Computer Science 159

Me: Gara Pruesse, Ph.D.

What I love about computer science: Algorithms

I will take attendence.

"Coding was hard until I learned these 5 things"

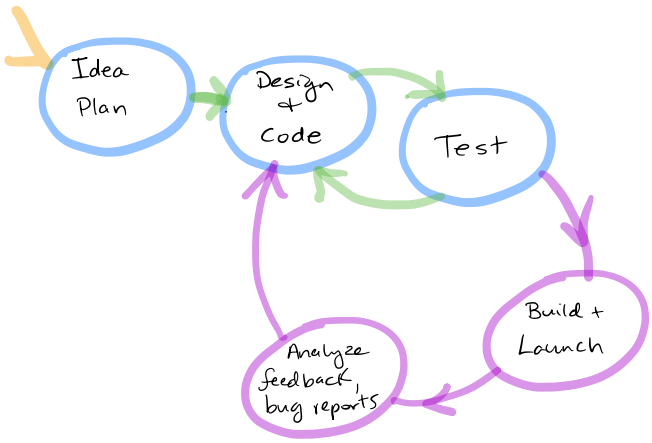
- Elsa Scola, YouTube

1. Learn by doing
2. Learn to program, not a programming language
   - does not make sense to "memorize the algorithm in a given language"
   - understand the algorithm underlying the code.
3. Create a roadmap
   - What do you want to build
4. Prioritize Understanding
   - When you fix broken code, make sure you know why the solution works
5. Get used to Failing
   - Seek failure

No discomfort, no expansion

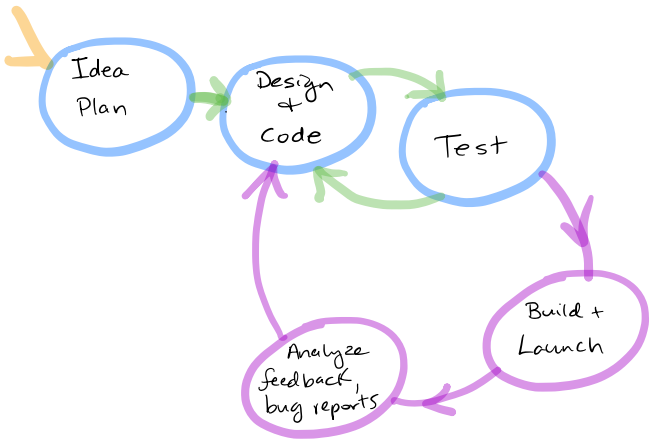
Sept 2-4 2024

Software Development Life Cycle (SDLC)



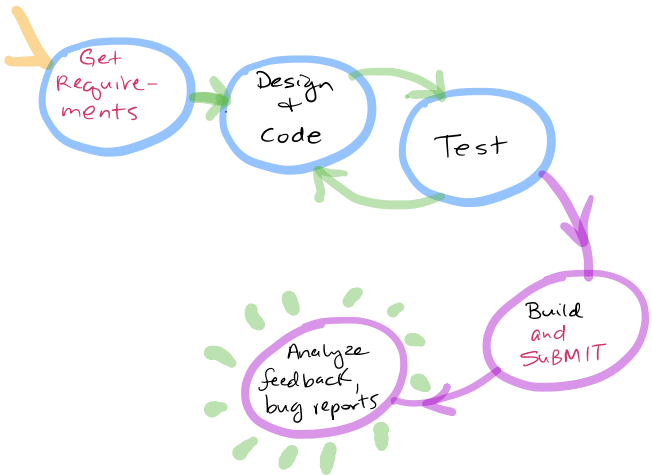
Sept 2-4 2024

Software Development Life Cycle (SDLC)



