CSSE2002/7023

Programming in the Large / Advanced Software Engineering

Week 1.1: Course Overview

Programming in the small

For those of you coming straight from CSSE1001, code you have written up to now has likely been:

1. small
2. written solely by you
3. relatively easy to debug
4. for a limited purpose
5. to exist for a limited time

Programming in the small

For those of you coming straight from CSSE1001, code you have written up to now has likely been:

1. small
2. written solely by you
3. relatively easy to debug
4. for a limited purpose
5. to exist for a limited time

... but large sofware projects are often none of these.

CSSE2002 — Programming in the large

CSSE2002 is about learning some practices that will make it easier for your code to scale up in size without becoming a broken unmaintainable mess. (While getting some practice with Java).

CSSE2002 — Course division

There are two people running this course:

- Dr Jessica Korte (Course coordinator and lecturer)
- Dr Scott Heath (Lecturer)

Scott will take the lectures for Weeks 1-8.
Jessica will be taking the lectures for Weeks 9-12/13?

Coming soon:

- Piazza discussion board
- Course-wide inbox

CSSE2002 — Course coordinator and lecturers

Dr Jessica Korte (j.korte@uq.edu.au)

- Teaching and Research postdoc
- Research area: collaborative design with young Deaf children
- Teaching at university level since 2010
- First tutoring job: Java programming!

CSSE2002 — Course coordinator and lecturers

Dr Scott Heath (scott.heath@uq.edu.au)

- Research focused postdoc
- I work in a robotics group (The Social Robotics Group)
- Programming is a big part of the development of robots

CSSE2002 — Other contributions

Lectures, tutorials, and practicals:

- Organised for us by Joel Fenwick.
- Derived from materials developed by Phil Stocks, Larissa Meinicke, and Graeme Smith.
- Modified by us.

Assignments:

- Written by Joel Fenwick.
- Modified by us.

Ethicality in Programming

- Programming is powerful, and becoming more powerful as technology becomes more ubiquitous.
- The way you design and code software impacts on people's lives:
    - Assumptions written into software:
        - "Dr" = male
    - Unethical client requests influences lives:
        - Fake quiz which advertises drugs to young girls - with side effects of depression & suicide
        - Data mining to look for patterns which would allow health insurers to deny health care (in USA!)
    - Ethical decision making has to be coded in:
        - How does your self-driving car behave when a pedestrian steps in front of it?
        - How does your military drone evaluate targets?

Ethicality in CSSE2002

You won't have to make decisions like that in this course, but you should be aware of them.

We want you to re-frame your thinking about university courses:

The purpose of a university course is that by the end of the course, you (as an individual) have acquired certain skills and knowledge.

Assessment is used to measure skills and knowledge as accurately as possible, but it is not perfect.

We encourage you to learn as much as possible from this course, in addition to the skills and knowledge that are directly assessed.

While assessment counts towards a grade for this course, additional skills and knowledge will count towards an entire career.

Cheating and collusion

All code which you submit for assessment must be, either

- supplied by teaching staff **for this run** of the course.
  OR
- written entirely individually by you.

All code you submit will be checked for collusion and plagiarism (including but not limited to using MOSS). If similarity is found, a school investigation may be started (these are not fun).
Seek help from course staff in plenty of time if you get into trouble.
Be very careful when seeking outside help — outside tutors may facilitate cheating (and get you caught) rather than your learning.

Misconduct — Things not to do

1. Do not show your code to other students
2. Do not look at other student's code
3. Do not use code given to you in other semesters or courses
4. Do not **try** to get other people to write code for you
5. Do not store your code in any online repository which **could** be accessible to others. (This will be deemed as if you have broken Item 1).
6. Do not start a few days before the deadline and then claim that cheating was the only way to get it done in time.

What's wrong with focussing only on assessment?

What's wrong with focussing only on assessment?

Taking assessment seriously is good. That is different from a primary focus on
"not failing".
How you think is your business but here are some "counterproductive" lines of thought which seem to often follow from such a focus:

What's wrong with focussing only on assessment?

Taking assessment seriously is good. That is different from a primary focus on
"not failing".
How you think is your business but here are some "counterproductive" lines of thought which seem to often follow from such a focus:

- Ignoring feedback
  - don't focus solely on summative aspects to argue them.
  - don't wait 'till semester end to examine

What's wrong with focussing only on assessment?

Taking assessment seriously is good. That is different from a primary focus on
"not failing".
How you think is your business but here are some "counterproductive" lines of thought which seem to often follow from such a focus:

- Ignoring feedback
  - don't focus solely on summative aspects to argue them.
  - don't wait 'till semester end to examine
- Teaching staff are the enemy
  - They set the assessment and deny students the marks they need. Right?
  - Why is that a problem?

What's wrong with focussing only on assessment?

Taking assessment seriously is good. That is different from a primary focus on
"not failing".
How you think is your business but here are some "counterproductive" lines of thought which seem to often follow from such a focus:

- Ignoring feedback
  - don't focus solely on summative aspects to argue them.
  - don't wait 'till semester end to examine
- Teaching staff are the enemy
  - They set the assessment and deny students the marks they need. Right?
  - Why is that a problem?
    It's very hard to ask someone for help if you think they are out to get you.

