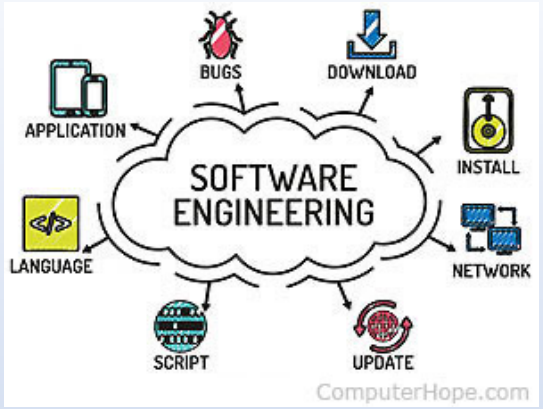
Course Introduction

CSCI2340: (Graduate) Software Engineering

Steven P. Reiss

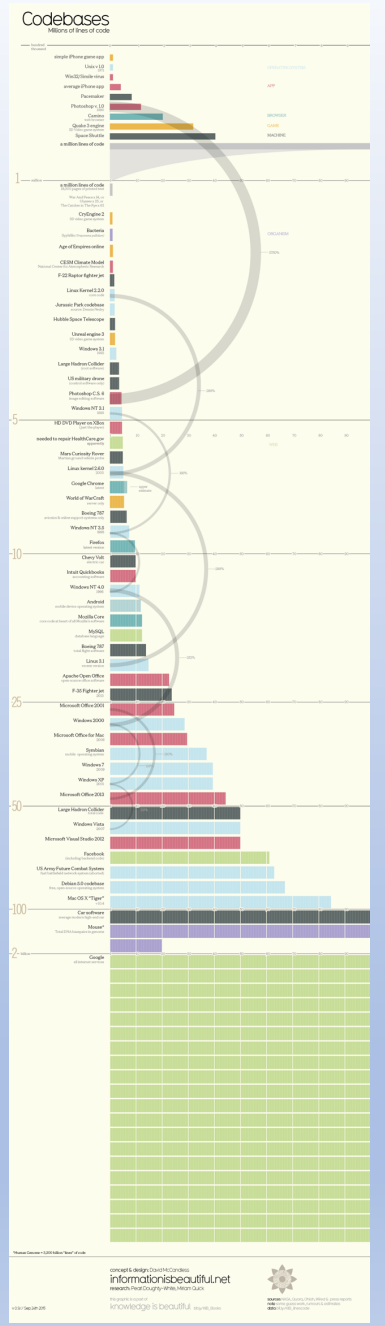
What is Software Engineering



- The systematic application of engineering approaches to the development of large software systems.
- The application of a systematic, disciplined, computable approach for the development, operation, and maintenance of large software systems.
- A process of analyzing user requirements and then designing, building, and testing a software application which will satisfy those requirements.

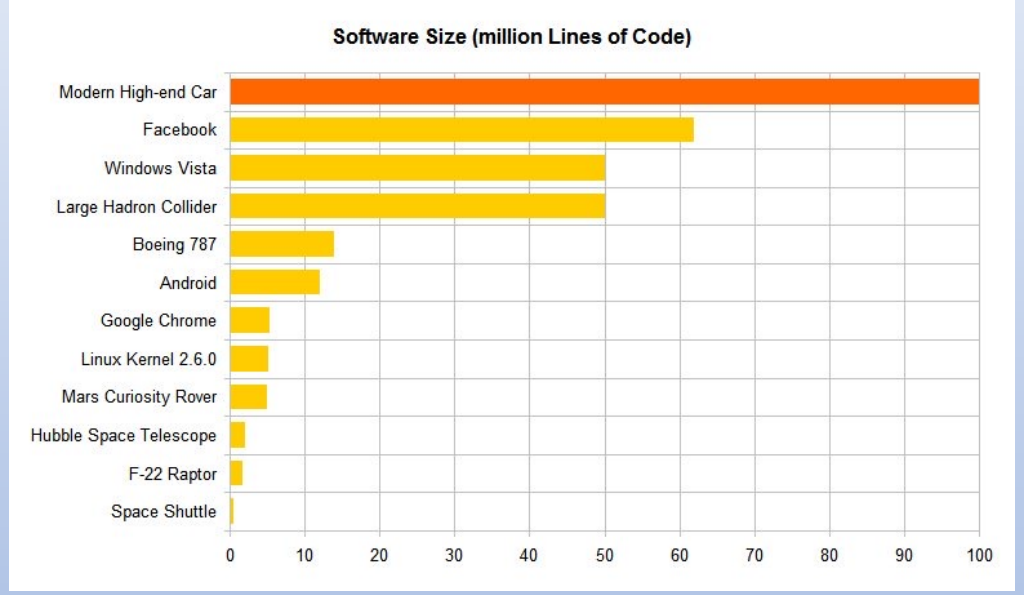
What Has It Accomplished

- Size of software systems has increased
  - By about a factor of 10 each decade
- History
  - 1970: Box of punch cards (2000), large program: 10,000 LOS
    - Compile time in lines per minute (10-1000)
  - 1980: 5M disk was large, large program: 100,000
  - 1990: 40M disk was std, large program: 1,000,000
  - 2000: 10G disk was std, large program: 10,000,000
  - 2010: 1T disk is std, large program: 50,000,000 (windows)
  - 2020: 10T disk is std, large program: 2,000,000,000 (google web)
    - 100,000,000 in a car



Question: What is YOUR Largest System

- 1,000 - 2,000 lines (CS15/18)
  - You shouldn't be here
- 10,000 - 20,000 lines (CS32)
  - Probably a bit less, but designed for
- 200,000 lines (Internship)
- 2,000,000 lines
- Larger
  - How much of this did you understand



Software Engineering

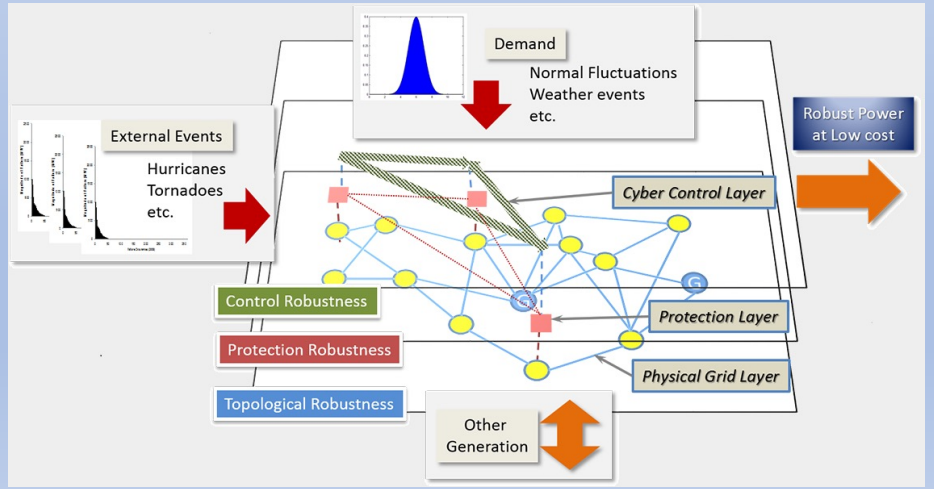
- Has focused on how to build large software systems
  - How to build software at scale
- Has developed tools, techniques & frameworks
  - For building software at scale
- Attempts to address the various problems
  - When building software at scale
- Let’s get an overview of what all this means