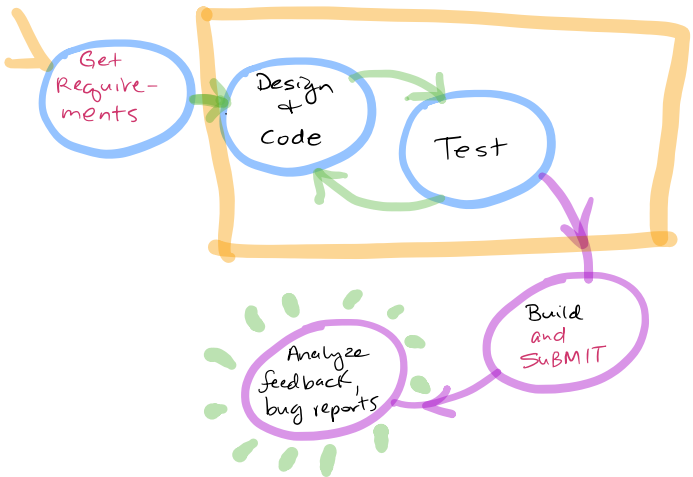
Sept 2-4 2024

Software Development Life Cycle (SDLC)



Sept 2-4 2024

Software Development Life Cycle (SDLC)

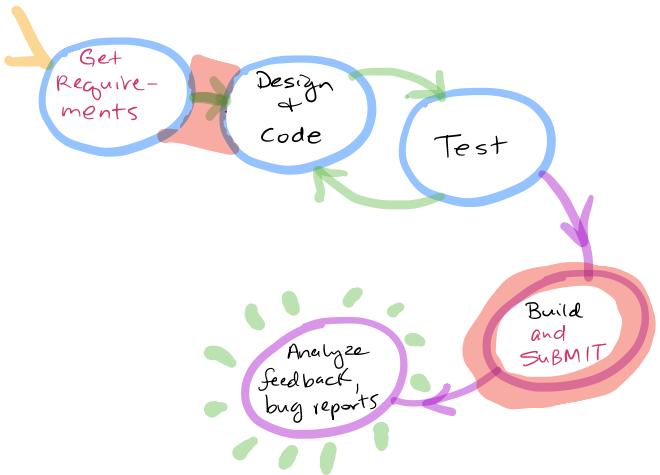


← most of the work of the course

Sept 2-4 2024

Software Development Life Cycle in CSCI 159

(SDLC)

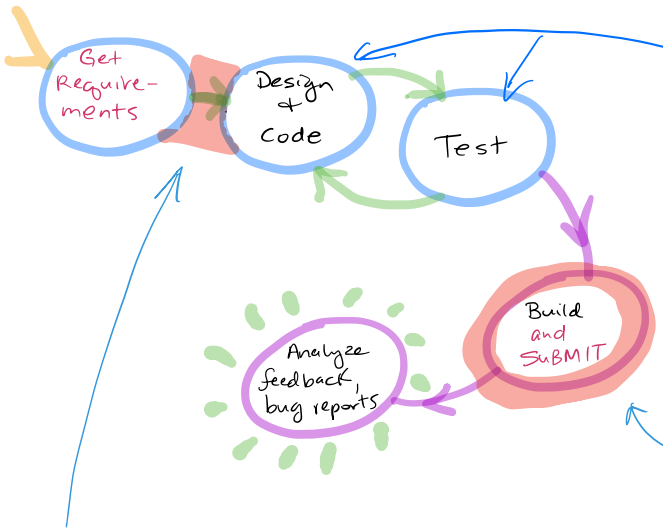


← most of the work of the course

← there can be glitches here, too

Software Development Life Cycle in CSCI 159 (SDLC)

Sept 2-4 2024



most of the work of the course

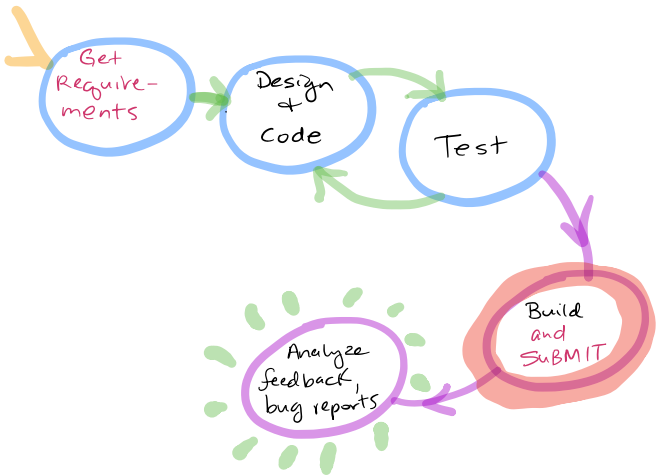
← there can be glitches here, too

solve these problems by talking to
- Gara
- CSCI Help Centre
- peers
- Merlin

- Solve these problems by
  - asking Gara
  - discussing with peers
  - talking to Help Centre staff

