose an appropriate tool to solve a given problem
- apply that tool to develop a software solution
- use appropriate tools to assist in the development
- show that your solution is reliable, efficient, portable

Syllabus Overview

1. Qualities of software systems
   - Correctness, clarity, reliability, efficiency, portability, ...
2. Techniques for software construction
   - Analysis, design, coding, testing, debugging, tuning
   - Interface design, documentation, configuration
3. Tools for software construction
   - Filters (*grep, cut, sort, uniq, tr, sed*...)
   - Scripting languages (*shell*, *Perl*, *Python*)
   - Intro to Programming for the web
   - Software tools (git, ...)

Lectures

- Tuesday, 14:00—16:00; Friday 10:00—12:00;
  delivered via Microsoft Teams Live Events
  - you will have email about how to access the event
  - feel free to ask questions via chat
  - lectures recorded and linked from course home page.
- present a brief overview of theory
- focus on practical demonstrations of coding
- demonstrate problem-solving (testing, debugging)
- Lecture slides available on the web before lecture.

Tutorials

- Tutorials start in week 1.
- Tutorials & labs online, via Blackboard Collaborate
  - you will have email about how to access Collaborate
- tutes clarify lecture material
- work through problems related to lecture topics
- give practice with design *(think before coding)*
- answers available on the web after the week's last tutorial.

To get the best out of tutorials

- attempt the problems yourself beforehand
- ask if you don't understand a question or how to solve it
- Do *not* keep quiet in tutorials ... talk, discuss, ...
- Your tutor may ask for your attempt to start a discussion.

Lab Classes

Each tutorial is followed by a two-hour lab class.

- Several exercises, mostly small implementation/analysis tasks
- Aim to build skills needed for assignments, exam
- Aim to give experience applying tools/techniques
- Done individually
- Submitted via `give`, before Tuesday 12:00
- Automarked (with partial marks) — 15% of final mark
- Labs may include challenge exercises:
  - may be silly, confusing, or impossibly difficult
  - full marks possible without completing any challenge exercises

Weekly Tests

From week 3, weekly tests:

- programming tests
- immediate reality-check on your progress.
- done in your own time under self-enforced exam conditions.
- Time limit of 1 hour
- Automarked (with partial marks) — 10% of final mark
- best 6 of 8 tests used to calculate the 10%
- any violation of test conditions ⇒ zero for whole component

Assignments

- Assignments give you experience applying tools/techniques to larger programming problems than lab exercises
- Assignments will be carried out individually.
- They *always* take longer than you expect.
- Don't leave them to the last minute.
- There are late penalties applied to maximum marks, typically 2%/hour — organising your time ⇒ no penalty

Code of Conduct

CSE offers an inclusive learning environment for all students.
In anything connected to UNSW, including social media,
these things are student misconduct and will not be tolerated:

- racist/sexist/offensive language or images
- sexually inappropriate behaviour
- bullying, harassing or aggressive behaviour
- invasion of privacy

Show respect to your fellow students and the course staff

Plagiarism

What is plagiarism?

Presenting the (thoughts or) work of another as your own.

Cheating of any kind constitutes academic misconduct and carries a range of penalties. Please read course intro for details.

Examples of inappropriate conduct:

- groupwork on individual assignments (discussion OK)
- allowing another student to copy your work
- getting your hacker cousin to code for you
- purchasing a solution to the assignment

Remember

You are only cheating yourself and chances are you will get caught!

Plagiarism

- Labs, tests, assignments must be entirely your own work.
- You can not work on assignment as a pair (or group).
- Plagiarism will be checked for and *penalized*.
- Plagiarism may result in suspension from UNSW.
- Scholarship students may lose scholarship.
- International students may lose visa.
- Supplying your work to any another person may result in loss of all your marks for the lab/assignment.

