CPE 212 - Fundamentals of Software Engineering

Introduction to Software Engineering

** Unless otherwise noted, the following lecture notes are based upon the required course textbook, ***C++ Plus Data Structures, 5th edition,*** by Nell Dale.

ABOUT ME

Bachelors of Science in Computer Engineering at UAH

Masters of Software Engineering at Penn State

Research Engineer at Georgia Tech Research Institute

Class objective:
Introduction to structured programming using C++. Search and sort algorithms. Introduction to data structures and software engineering.

Grading Policy

Programming Assignments	20%
Exam #1	20%
Exam #2	20%
Quizzes	10%
Comprehensive Final	30%

Important Dates

Exam #1	Wednesday Feb 12, 2020
Exam #2	Wednesday Mar 25, 2020
Final Exam	Friday April 24, 2020

Project 01

C++ Review

Final Code Due

Friday January 17th by noon

Topics Covered

C++ Concepts	• Objects • Templates • Overloads • Stacks • Queues
Data Structures	• Lists • Graphs • Heaps • Trees
Algorithms	• Searching • Sorting

Objective:
Overview and introduction to Software Engineering.

What is High Quality Software?

- It works
  - Delivers all required Functionality, Usability, Compatibility, Reliability, Security, etc.
- It can be modified without excessive time or effort
  - Should be readily understandable, well documented, and modular
- It is reusable
  - Use proven techniques such as generic programming and object-oriented design to simplify reuse
- It is completed on time and within budget
  - In academia, late work may be penalized or refused altogether
  - In industry, late, incomplete, or incorrect work impacts product delivery schedules, customer confidence, product safety, and profits -- not to mention your reputation as an engineer

Software Engineering

- Software Engineering
  - Discipline which focuses on the design, production, and maintenance of software on time and within budget
    - Apply appropriate tools, methods, and theories to solve practical problems
    - Address both technical and project management issues
    - Work within constraints (organizational, financial, and schedule)
- History of software engineering
  - Term originated in 1967 as a result of a NATO study group investigating the ‘Software Crisis’
  - Increased hardware capabilities permitted development of more complex computer programs
  - Ad hoc development methods were proving inadequate with respect to schedule, budget, and product quality

[Adapted from Ian Sommerville, Software Engineering, 6th Edition, 2001]

Software Project Success Rates

- Successful - Completed on time, within budget, & with all specified features
- Challenged - Completed but late, over budget, & missing some features
- Failures - Cancelled before completion

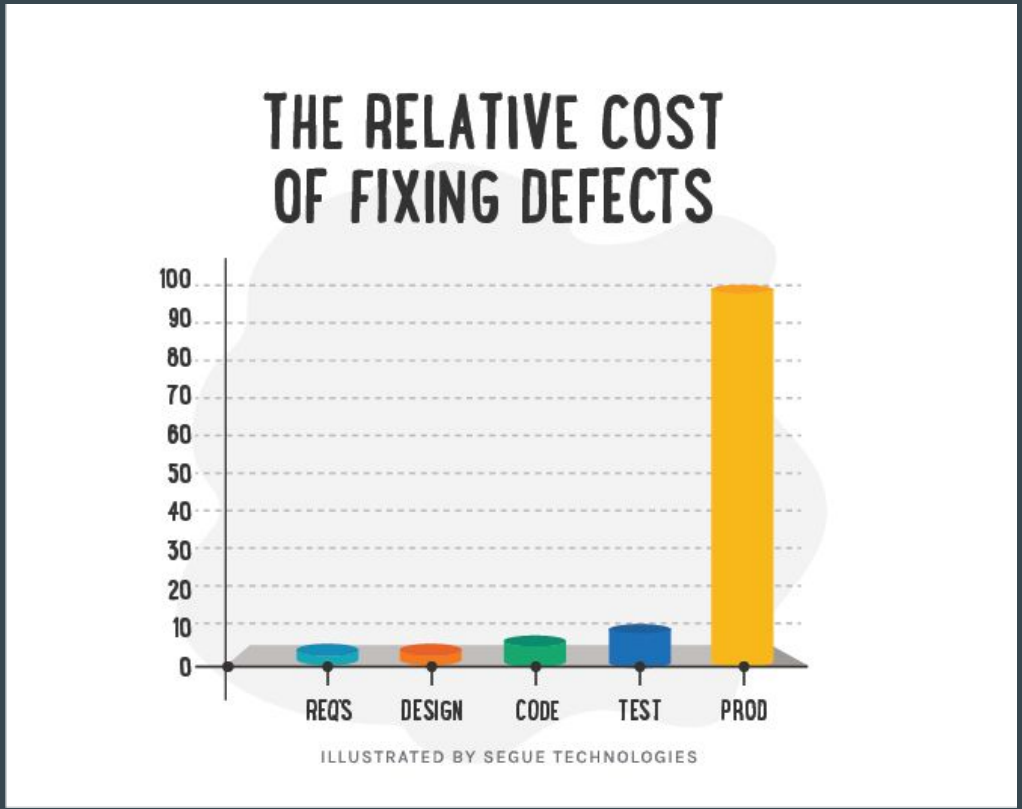
Category	1994	1996	1998	2000	2004	2009
Successful	16%	27%	26%	28%	29%	32%
Challenged	53%	33%	46%	49%	53%	44%
Failures	31%	40%	28%	23%	18%	24%

