COMP[29]041 - Software Construction 16s2

Lecturer/Admin: Andrew Taylor, andrewt@cse.unsw.edu.au

http://www.cse.unsw.edu.au/~cs2041/

Course Goals

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

- some aspects of programming (e.g. basics, correctness)
- on relatively small tightly-specified examples

COMP[29]041 deals with ...

- other aspects of programming (e.g. testing, performance)
- using a wider range of tools (e.g. filters, Perl)
- 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. make, gprof, svn)
- building reliable, efficient, maintainable, portable software

Ultimately: get you to the point where you could build some software, put it on sourceforge/google code/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, sed, cut, sort, uniq, tr,...*)
   - Scripting languages (*shell*, *Perl*+CGI, optionally Python)
   - Analysis/configuration/documentation tools (make, git, doxygen, gprof, ...)

Lectures

Lectures will:

- present a brief overview of theory
- give practical demonstrations of tools
- demonstrate problem-solving methods

Lecture notes available on the Web before each lecture.
Notes are available in 4-up PDF or HTML formats.
Lecture slides are a summary of notes + exercises.
Feel free to ask questions, but otherwise *Keep Quiet*.

Tutorials

Tutorials aim to:

- clarify any problems with lecture material
- work through problems related to lecture topics
- give practice with design skills *(think before coding)*

Tutorials start in week 2.
Tutorial questions available on the web the week before.
Tutorial answers available on the web after the week's last tutorial.
Use tutorials to discuss *how* solutions were reached.

Tutorials

To get the best out of tutorials

- attempt the problems yourself, before reading answers
- if you understand OK and get feasible answer, fine

Ask your tutor if ...

- if you aren't sure your answer is feasible/correct
- if you don't know *how* the solution was reached
- 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.
Lab exercises aim to build skills that will help you to

- complete the assignment work
- pass the final exam

Lab classes give you experience applying tools/techniques.
Each lab exercise is a small implementation/analysis task.
Often includes a challenge exercise needed to get an A (may be Python!)
Do them yourself! and *Don't fall behind!*

Lab Classes

Lab exercises contribute 10% to overall mark.
In order to get a mark, lab exercise for Week X must be

- submitted via `give` before Monday at end of week X
- demonstrated to the tutor *during* the Week X lab
  OR, demonstrated *at the start of* the Week X+1 lab

There are more than 10 marks available for labs.

Lab Exemption

COMP9041 students can apply for lab exemption if they have:

- full-time work or other commitments
- previous programming, particularly scripting, experience
- good marks (70+) in previous courses

If grranted - final mark calculated without lab component.
Strongly recommended theystill complete labs.

Assignments

Assignments give you experience applying tools/techniques
(but to larger programming problems than the 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 assignment marks, typically

- 5%/day for first 2 days
- 20%/day after that

Organising your time

$$\Rightarrow$$

no penalty.

Plagiarism

Labs and Assignments must be entirely your own work.
No group work in this course.
Plagiarism = submitting someone else's work as your own.
Plagiarism will be checked for and *penalized*.
Plagiarism may result in suspension from UNSW .
International students may lose their visa.
Supplying your work to any another person may result in loss of all your marks for the lab/assignment.

Final Exam

3-hour closed-book exam during the exam period.
2 written parts, 1 implementation part
The exam contributes 60% to total mark for the course.
Requirements on exams to pass course:

- must score at least 50% (30/60) on exam
- must solve two implementation tasks satisfactorily

If you fail to meet either hurdle, you receive UF grade.

Final Exam

Held in the CSE Labs (must know lab environment)
Format:

- we supply printed and on-line documentation
- we give you small programming exercises
- you solve them using an appropriate language
- prac tasks will be similar in style/difficulty to labs

Final Exam

Multiple tests applied to each solution.
A solution that fails 1 or 2 test cases may be regarded as partly solved, and will receive part marks.
May be requirements about which languages can be used.
Must solve a minimum number of the programming tasks.

Final Exam

Written sections:

- some multiple-choice/short answer questions
- some descriptive/analytical questions

Questions will be similar in style to tutorial questions.

