SENG 265:
Software Development Methods

Summer 2019

University of Victoria
Department of Computer Science

SENG 265

- **Software Development Methods**
- Instructor: Michael Zastre
  - ECS 528
  - e-mail: zastre@uvic.ca (please include "SENG 265" in subject line)
  - Office hours: Wednesdays 1:30 pm to 3:00 pm; Thursdays 2:30 pm to 4:00 pm; or by appointment
- Labs:
  - Begin week of May 13th
  - ELW Building, room B238

Administrative Details

- Course website:
  - Via “connex.csc.uvic.ca”
  - Course appears as a tab when you log into conneX
  - The tab might not immediately appear if you've taken several CSC courses already
- Lab sections:
  - Our focus is on hands-on + tutorial components
  - You must register for a lab section
  - Attendance at labs is mandatory

Your course account

- The details below (and more) will be covered in the first lab session
- Use your Netlink credentials
- You'll be able to remotely login to any of the lab machines or into the server
- These machines all run Centos 7 Linux

Grading

- Breakdown:
  - assignments: 40% (four assignments @ 10%)
  - lab work: 6% (10 labs)
  - midterm exam: 18%
  - final exam: 36%
- Marking concerns (“one-week rule”)
- Midterm: June 18 (Tuesday)
- Final exam: Scheduled by University
- Course outline: `https://bit.ly/2PTIi05`

Purpose of Course

- General introduction to:
  - UNIX/Linux environment and scripting
  - production languages (C & Python)
  - software development methodologies
- Introduction to software-engineering concepts/vocabulary
- Working at a higher level of abstraction
- Acquiring and reinforcing good habits when writing software and software systems

Context

- Your experience thus far:
  - small, relatively simple programs
  - provided with steps to solving specific problem
  - written alone
  - no ongoing maintenance
- What awaits in industry:
  - large and complex projects
  - do not know ahead of time how to solve the problem
  - work in teams (often very specialized)
  - ongoing maintenance is critical

Course topics

- UNIX/Linux fundamentals
- C programming
- Python programming
- Source code control, code revision and change management
- Inspection, profiling, testing and debugging
- Introduction to software development "process" and related concepts

By the end of this course…

- You should be able to:
  - program with some comfort in a UNIX environment
  - use Python for prototyping, and other kinds of scripting for support of code testing and debugging
  - recognize a problem statement that can become a program specification
  - use general-purpose languages such as C and Python to solve programming problems
  - work with code versioning systems (git) to manage changes in your own code
  - apply some general software engineering techniques to your own projects
  - be ready to delve deeper into more formal software engineering approaches

Academic integrity

- Guiding principles
  - discussion is encouraged …
  - .. but work submitted for credit must be your own
  - in cases where attribution is appropriate, it must be given
  - example: code taken from a textbook or web-based tutorial
  - example: algorithm based on a journal paper
- If you are unclear, please ask the instructor.
- Attribution for these slides!
  - They were originally created by myself and then lightly edited by Nigel Horspool, then more by me, then more by him, etc. etc.

Development environment

- At first glance there would appear to be two families of development tools:
  - those which employ a **GUI** (graphical user interface), typically as part of an **IDE** (integrated development environment)
  - those which employ a **CLI** (command line interface)
- An IDE tends to hide many of the details of the development process

Development environment (2)

- In this course we use a command-line interface;
  - this gives us a better understanding of aspects of the development process which might be hidden inside an IDE
  - compilation
  - source code management
  - profiling
  - testing, etc.
- Our command line interface ("bash") is run within the Linux variant of Unix.

Purpose of our environment choice

- Simplicity
- Universality
- Professional (sometimes more powerful) tools
- Less “mysterious automation” of programming steps
- Not intended to make your life harder:
  - absence of tools with which you are familiar is not necessarily bad
  - goal is that you should be able to make an informed choice when deciding upon your tools and environment for a given task

Software development process

- Building software systems is a social task
  - For which we use – amongst other things – technical tools and systems
- 50+ years of experience on what can work when building large software systems
  - ... along with spectacular and costly failures
- One outcome of the course:
  - Become aware of what is meant by such a "process"...
  - ... and recognize the way in which techniques learned in this course apply to some parts of the process