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF
  - Wanting worked answers for everything.

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF
  - Wanting worked answers for everything.
  - Needing copies of everything regardless of context.

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF
  - Wanting worked answers for everything.
  - Needing copies of everything regardless of context.
- Exams

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF
  - Wanting worked answers for everything.
  - Needing copies of everything regardless of context.
- Exams
  - Trying to learn exam answers by rote.

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF
  - Wanting worked answers for everything.
  - Needing copies of everything regardless of context.
- Exams
  - Trying to learn exam answers by rote.
  - Trying to learn mechanical techniques for solving exam questions.

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF
  - Wanting worked answers for everything.
  - Needing copies of everything regardless of context.
- Exams
  - Trying to learn exam answers by rote.
  - Trying to learn mechanical techniques for solving exam questions.
- Cannot do anything where there is the slightest risk of being wrong.

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF
  - Wanting worked answers for everything.
  - Needing copies of everything regardless of context.
- Exams
  - Trying to learn exam answers by rote.
  - Trying to learn mechanical techniques for solving exam questions.
- Cannot do anything where there is the slightest risk of being wrong.
  - Being unwilling to experiment with code.

- “If we don’t get marks for it, we won’t do it.” (tutes, practice exercises, …)
- “The protective wall of downloads” — JF
  - Wanting worked answers for everything.
  - Needing copies of everything regardless of context.
- Exams
  - Trying to learn exam answers by rote.
  - Trying to learn mechanical techniques for solving exam questions.
- Cannot do anything where there is the slightest risk of being wrong.
  - Being unwilling to experiment with code.
  - “Just give me the answer.”

- "If we don't get marks for it, we won't do it." (tutes, practice exercises, …)
- "The protective wall of downloads" — JF
  - Wanting worked answers for everything.
  - Needing copies of everything regardless of context.
- Exams
  - Trying to learn exam answers by rote.
  - Trying to learn mechanical techniques for solving exam questions.
- Cannot do anything where there is the slightest risk of being wrong.
  - Being unwilling to experiment with code.
  - "Just give me the answer."
  - note: software is generally *low risk* in introductory courses.

How to learn

- Experiment with coding (learning by doing)
  - Try compiling and running examples
  - Try changing parts and see if the effects are what you expect
- Ask questions
  - You can ask questions in the lectures, discussion forums, or consult tutors directly in the practical and tutorial sessions.
- Use Internet resources
  - Java API: Everything built into Java, in one place
  - `stackoverflow.com` is one of the richest programming resources
  - Often `stackoverflow.com` pages are listed at the top of search engine results
  - You can also ask general questions on these sites (but not how to do specific pieces of assessment)

Good practices

- Keep a question log
- *"Start"* assignments as soon as you get them:
- Leave time for problems to arise
- Regular study/review
- Experiment with coding
- Read the discussion forum regularly
- Be mentally active during lectures — be careful of coms devices.
- Talk to student services if you have an ongoing diagnosed issue impacting your studies and you want an SAPD.

Good practices

- Keep a question log
  - Make efficient use of tutor time.
  - Don't need to remember it.
- *"Start"* assignments as soon as you get them:

- Leave time for problems to arise
- Regular study/review
- Experiment with coding
- Read the discussion forum regularly

- Be mentally active during lectures — be careful of coms devices.
- Talk to student services if you have an ongoing diagnosed issue impacting your studies and you want an SAPD.

Good practices

- Keep a question log
  - Make efficient use of tutor time.
  - Don't need to remember it.
- *"Start"* assignments as soon as you get them:
  - Get the spec
  - Read it
  - Note any immediate questions
- Leave time for problems to arise
- Regular study/review
- Experiment with coding
- Read the discussion forum regularly

- Be mentally active during lectures — be careful of coms devices.
- Talk to student services if you have an ongoing diagnosed issue impacting your studies and you want an SAPD.

Good practices

- Keep a question log
  - Make efficient use of tutor time.
  - Don't need to remember it.
- *"Start"* assignments as soon as you get them:
  - Get the spec
  - Read it
  - Note any immediate questions
- Leave time for problems to arise
- Regular study/review
- Experiment with coding
- Read the discussion forum regularly
  - So you know what is there when you need it later.
  - So you know what sort of questions are being asked.
- Be mentally active during lectures — be careful of coms devices.
- Talk to student services if you have an ongoing diagnosed issue impacting your studies and you want an SAPD.

CSSE2002 — You

We (the teaching staff) will assume that you have passed CSSE1001 or equivalent, so you can use the following in Python:

We do not assume:

CSSE2002 — You

We (the teaching staff) will assume that you have passed CSSE1001 or equivalent, so you can use the following in Python:

- variables
- control flow (loops, ifs)
- functions
- lists

We do not assume:

CSSE2002 — You

We (the teaching staff) will assume that you have passed CSSE1001 or equivalent, so you can use the following in Python:

- variables
- control flow (loops, ifs)
- functions
- lists

We do not assume:

- that you know Java (at first).