Final Exam

How to succeed on the Implementation Exam:

- do the Lab exercises *yourself*
- do the Assignments *yourself*
- practise programming outside classes
- treat each Lab class like a mini Prac Exam

How to succeed the Written Exam:

- understand the Tutorial questions
- read the lecture notes
- prepare as for other written exams

Supplementary Exams

Supplementary exams are only available to students who

- do *not* attend the exam
- have a serious documented reason for not attending
- score $\geq$ 40% for labs and assignments

If you attend an exam

- you are making a statement that you are "fit and healthy enough"
- it is your only chance to pass (i.e. no second chances)

Assessment Summary

```
ass     = mark for assignments      (out of 30)
labs    = mark for assessed labs    (out of 10)
exam    = mark for exam             (out of 60)

okExam = (two implementation tasks solved)

mark = ass + labs + pexam + texam
grade     = HD|DN|CR|PS  if mark >= 50 && okExam
          = FL           if mark < 50 && okExam
          = UF           if !okExam
```

How to Pass this Course

Software construction is a *skill* that improves with practice.
The more you practise, the easier you will find assignments/exams.
Don't restrict practice to lab times and two days before assignments due.
It also helps to pay attention in lectures and tutorials.

Reading Material

General References:

- *Kernighan & Pike*, The Practice of Programming, Addison-Wesley, 1998.

  (Inspiration for 2041 - philosophy and some tool details)
- *McConnell*, Code Complete (2ed), Microsoft Press, 2004.

  (Many interesting case studies and practical ideas)

Reading Material

Perl Reference Books:

- *Wall, Christiansen & Orwant* , Programming Perl (3ed), O'Reilly, 2000. (Original & best Perl reference manual)
- *Schwartz, Phoenix & Foy*, Learning Perl (5ed), O'Reilly, 2008. (gentle & careful introduction to Perl)
- *Christiansen & Torkington*, Perl Cookbook (2ed), O'Reilly, 2003. (Lots and lots of interesting Perl examples)
- *Schwartz & Phoenix*, Learning Perl Objects, References, and Modules (2ed), O'Reilly, 2003. (gentle & careful introduction to parts of Perl mostly not covered in this course)
- *Schwartz, Phoenix & Foy*, Intermediate Perl (2ed), O'Reilly, 2008. (good book to read after 2041 - starts where this course finishes)
- *Sebesta*, A Little Book on Perl, Prentice Hall, 1999. (Modern, concise introduction to Perl)
- *Orwant, Hietaniemi, MacDonald*, Mastering Algorithms with Perl, O'Reilly, 1999. (Algorithms and data structures via Perl)

Reading Material

Shell Programming:

- *Kochgan & Wood 2003*, UnixÂ® Shell Programming, Sams Publishing 2003 (Careful intoduction to Shell Programming)
- *Peek, O'Reilly, Loukides*, Bash Cookbook, O'Reilly, 2007. (Recipe(example) based intro to Shell programming)

Reading Material

Unix Tools Reference Books:

- *Powers, Peek, O'Reilly, Loukides*, Unix Power Tools (3ed), O'Reilly, 2003. (Comprehensive guide to common Unix tools)
- *Loukides & Oram*, Programming with GNU Software, O'Reilly, 1997. (Tutorial on the GNU programming tools (gcc,gdb,...))
- *Robbins*, Unix in a Nutshell (4ed), O'Reilly, 2006. (Concise guide to Unix and its toolset)
- *Kernighan & Pike*, The Unix Programming Environment, Prentice Hall, 1984. (Pre-cursor to the 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 of the tools in the course are available under Unix and Linux.
Most have also been ported to MS Windows.
(generally via the CygWin project)
All tools should be available on Mac.
(given that Mac OS X is based on FreeBSD Unix)
Links to downloads will be placed on course Web site.

Home Computing

There may be minor incompatibilities between Unix, Windows and Mac versions of tools.

Therefore ... *test your assignments at CSE* before you submit them.

Note: we expect that any software that *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