Development models: waterfall

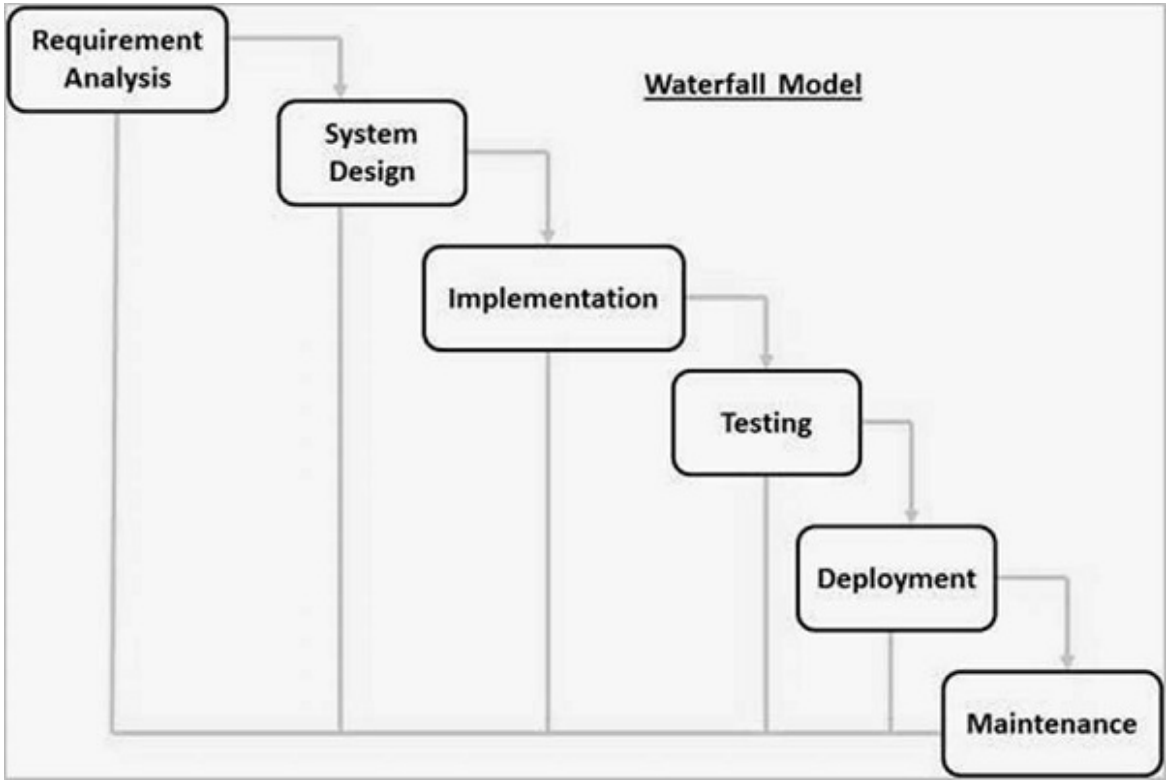


Image: https://tiny.cc/3yx91y