What is a Large Software System

- Could be measured in lines of code or source
  - But this varies over time, language, etc.
  - Still the only meaningful measure
- A system that is too complex for one person to understand
- A system that takes more than K man-years to build
  - K = 10?
  - Mythical Man Month

Large Software System Characteristics

- Require multiple developers
- Long-lived
- Evolve, not built all at once
- Distributed (client-server, web-based, multiple servers)
- Concurrent (multithreaded, multiprocess)
- Multiple languages
- Prone to failure
  - Guaranteed to have bugs
  - Often 1000's



Software Engineering at Scale

- How to build large systems
  - *Multiple person development*
  - Long-lived
  - *Evolve, not build at once*
  - *Distributed*
  - *Concurrent*
  - Avoiding failure

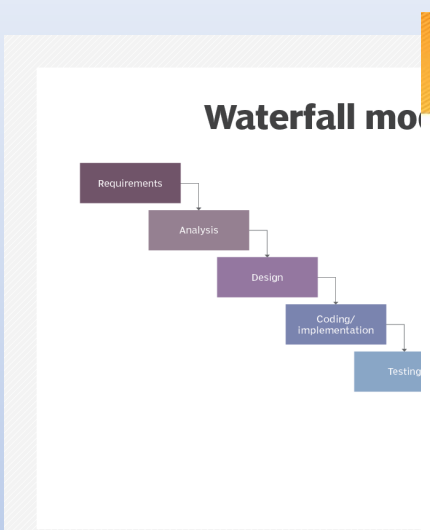
Software Engineering at Scale

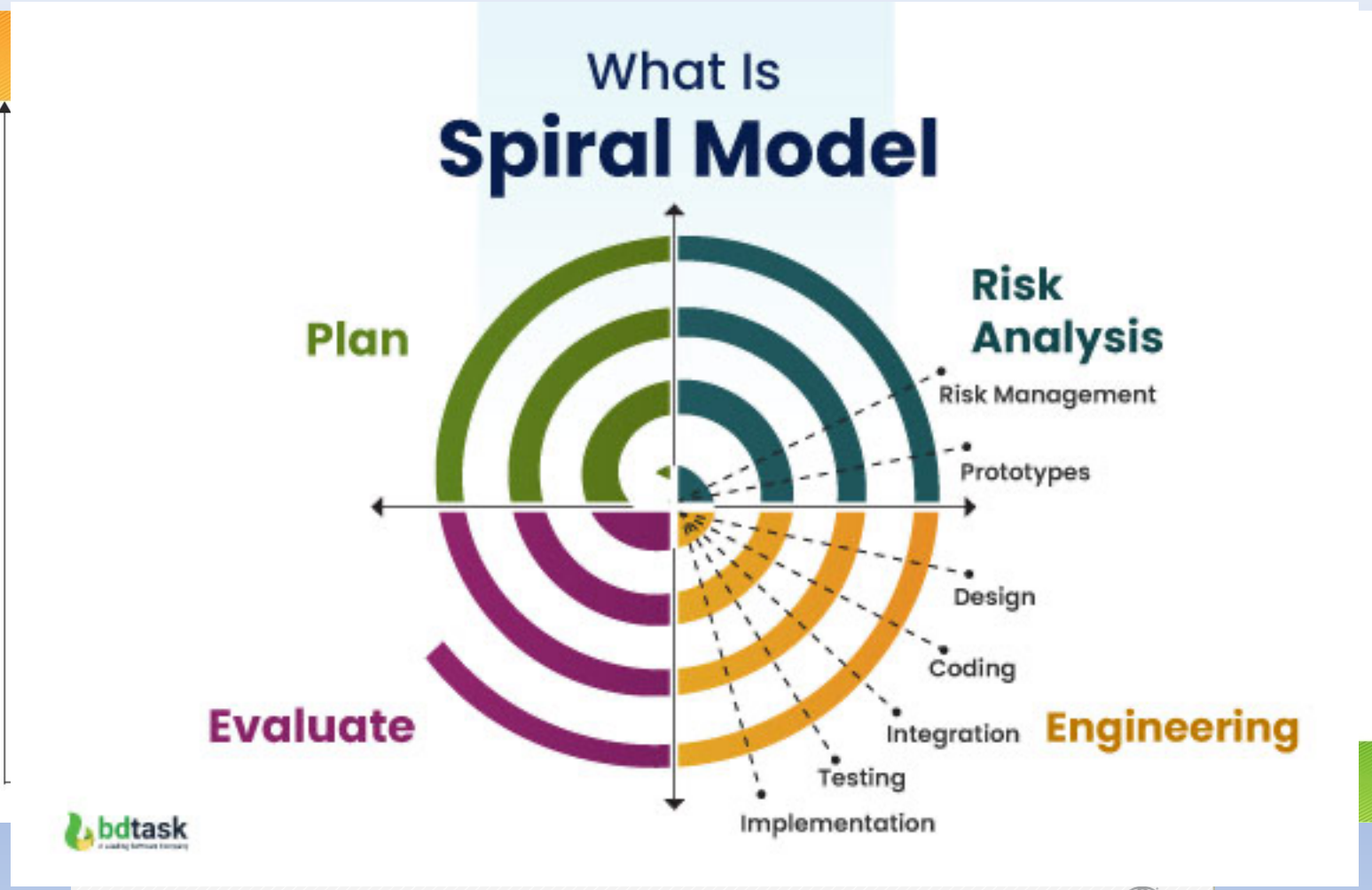
- Importance of Maintenance
  - Multiple person development
  - *Long-lived*
  - *Evolve, not build at once*
  - Distributed
  - Concurrent
  - *Avoiding failure*

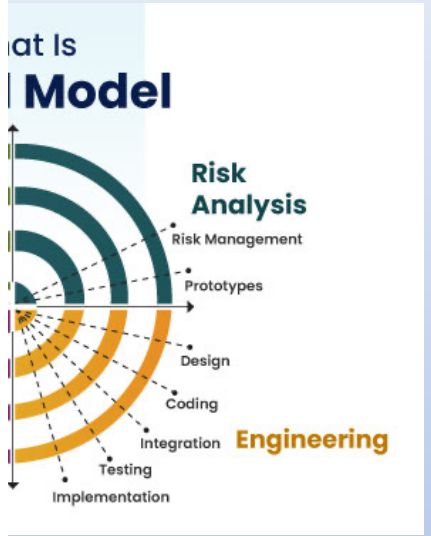
Software Engineering at Scale

- Importance of Managing Risk
  - *Multiple person development*
  - *Long-lived*
  - *Evolve, not build at once*
  - *Distributed*
  - *Concurrent*
  - *Avoiding failure*