What 2002 isn't

2002 is not a Java course — We will be teaching you *some* Java but knowing everything (or even most things) about Java is not the goal.

Programming in the small (again)

For those of you coming straight from CSSE1001, code you have written up to now has likely been:

1. small ($< 10,000$ lines of code)
2. writen solely by you
3. probably relatively easy to debug
4. for a limited purpose (i.e. receive marks)
5. to exist for a limited time

Programming in the small (again)

For those of you coming straight from CSSE1001, code you have written up to now has likely been:

1. small ($< 10,000$ lines of code)
2. writen solely by you
3. probably relatively easy to debug
4. for a limited purpose (i.e. receive marks)
5. to exist for a limited time

The goal of programming up until now has been to write small amounts of code that works until it is marked.

What does a “large” software project look like?

`Linux kernel, Version 4.1`

- ≈ 19.5M lines of code (2015)
- Contributions from more than $13,500$ developers (2015)
- ≈ 24 years of development (as at 2015)
- Estimated at \$3 Billion cost to redevelop (2011)
- The Linux kernel powers mobile phone OS Android, which according to Google has over 2 Billion monthly users (2017)

(`https://www.phoronix.com/scan.php?page=news_item&px=Linux-19.5M-Stats`)

(`http://linuxcost.blogspot.com/2011/03/cost-of-linux.html`)

(`https://twitter.com/Google/status/864890655906070529`)

Is this software…

1. small?
2. writen by one person?
3. easy to debug?
4. for a limited purpose?
5. to exist for a limited time?

Is this software…

1. small?
2. writen by one person?
3. easy to debug?
4. for a limited purpose?
5. to exist for a limited time?

? - no not really.

You may end up working on software which is:

1. large — 100KLOC and up
2. writen by many people
3. impossible for one person to understand all at once
4. the purpose and features of which develops over time
5. needs to work for years

Tasks programmers do

Professionally, you will need to:

- Document your code
- Debug it
- Test it

Tasks programmers do

Professionally, you will need to:

- Document your code — because someone who didn't write it (or doesn't remember it) may need to work on it.
- Debug it
- Test it

Tasks programmers do

Professionally, you will need to:

- Document your code — because someone who didn't write it (or doesn't remember it) may need to work on it.
- Debug it — the more code you write the more opportunity to make mistakes.
- Test it

Tasks programmers do

Professionally, you will need to:

- Document your code — because someone who didn't write it (or doesn't remember it) may need to work on it.
- Debug it — the more code you write the more opportunity to make mistakes.
- Test it — it is **far** better for you to find bugs than for someone else to find them.
  You want to have a reason for confidence in your work beyond "I hope it is okay".

Tasks programmers do

Professionally, you will need to:

- Document your code — because someone who didn't write it (or doesn't remember it) may need to work on it.
- Debug it — the more code you write the more opportunity to make mistakes.
- Test it — it is **far** better for you to find bugs than for someone else to find them.
  You want to have a reason for confidence in your work beyond "I hope it is okay".

CSEE2002 teaches practices that support documenting, debugging, and testing of code to allow programming to scale to large software projects.

Contact hours

Each week:

Activity		Hours / Week	
Lectures	-	two x 1	
Tutorial	-	1	Theory exercises
Prac	-	2	Programming exercises

Tutorials and Pracs start in Week 2.

Make sure you have signed up in SINet soon. Contact `signon@eait.uq.edu.au` if you have problems.

Discussion in pracs etc

Pracs involve:

- Exercises
- Assignment help

You can discuss exercises but the aim is to help you learn for the assignments. Not for your group to substitute for individual competance. Assignments must be solo.

Assessment

2002

Assessment Task	Due Date	Weighting	Learning Objectives
Programming assignment Assignment 1	31 Aug 18 12:00	10%	3, 4, 5, 6
Programming assignment Assignment 2	21 Sep 18 12:00	15%	1, 2, 3, 4, 5, 6
Programming assignment Assignment 3	19 Oct 18 12:00	15%	1, 2, 3, 4, 5, 6
Exam - during Exam Period (Central) Final examination	Examination Period	60%	1, 2, 3, 4, 5, 6

7023

Assessment Task	Due Date	Weighting	Learning Objectives
Online Quiz Java Quiz	27 Aug 18 18:00 - 30 Aug 18 23:59	8%	2, 5
Programming assignment Assignment 1	31 Aug 18 12:00	8%	3, 4, 5, 6
Programming assignment Assignment 2	21 Sep 18 12:00	13%	1, 2, 3, 4, 5, 6
Programming assignment Assignment 3	19 Oct 18 12:00	13%	1, 2, 3, 4, 5, 6
Exam - during Exam Period (Central) Final examination	Examination Period	58%	1, 2, 3, 4, 5, 6

- You must get a mark of at least 40/100 in the final exam in order to pass the course.
- **No extensions for assignments.**
- Read the ECP for dates and rules.