Development model: spiral

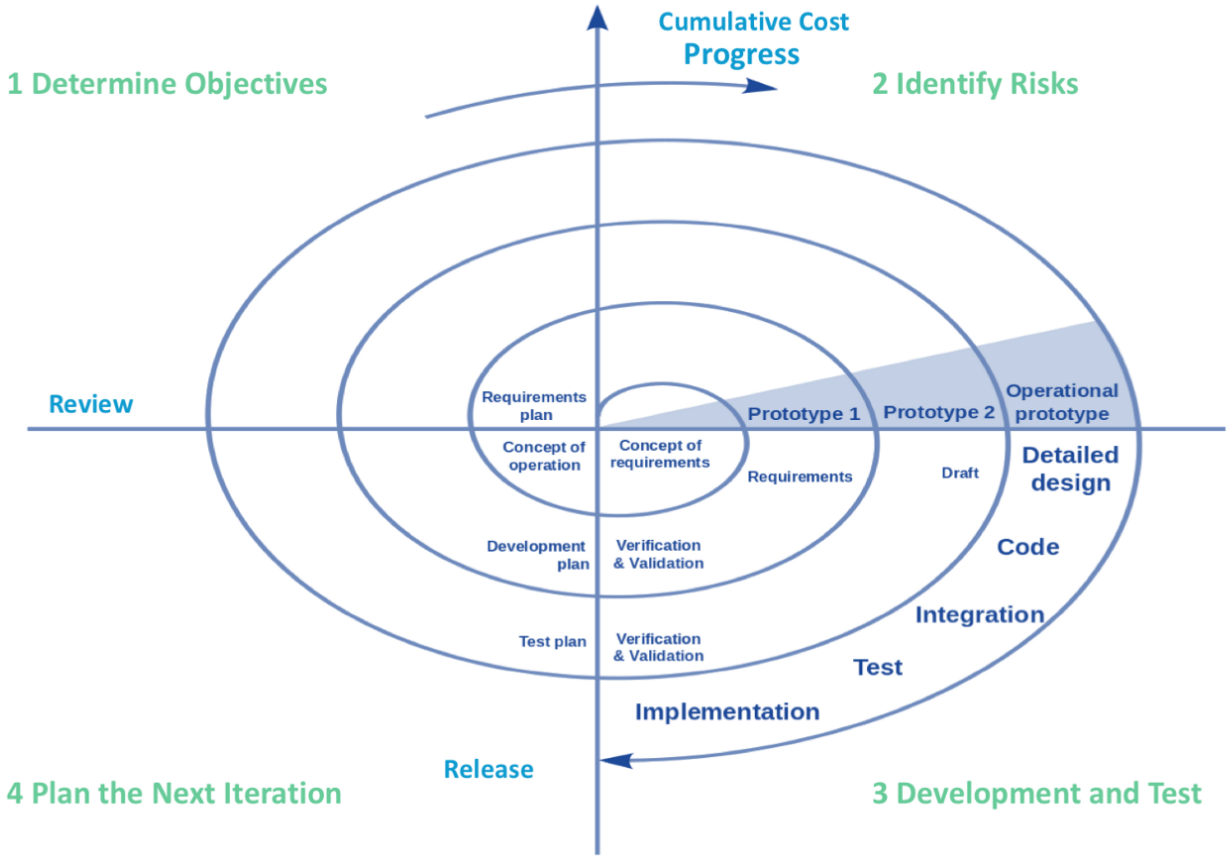


Image: https://tiny.cc/45x91y