Traditional Software Engineering

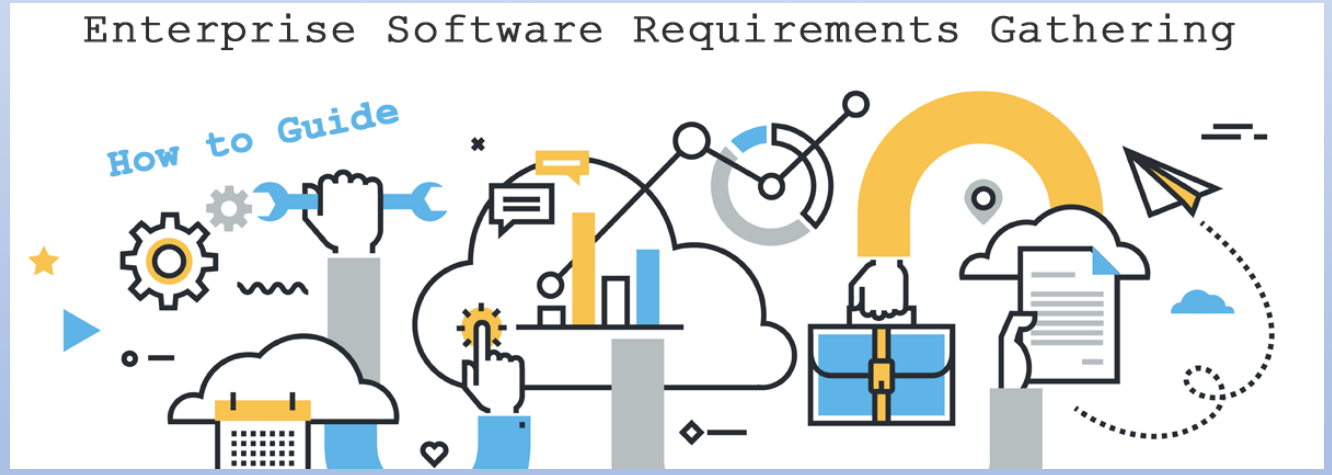






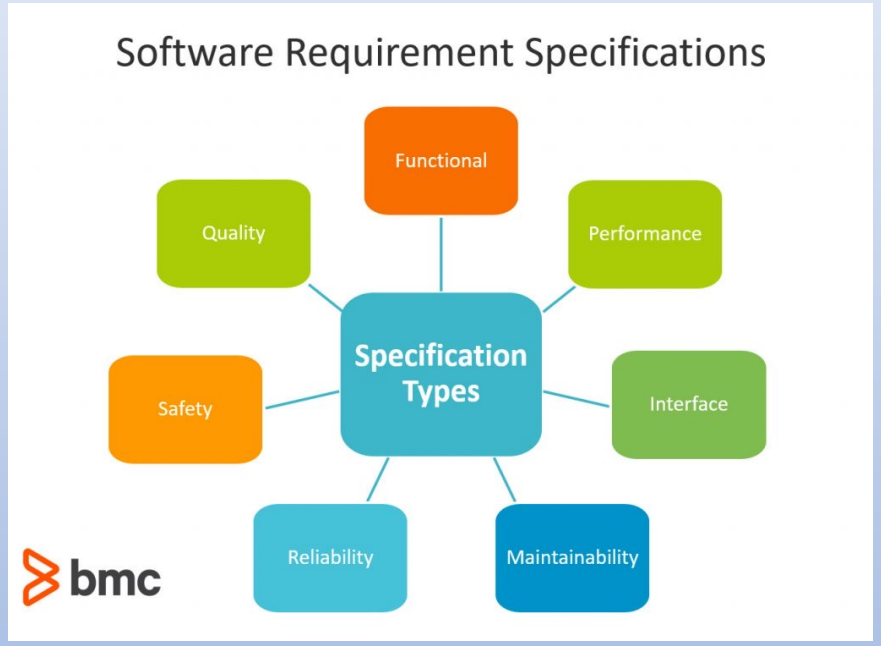
Software Engineering at Scale: Requirements

- Determine what the user needs
- Define the problem to be solved
- Done over time
- Evolving
- More user-oriented



Software Engineering at Scale: Specifications

- Determine what should be built
- Feature-based specification
- Risk-based specification
- Minimal Viable Product concept
- Include more than just functionality
  - Safety, performance, reliability
  - Quality, maintainability
  - Documentation

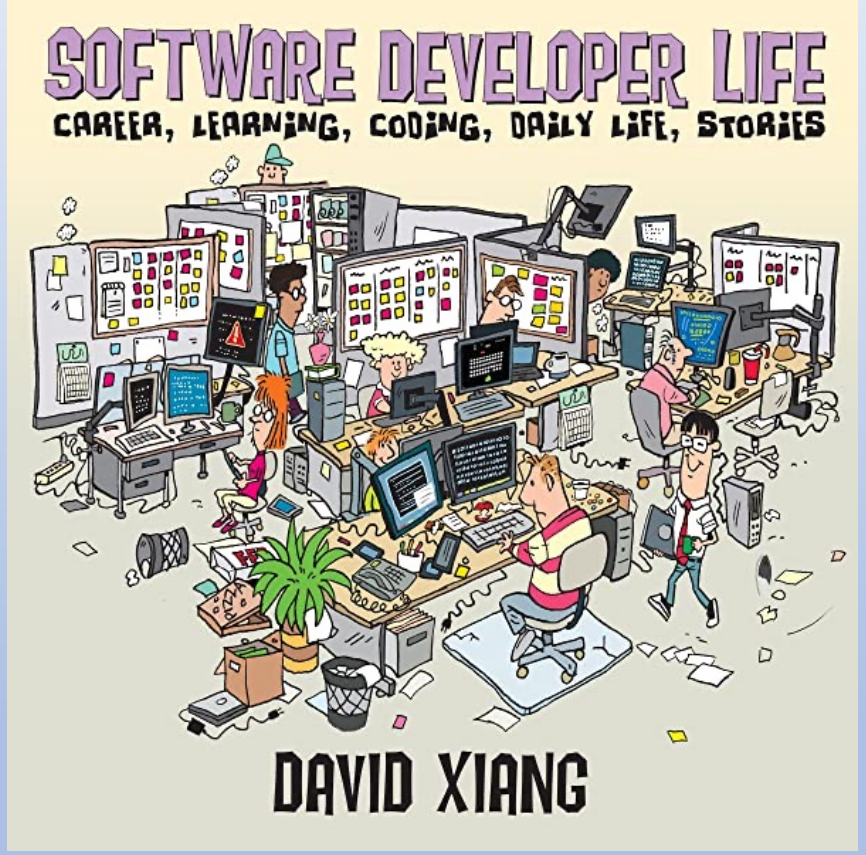


Software Engineering at Scale: Design

- Design what will be built
- Software Architectures
- High-level design using interfaces
- Object-oriented design
- Risk-based design
- Team-based design
- Design for maintenance
- User interface design
- Distributed/concurrent system design