Research by Standish Group, www.standishgroup.com

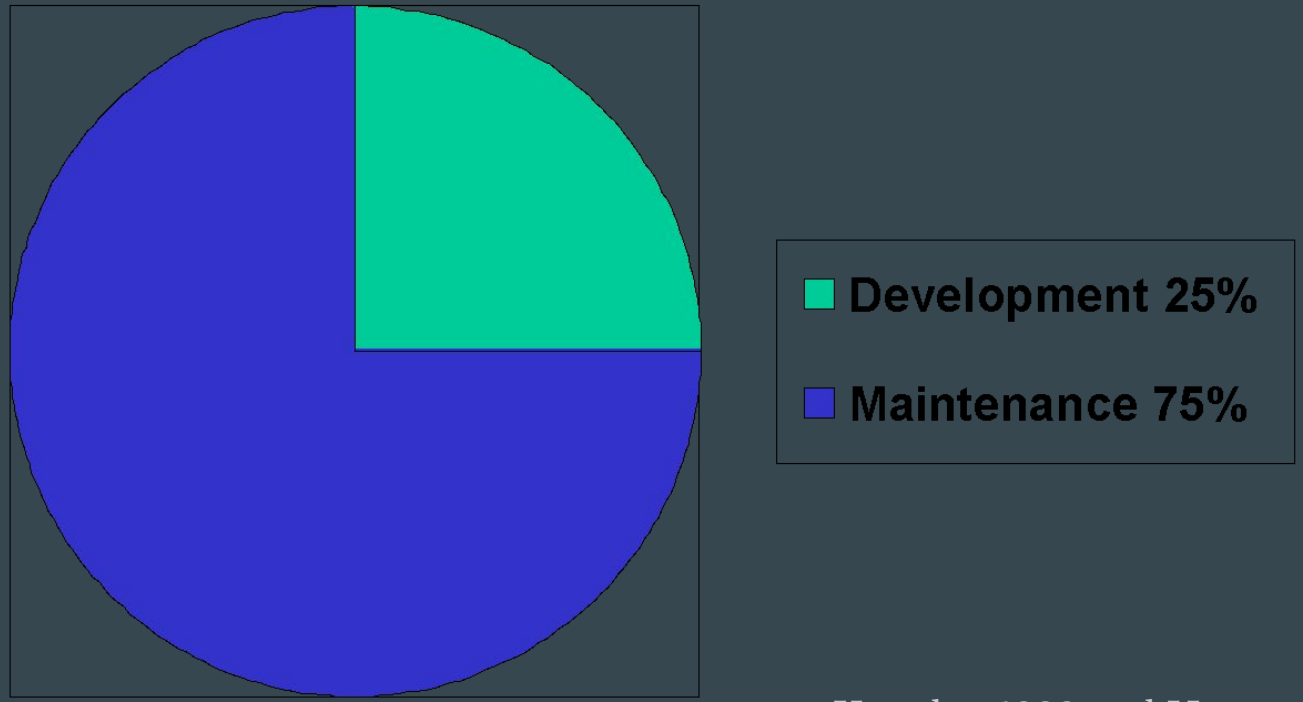
Typical Defect Density

- Experienced software engineers
  - 1 defect for every 10 Lines Of Code (LOC) [Winning with Software: An Executive Strategy (2002) Watts S. Humphrey]
- A 300,000 LOC program will have approximately 30,000 defects that must be identified and removed!!