Development model: agile

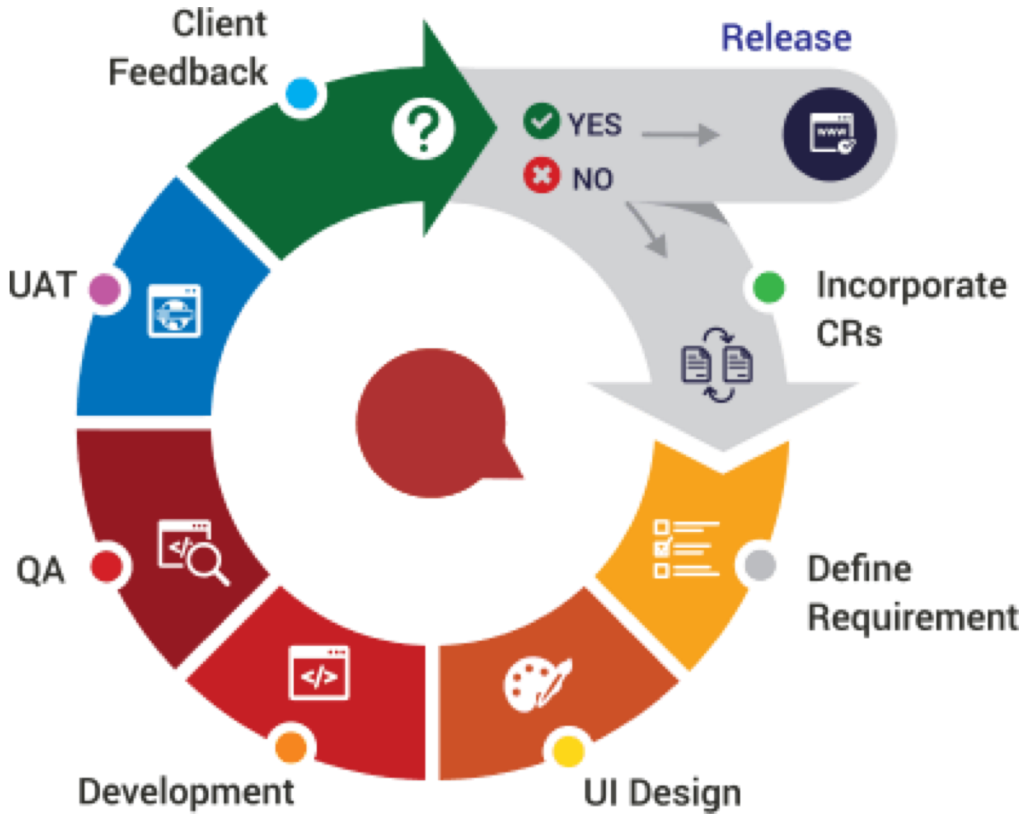
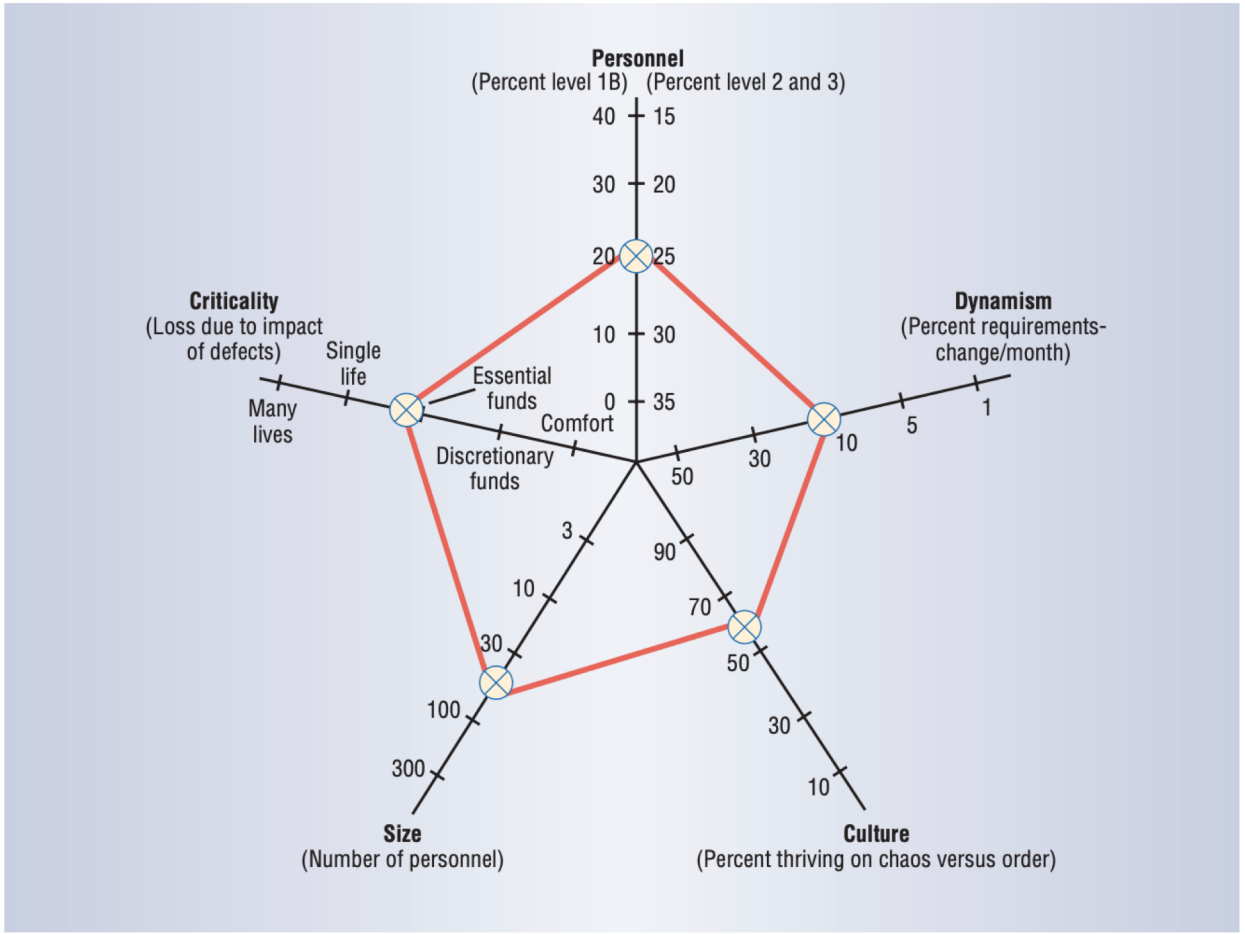