Sept 2-4 2024

The CSCI 159 Toolchain



Sept 2-4 2024

The CSCI 159 Toolchain

Get requirements

- go to csci.viu.ca/~gpruesse/teaching/159
- Click on labs ⇒ click on lab for the week.

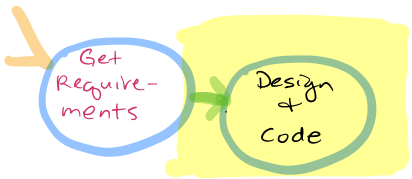
unclear what is being asked?

① Think about how the program's behaviour is described - that is the goal or end result

② Think about the specifications for the programs methods - algorithm
- language features that are to be utilized

Sept 2-4 2024

The CSCI 159 Toolchain

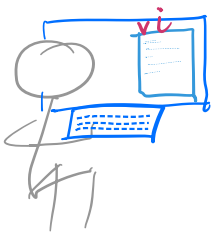


Design & Code

1.

- Think about what is required, and how you can make the computer accomplish it.
- Understand the algorithm that you will implement (i.e. step-by-step method that logically produces the desired behaviour)
- Select the language features you will use to implement the algorithm (if-then or switch? for-loop or recursion?)

2.



- Log on to a CSCI linux machine
  - go to the correct directory
  - edit a file that is correctly named for the current lab.
    You can use the editor of your choice, but your instructor only really knows vi
    vi gedit pico
  - using the selected High-Level Language (HLL) C++ language, code the program

code the program (in C++)

About that....

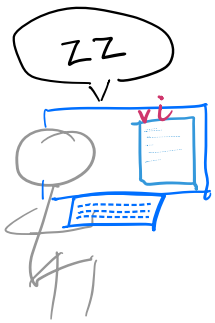
- no matter the editor, the end result of writing code is a **text file** which is a string of keyboard characters. (Unlike, say, a Word document, which is full of formatting data.)

The **text file** could be named **foo.bar**, and could contain

- Your thoughts on modern life
- ASCII art
- C++ code for a 159 lab

In all cases, you can

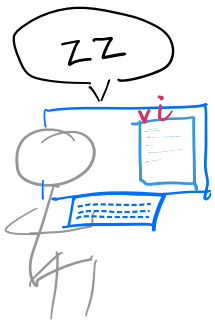
- save the file
- reopen it later and amend it, using same editor or a different one



After you have your first draft coded, save the file.

If the file is C++ code, you can compile it by invoking the C++ compiler g++ on it:

```
> g++ foo.bar
```

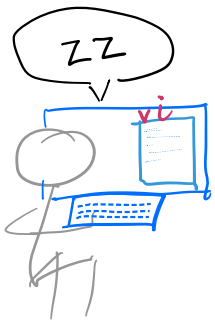


After you have your first draft coded, save the file.

If the file is C++ code, you can compile it by invoking the C++ compiler g++ on it:

```
> g++ foo.bar
```

this is the prompt - you don't type it, it is already there

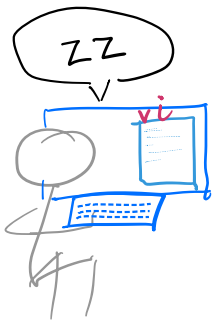


After you have your first draft coded, save the file.

If the file is C++ code, you can compile it by invoking the C++ compiler g++ on it:

```
> g++ foo.bar
```

↑ this is a call to the gnu C++ compiler



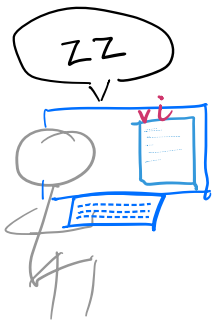
After you have your first draft coded, save the file.