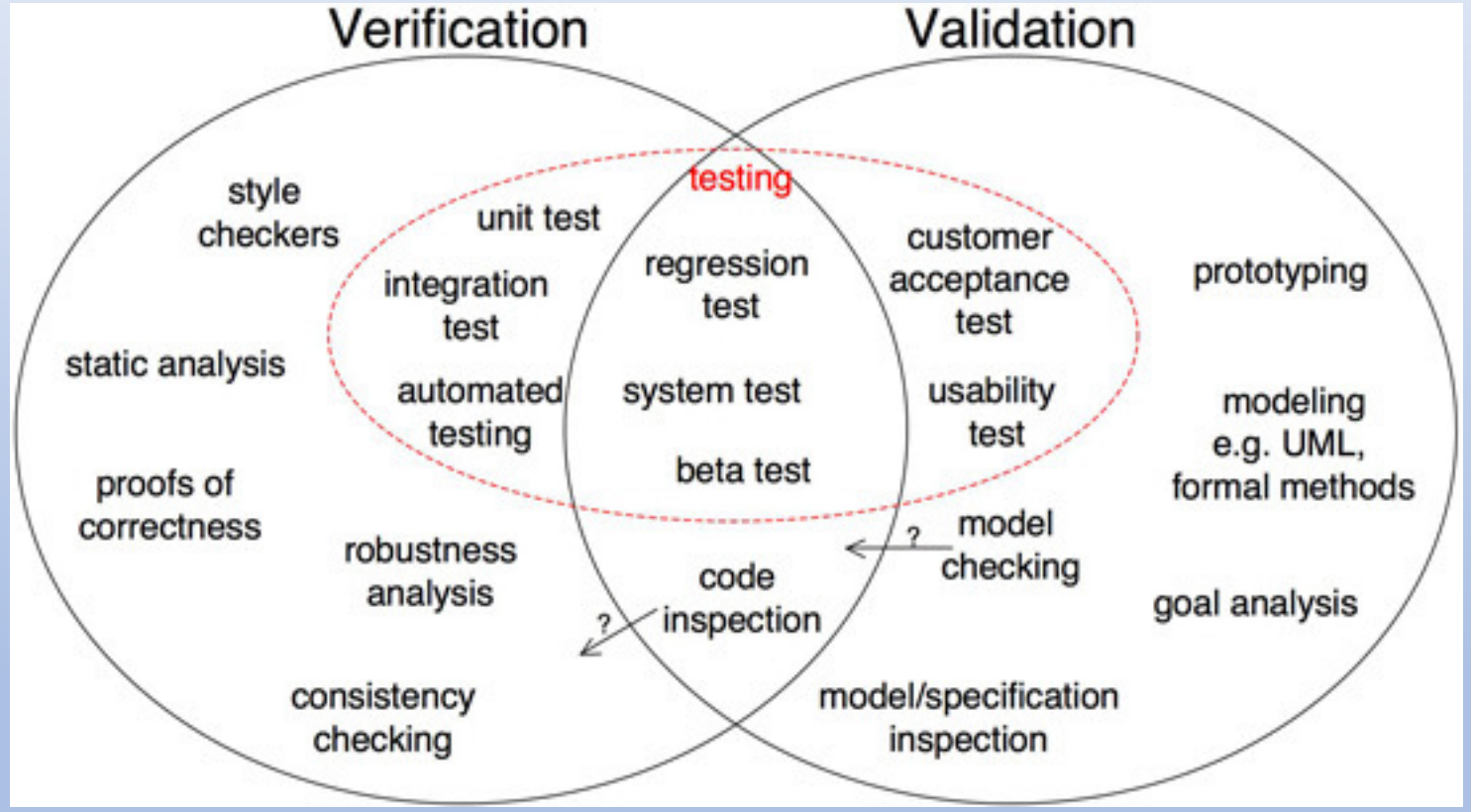
Software Engineering at Scale: Coding

- Coding for risk
- Coding for maintenance
- Coding for teams
- Coding user interfaces
- Coding for security
- Coding for privacy



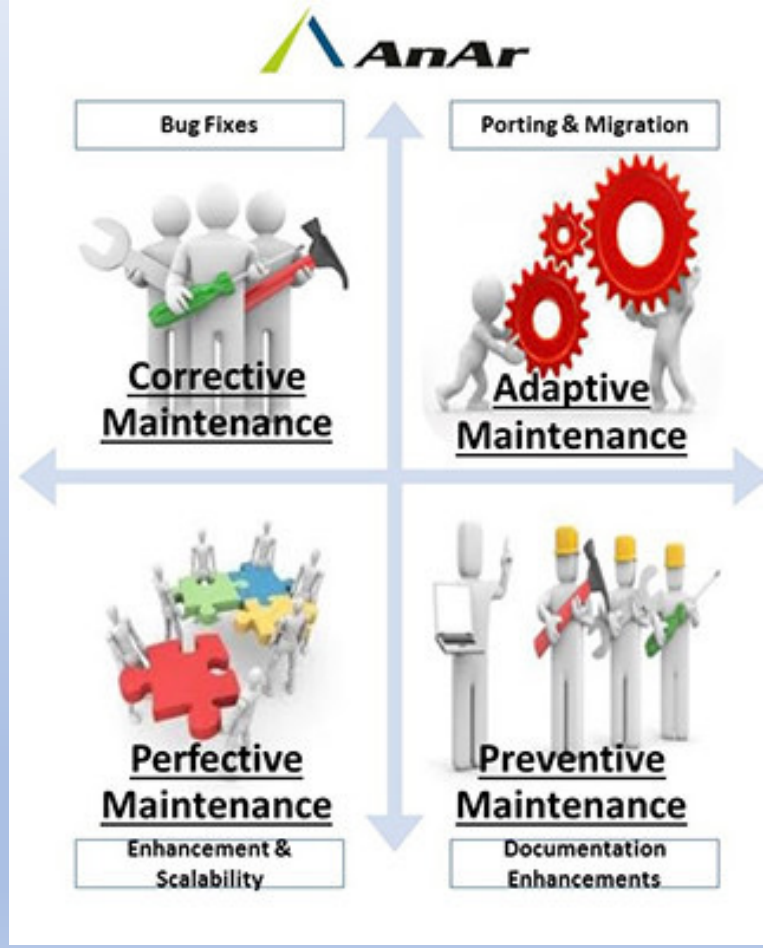
Software Engineering at Scale: Verification and Validation

- Debugging
- Testing Strategies
- Static Analysis
- Contracts
- Type Safety
- Model Checking