Image: https://tiny.cc/lay91y

Context

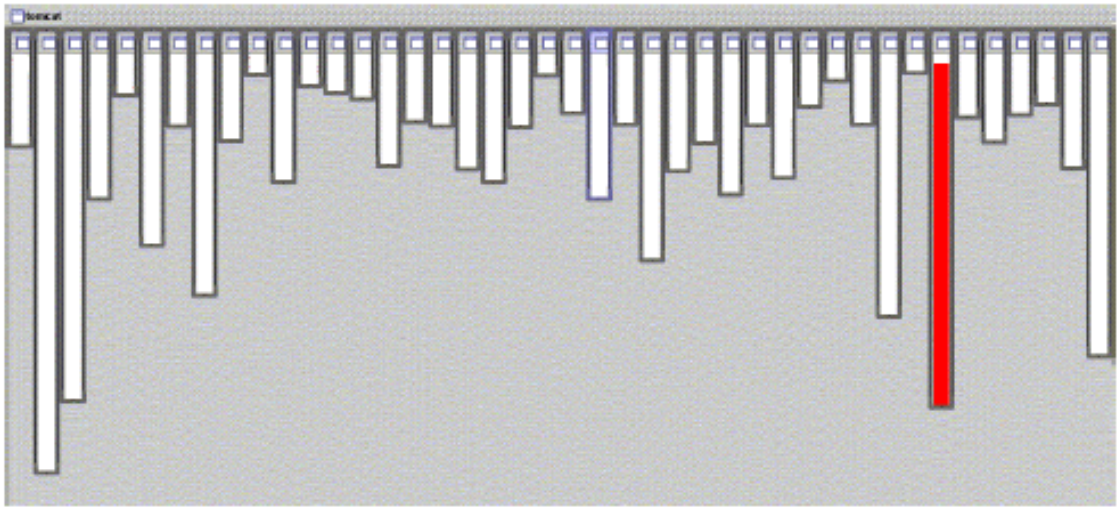


Full text of article: https://tiny.cc/yg091y

Another SENG concept

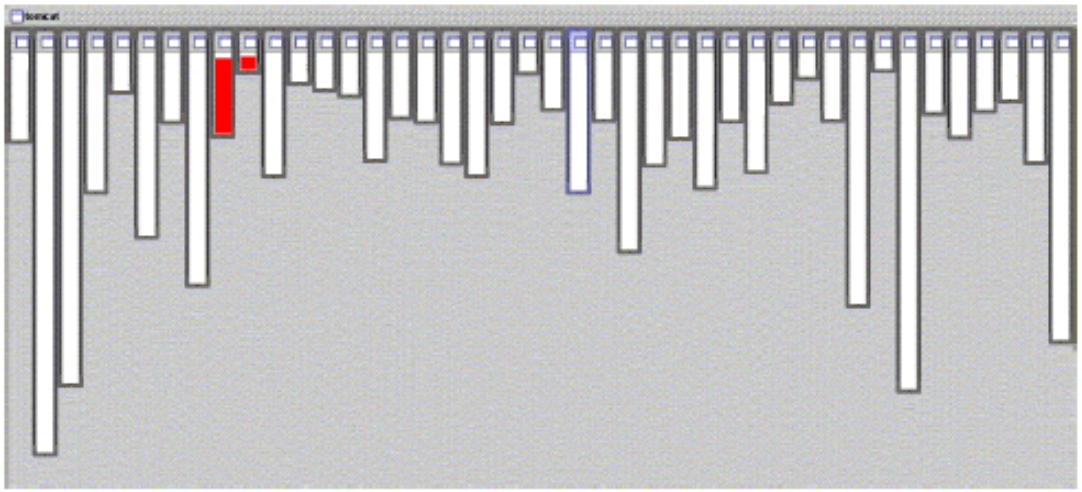
- "low coupling / high cohesion"
  - minimize dependencies between/amongst modules
  - maximize related functionality within module
- Insight:
  - Design modules such crucial information/data for understanding the module is within the module
  - (One of the reasons global variables are discouraged.)