Final Exam

- online practical exam, during exam period; you complete from home
- closed-book — limited on-line language documentation available
- some multiple-choice/short-answer questions, similar to tut questions.
- some questions will ask you to read shell, Perl, regex, ...
- six (probably) implementation questions, similar to lab exercises
- most marks for questions which ask you to write shell or Perl
- also may ask you to answer written questions
- you *must* score 18+/45 on the final exam to pass course

Assessment

- 15% Labs
- 10% Weekly Programming Tests
- 15% Assignment 1 — due week 7
- 15% Assignment 2 — due week 10
- 45% Final Exam

Above marks may be scaled to ensure an appropriate distribution
To pass you must:

- **score 50/100 overall**
- **score 18/45 on final exam**

For example:
55/100 overall and 17/45 on final exam $\Rightarrow$ **55 UF** not 55 PS

How to Pass this Course

- coding is a *skill* that improves with practice
- the more you practise, the easier you will find assignments/exams
- do the lab exercises
- do the assignments *yourself*
- practise programming outside classes
- treat extra tutorial questions like a mini prac exam

Reading Material

General References:

- Kernighan & Pike, 'The Practice of Programming',
  Addison-Wesley, 1998
  (Inspiration for 2041, in philosophy and tool details.)
- McConnell, 'Code Complete' (2/e),
  Microsoft Press, 2004
  (Many interesting case studies and practical ideas.)

Reading Material

Perl References:

- Christiansen, foy, Wall, Orwant, 'Programming Perl' (4/e),
  O'Reilly, 2012 (original and best Perl reference manual.)
- Schwartz, foy, Phoenix, 'Learning Perl' (7/e),
  O'Reilly, 2016 (gentle, careful introduction to Perl.)
- Christiansen & Torkington, 'Perl Cookbook' (2/e),
  O'Reilly, 2009 (lots and lots of interesting Perl examples.)
- Schwartz, foy, Phoenix, 'Intermediate Perl' (2/e),
  O'Reilly, 2012 (great to read after 2041: picks up where we finish.)
- Sebesta, 'A Little Book on Perl',
  Prentice-Hall, 1999 (very concise introduction to Perl.)
- Orwant, Hietaniemi, Macdonald,
  'Mastering Algorithms with Perl',
  O'Reilly, 2011 (algorithms and data structures via Perl.)

Reading Material

Shell Programming References:

- Kochgan & Wood, 'Unix Shell Programming',
  Sams Publishing, 2003 (careful introduction to shell programming.)
- Albing, Vossen, 'bash Cookbook',
  O'Reilly, 2007 (example-based introduction to shell programming.)

Reading Material

Unix Tools References:

- Powers, Peek, O'Reilly, Loukides, 'Unix Power Tools' (3/e),
  O'Reilly, 2003 (comprehensive guide to common Unix tools.)
- Loukides & Oram, 'Programming with GNU Software',
  O'Reilly, 1996 (tutorial on GNU tools: gcc, gdb, ...)
- Robbins, 'Unix in a Nutshell' (4/e),
  O'Reilly, 2006 (concise guide to Unix and its toolset)
- Kernighan & Pike, 'The UNIX Programming Environment',
  Prentice-Hall, 1984 (precursor to textbook; intro to Unix tools)

Reading Material

- All tools in the course have extensive on-line documentation.
- Links to this material are available in the course Web pages.
- You are expected to master these systems largely by reading the manuals.

However ...

- we will also give introductory lectures on them
- the lab exercises will give practice in using them

— note —

"The ability to read software manuals is an invaluable skill."

— jas, 1999

Home Computing

- All tools in this course are available on Unix, Linux systems
- Many have been ported to MS Windows.
  (generally via the Cygwin project)
- All should be available on Mac.
  (given that Mac OS X is based on FreeBSD)
- Links to downloads will be placed on course Web site.

Home Computing

- There are subtle, minor incompatibilities and quirks between different implementations and versions of tools — therefore . . . *test your assignments at CSE* before you submit. ('But it works on my machine!' isn't a valid excuse.)
- *Note:* we expect any software *you* produce will be portable to all platforms — this is accomplished by adhering to *standards*.

Conclusion

The goal is for you to become a better programmer

- more confident in your own ability
- producing a better end-product
- ultimately, enjoying the programming process