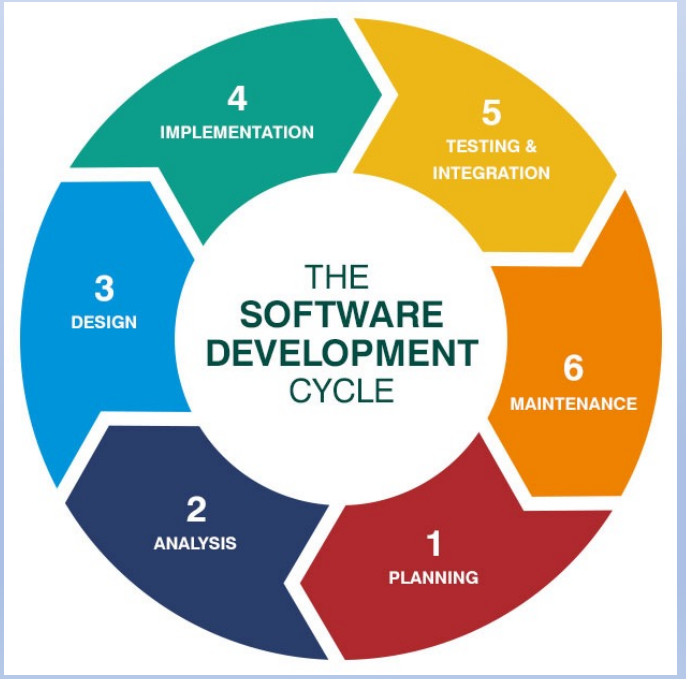
Software Engineering at Scale: Maintenance

- Keeping the system running
  - As hardware and languages evolve
  - Minimizing down time
- Fixing problems (bugs)
- Adding features (evolution)
- Making the system more robust
- Making the system more secure
- Making the system …

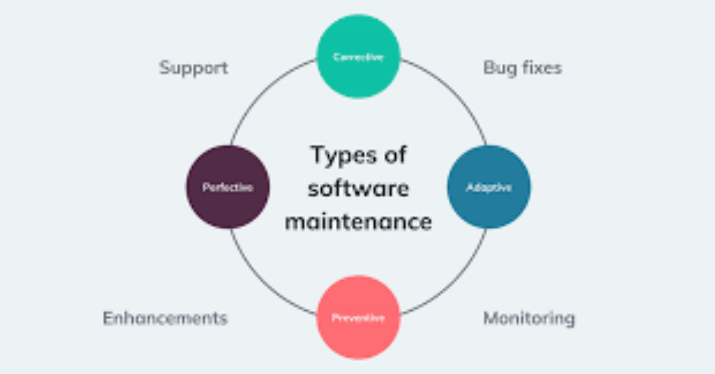


All Software Development is Maintenance

- Systems are too big to write all at once
  - Requirements will have changed by the time it is done
  - Systems evolve rather than being created
- Too much demand for change
  - Competing products, upgraded OS, platforms
- User's expectations are high
  - Expect software to be perfect
  - Expect software to evolve
  - Expect a better user interface
  - Expect new features and UIs continually

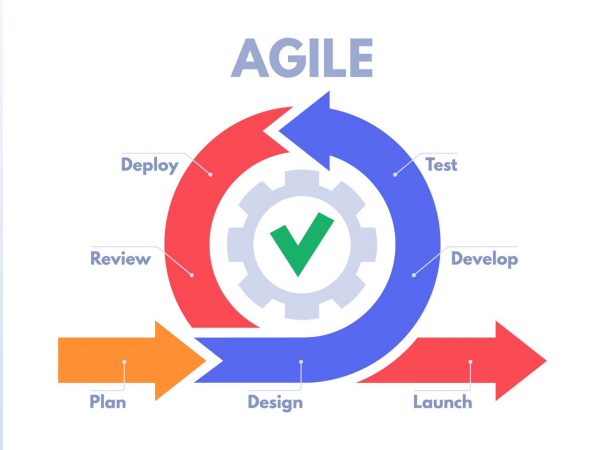


All Software Development is Maintenance



- Many software projects start with existing code
  - Convert that code to a new purpose
  - Extend that code for new functionality
  - Rewrite the code so it remains usable
- Most software is developed from an existing base
  - Templates or skeletons for various purposes
  - Based on prior systems
  - As extensions of existing systems
- Most software makes extensive use of libraries & frameworks
  - Don’t write something that has been written before
  - In-house, open-source, purchased
  - And existing systems

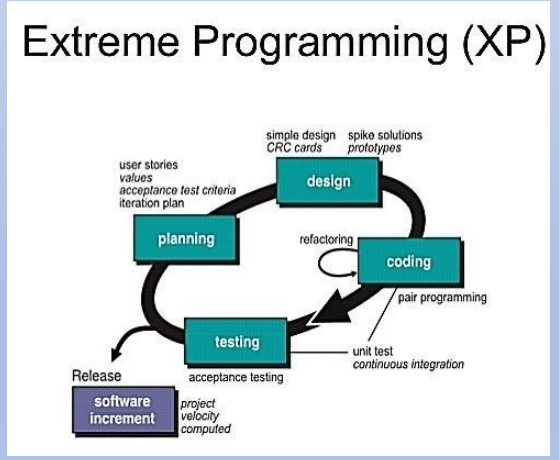
Agile Development



- Realize that most development is maintenance
  - Develop a software process to take this into account
- Goal is to provide an early working system
  - To get feedback from users to direct future development
  - To test the system to ensure it is robust
  - To provide satisfaction to the developers
  - And then extend that early system into a complete system
- Attempted to completely redo software development
  - But more evolutionary then revolutionary