code locality in Tomcat



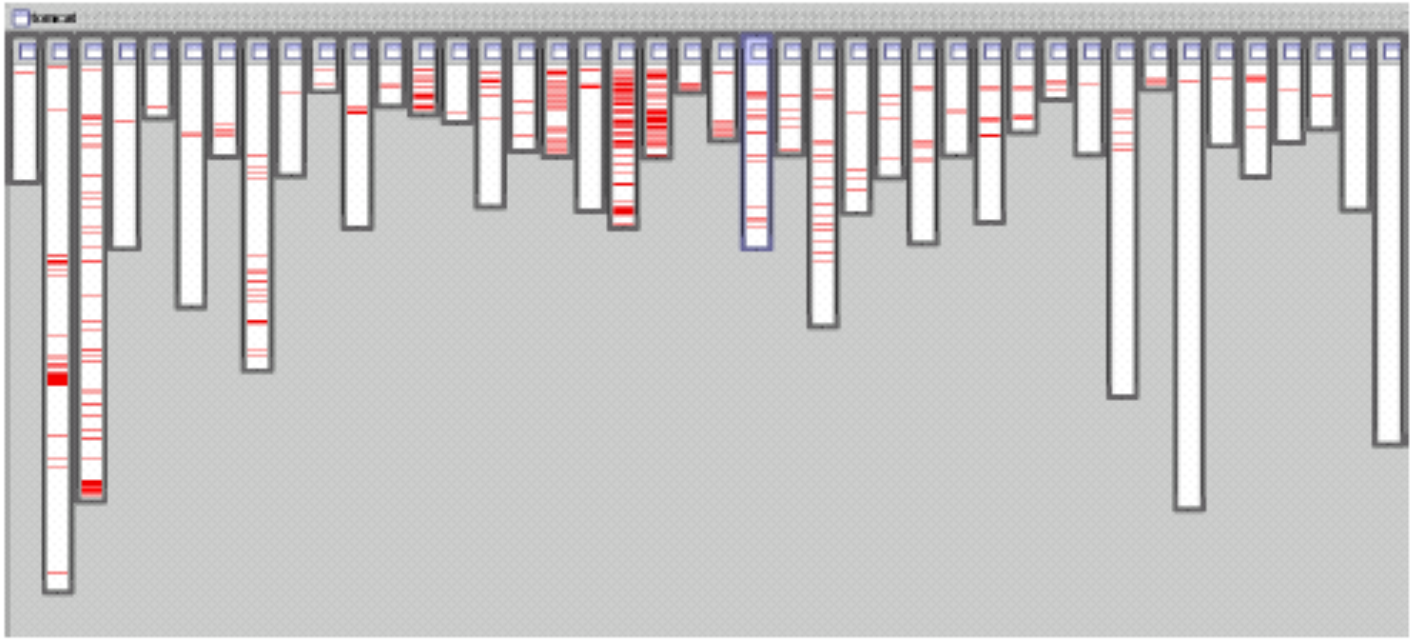
- XML parsing in org.apache.tomcat
  - each column corresponds to a class
  - length of each column indicates size of class
  - red colour represents the code lines relevant to XML parsing
  - what this shows is good modularity (i.e., all XML parsing code is in one module)

code locality in Tomcat (2)



- URL pattern matching in org.apache.tomcat
  - functionality is now spread over two classes
  - modularity is still good
    - any changes to URL pattern matching code restricted to these two modules

problems



- logging code in Tomcat is not well-modularized
  - again, red corresponds to lines of code devoted to logging
  - nearly every class has some logging code
  - classic example of “cross-cutting concern”
    - concern: logging
    - cross-cutting: across many classes

Next steps

- Introduction to Unix
  - Its architecture
  - Use of the shell
  - Working within the shell
- Git
  - open-source version control system (VCS)
  - widely used, yet with wildly varying workflows
  - (Swiss-army knife approach to VCS…)
  - we will use our own BSEng Git server (i.e., we will **not** use GitHub)