If the file is C++ code, you can compile it by invoking the C++ compiler g++ on it:

```
> g++ foo.bar
```

← this is the argument for the compiler — the file containing the C++ code.

↑ this is a call to the gnu C++ compiler



After you have your first draft coded, save the file.

If the file is C++ code, you can compile it by invoking the C++ compiler g++ on it:

```
> g++ foo.bar
```

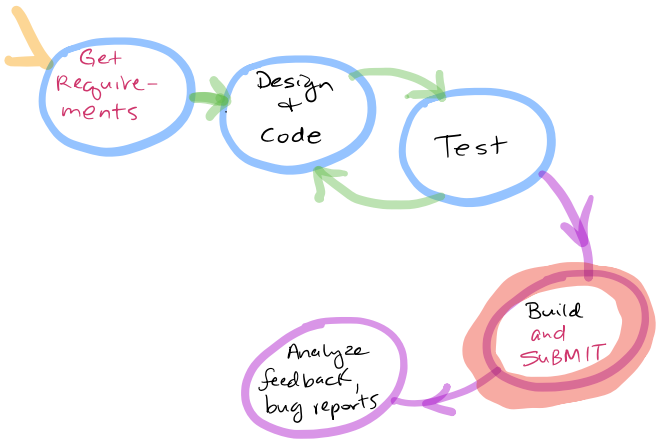
Wait! This always gives an error message.
Rename the file to foo.cpp

Our compiler requires that file names adhere to this convention.

```
> mv foo.bar foo.cpp
> g++ foo.cpp
```

Sept 2-4 2024

The CSCI 159 Toolchain



How to submit

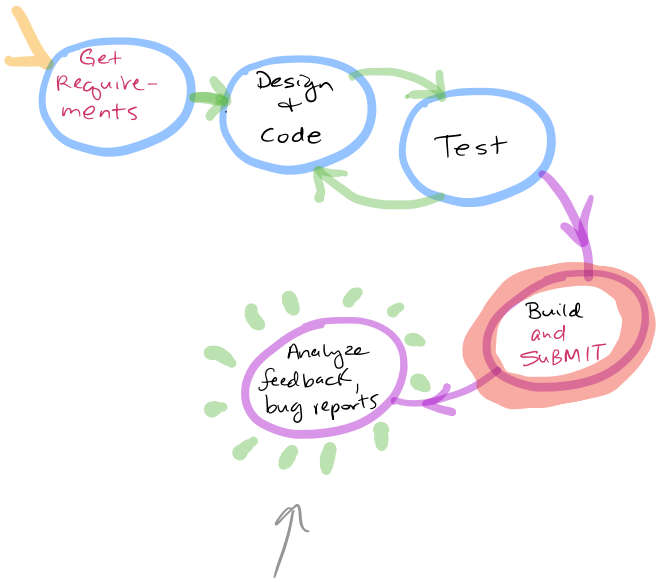
We will learn how to do it in the lab.

In brief:

- you can do all the coding, compiling, testing that you like in linux directories you create within your home directory
- But when it comes time to submit, you must create a special directory that has "links" to the csci159 git repository.
- I will be providing a "makefile" that automates the creation of this special directory, and possibly populates it with "seed" files.

Sept 2-4 2024

The CSCI 159 Toolchain



I will provide some feedback (possibly "terse mode")
Feel free to talk with me during my office hours, or in the lab, or talk to Help Centre staff

My office hour is Tue $11^{30} - 12^{30}$.
Once the Help Centre schedule is fixed, I'll find another hour that works within that framework.

Let's have fun learning to code!

How your mark in the course will be arrived at:

- about 6 lab exercises 36%
- about 4-5 quizzes, done in lab 25%
  (different quizzes for the two labs)
- final exam 39%

Best strategy for getting a good mark:

- learn by doing
- keep up
- practice coding
- before coding (or once you have your "frame" built) think about your algorithm/logic design.
  A half-hour spent improving the design saves tens of hours of debugging.
- document your code as you go.