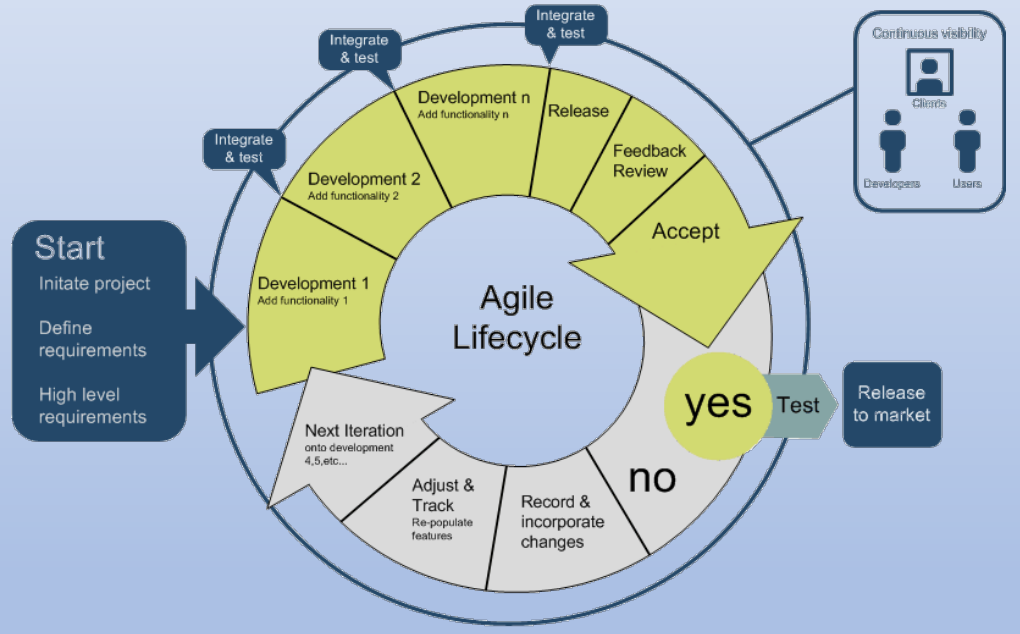
Extreme Programming

- Agile development derived from Extreme Programming (XP)
  - Develop in short bursts
  - Test-first development
  - Do specifications and requirements in terms of stories
  - Pair programming
  - Refactor the system as needed to add new features
- Lots of “new” ideas
  - That can be viewed separately

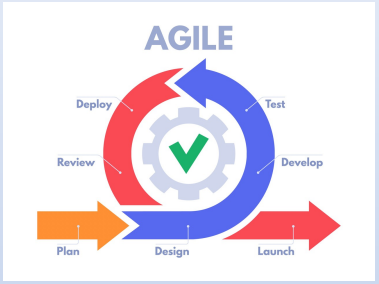


Agile Development Methodology

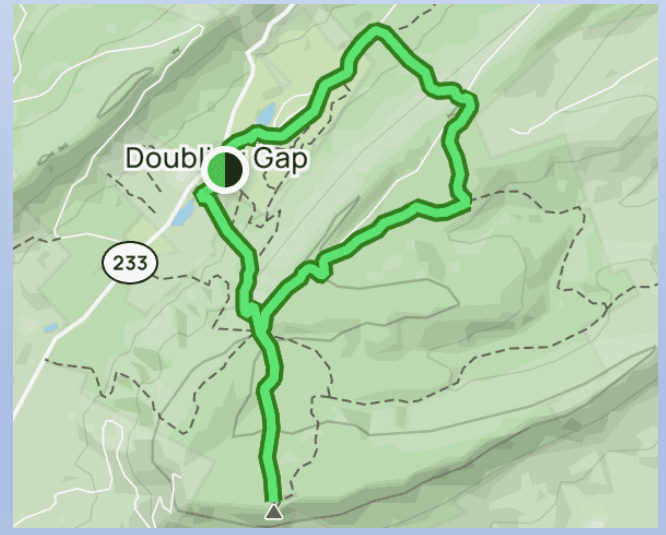
- Work in terms of sprints
  - Each sprint in 1-2 week of work
  - Each sprint adds features to the code
- Meet frequently to discuss progress
  - Weekly meeting after a sprint
  - SCRUM extension => daily meetings
  - Continuous integration
- Each sprint is its own cycle
  - Requirements, specifications, design, coding, testing



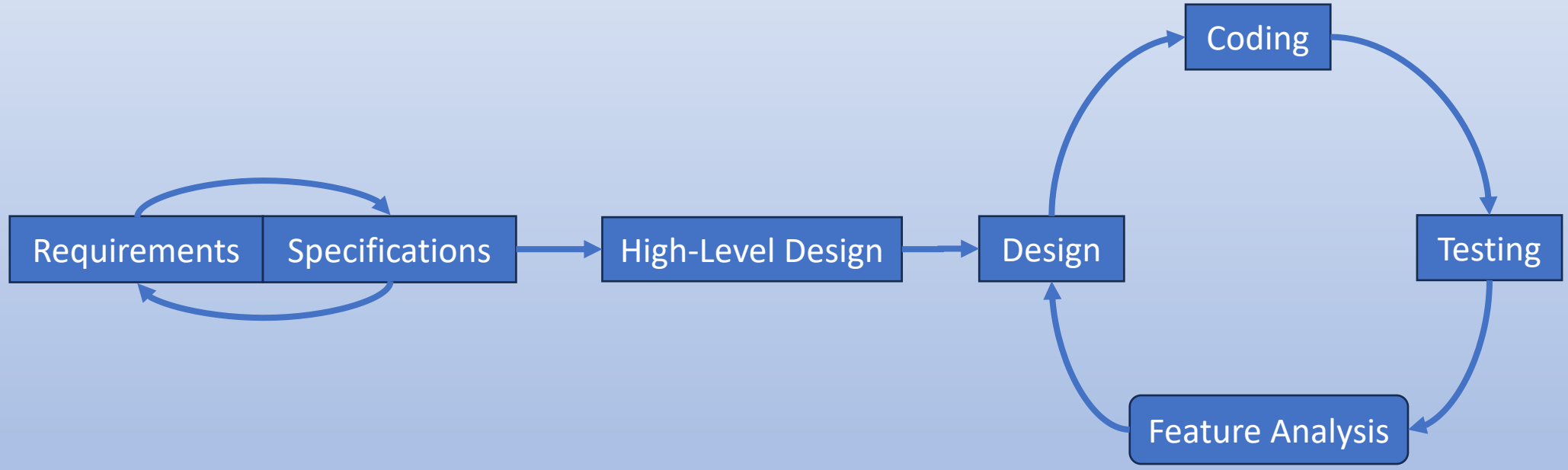
Lollipop (Balloon) Model of Development



- Agile development has its advantages
  - BUT it can require frequent refactoring of the overall system
  - AND it assumes the software is done at some point
- We prefer a hybrid or compromise model
  - Do complete requirements and specifications
  - Decide on a software architecture
  - Do a high-level design for that architecture
  - Then do agile development
  - But assume that development never stops
- We call this the lollipop or balloon model



Lollipop Development Model



Software Tools

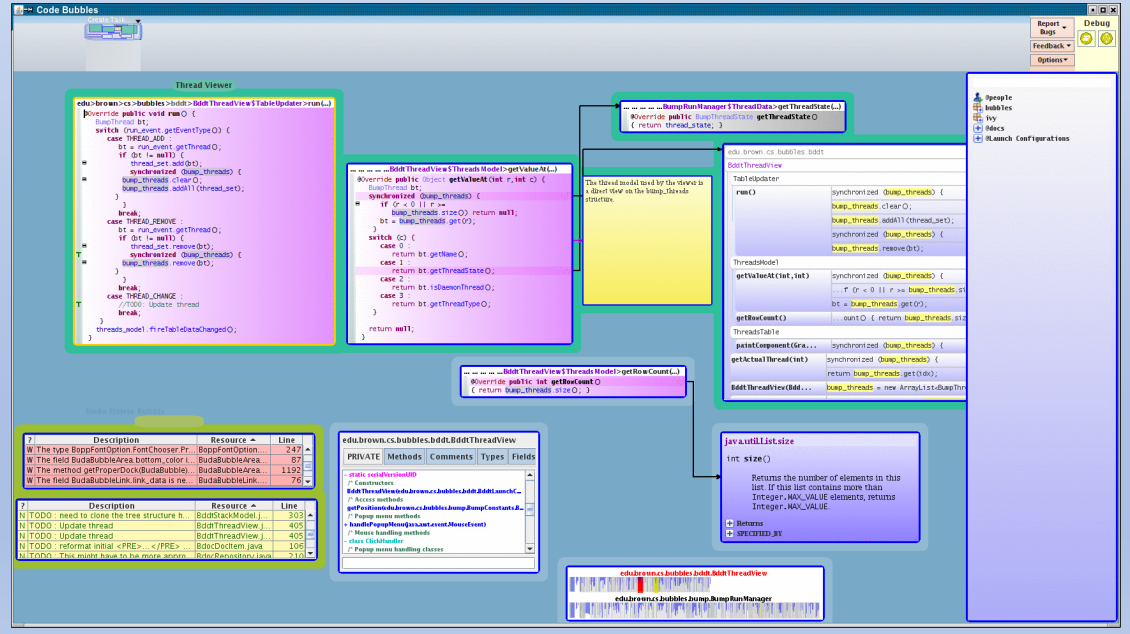
- Tools have been created to assist the developer
  - To support underlying techniques that work
  - This is a large part of what software engineering has done
- Tools have been developed for all phases and needs
  - With different degrees of success
- Wide and expanding variety of tools out there
  - Each developer/company has their favorites
  - We will cover a subset

Course Overview

- This course will cover much of software engineering
  - How to build large systems
  - Techniques that have been developed
- It will cover the various phases
- It will teach you how to create large systems
  - By example and by doing
- It will give you experience with software tools
  - And working in teams
- It will provide a foundation for research in the field

Tools We Will Use

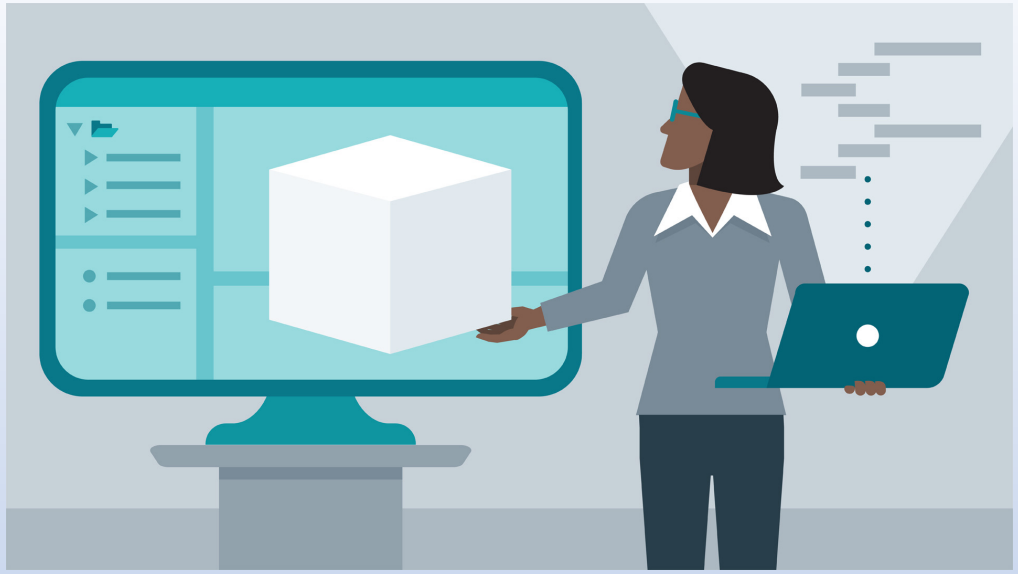
- IDE: Code Bubbles, VS Code, Eclipse, IntelliJ, …
- Teamwork: ZOOM, GIT, SLACK, GITHUB
- Team Organization: GITHUB
- Design: UML editor
- Coding: ant, maven, Gradle
- Testing: Junit
- Bug databases: GITHUB
- Deployment: Docker
- Visualization and understanding



QUESTION: Why are you taking this course?

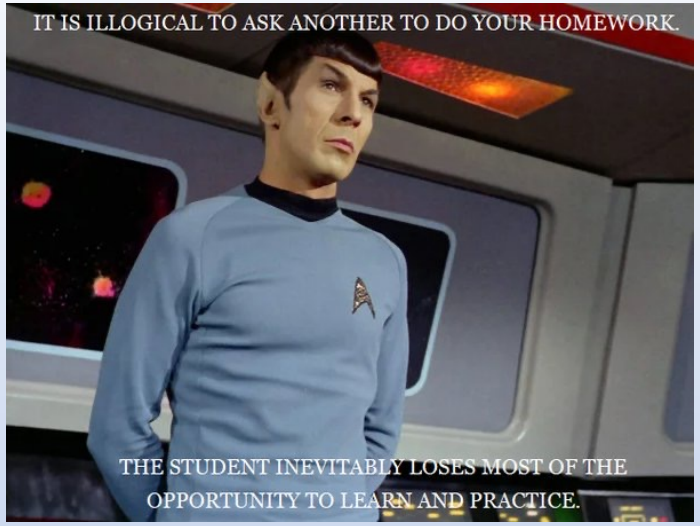
- To learn how to build large software systems
- To become a better programmer
- To become acquainted with modern software techniques
- To prepare to do research in software engineering
- Because it sounds like fun?

Course Mechanics



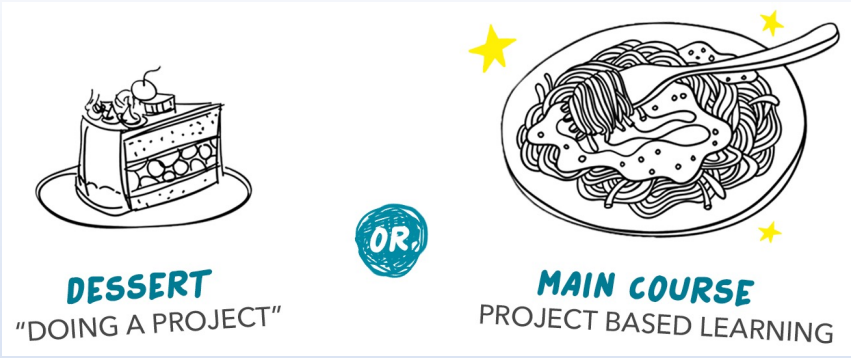
- Web site kept up to date
  - More or less
  - Canvas for turning in assignments
    - But little else
- Programming assignment
  - Handed in and possibly graded
  - Evolving: done multiple times (requirements will change)
  - Should be coded as if it were a large, long-lived system
  - Should be coded to evolve as the course progresses
  - Will tolerate some lateness
- Class participation
  - Group activities, feedback, questions
  - Will tolerate some absences
- Team project

Collaboration



- This course is highly collaborative
  - Project teams work together
  - Programming is better when done collaboratively
  - You don't have to invent everything from scratch
  - But you should still do your own work
- All collaboration must be identified and cited
  - Outside sources, libraries, AI, web pages
  - Collaborators
  - Citations should be part of the code
- Missing citations are a violation of academic integrity

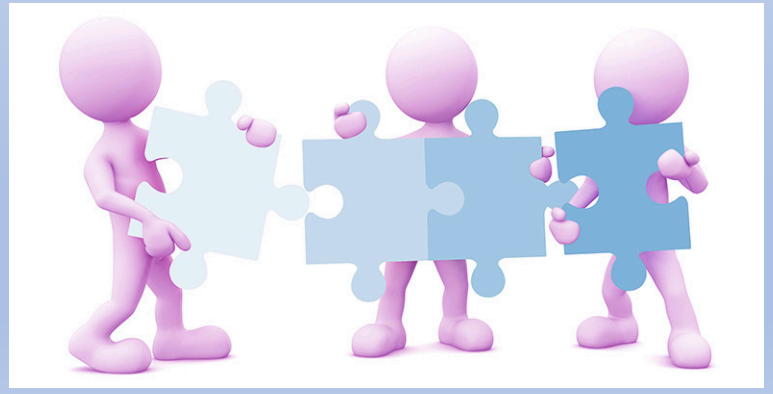
Course Project



- List of potential projects on the web site
  - Google doc for commenting, expressing interest, asking questions
    - Feel free to add comments, questions, express interest, …
  - Projects have different emphases
    - Development, testing, feasibility, augmenting existing systems, …
  - Software-engineering research oriented
- Soliciting new ideas through the weekend
  - Google form for submitting on the web site
    - What software would benefit society
    - What software would you or your research lab like to have
    - What software could be used to start a company
    - What software do you want to modify or adapt
  - Earlier is better (will probably require a back and forth)
    - And I'm not around 24/7 on weekends
- Form for project preferences
  - Available Monday, due Tuesday

Course Project

- Project team selection next week
  - I will choose project teams based on preferences
    - Probably won't satisfy everyone, but I will do my best
  - Each team responsible for project from start to finish
  - Initial team meeting in class next Thursday
  - Target team size 6-10
- Weekly deadlines for project
  - Include hand-ins, presentations, demo videos
  - Or just where the project should be
- Weekly meetings of project teams
  - I'll try to attend occasionally if during the day
- Should be coded as if it were a large, long-lived system
  - Grading based on code quality and extensibility



Research in Software Engineering

- Software engineering is an evolving field
  - Much has been done
  - Much remains to be done
- This course will offer starting points for potential research
  - New research directions
  - What is currently going on in the field
  - What I've been doing over the past 10 years
- Projects can be research oriented if you want
  - Most of the suggested projects have a research feel
  - But other projects are still welcome

Do You Belong Here?

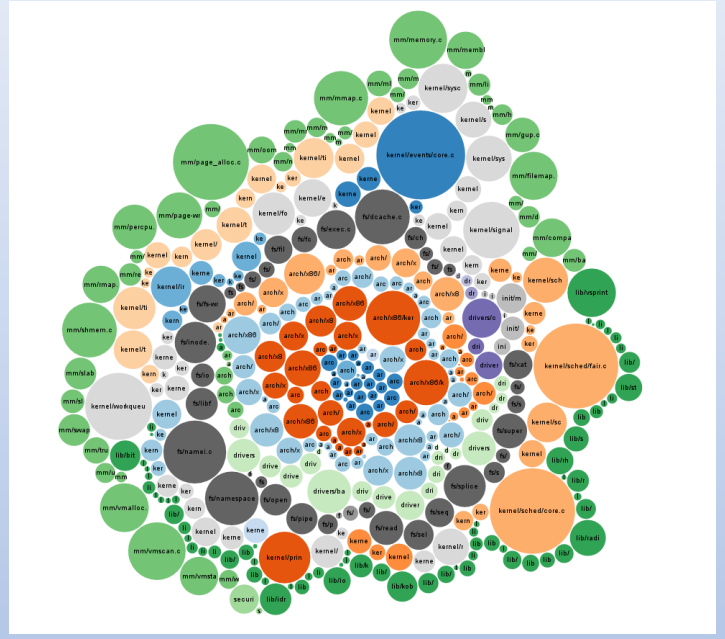
- If I asked you to write a system on your own that could:
  - Simulate 100,000 objects interacting via gravity
  - With an XML file giving the initial configuration
  - Use a complex data structure (Oct-Trees) [NlogN vs N^2]
  - Handling collisions; using a separate package to do 3D output
  - Using multiple threads
- Would it be easy for you?
  - Oh, give me a week or two
- Would you feel confident that you could do it in time?
  - You can identify the classes needed and define a high-level design quickly
  - Have everything working in 3-4 weeks
- Would you feel overwhelmed?

Alternative Problem (for Web developers)

- I have a large collection geolocated tweets from USA
  - Each with zip code, district, state, author, tweet, time, …
- Create an application to let users query & visualize these
  - Query by keyword, time range, …
  - Sample the results
  - Refine the queries to get better results
  - View the data on a map over time
  - Download CSV result for further analysis
- Would this be in your current skill set?
  - Front end and back end
  - How long would it take you?

Why Do We Consider These Problems

- We will cover high-level design
- We will cover detailed design only briefly
  - Selecting classes, methods, fields, …
  - How to code methods
  - Class design (OO patterns, …)
  - We assume you know this (32/134)
- If writing the detailed stuff is difficult
  - You should learn this first (take 32/134)
- We don’t want to drag down the project teams



Getting Help

- Office Hours
  - Monday 1-3
    - ZOOM: 781-445-5513
    - Exceptions should be noted in calendar
  - Thursday 11-12:30
    - CIT 403
  - Open-office (8-3 via Zoom or in my office)
    - Probably tell from my status (sign or web page)
- Note there are no TAs for this course
  - Use your project teams for project help
  - There will be minimal grading (this is a graduate course)

Questions on the Course

Exercise: Project discussion

- Project Discussion
  - Questions
    - What types of projects are you interested in
    - What would you like to get out of the project
    - What do you see as your role in the project
  - Let's go round the room and get feedback

Homework

- Download and install Code Bubbles
  - Need to install Eclipse first if not already there
- Run Code Bubbles on a project
  - Existing Eclipse workspace
  - Existing Java project
  - New code (hello world, nim)
- Relate experiences in a canvas hand-in
- Due 9/12 (one week)

Recap

- Project ideas due by the end of the day Saturday
  - Will be vetted, might ask follow-up questions
  - Projects should be approved by Sunday
  - Review project discussion document
- Project preference forms out on Monday
  - Due by the 5:00 pm on Tuesday
- Further Reading
  - Sommerville's Software Engineering
    - Look at Chapter one.
  - Textbook chapters 1 and 2