Relative Defect Removal Costs



Development vs Maintenance Costs



Yourdan 1992 and Hatton 1998

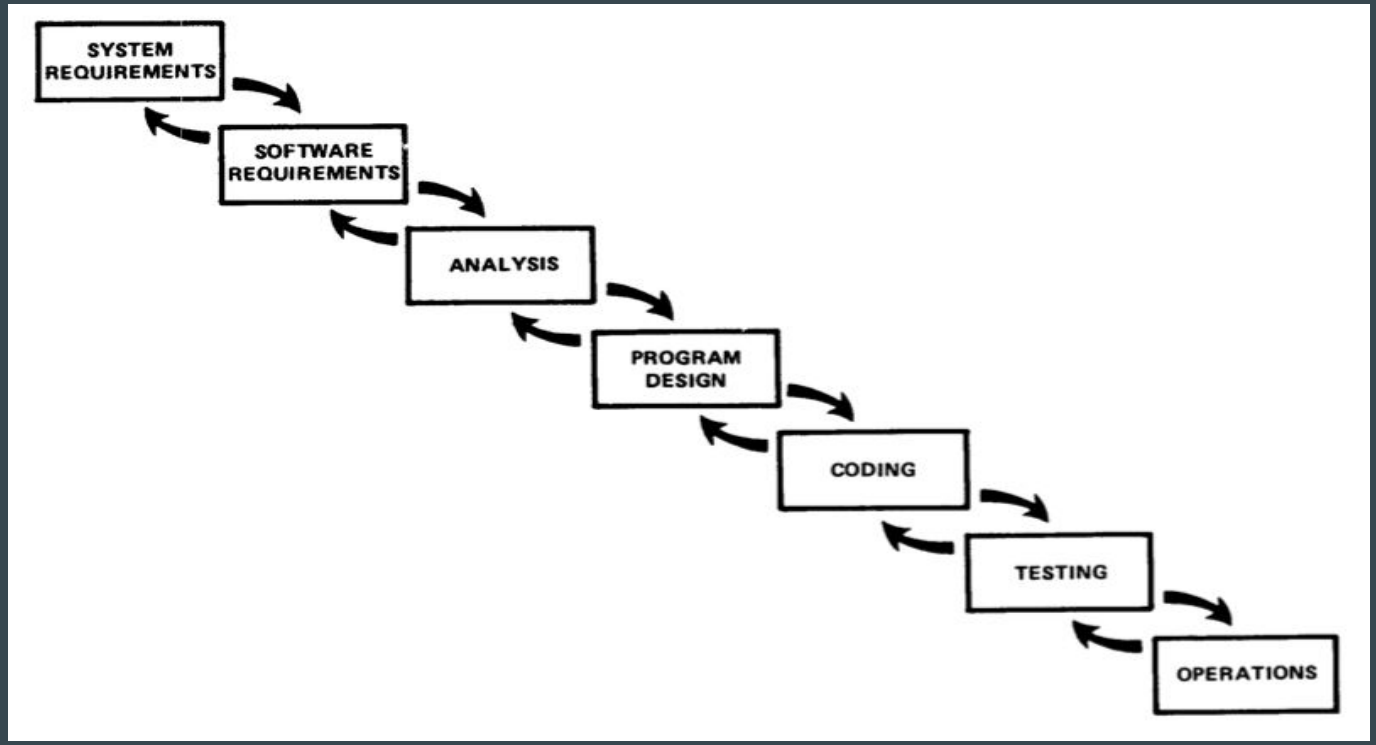
Software Development Activities

Activity	Description
Requirements	Prototype and/or high-level description of product
Specification	Develop detailed description of functional requirements and non-functional requirements (constraints).
Design	Architectural design (high-level design) and detailed design (low-level design)
Implementation	Translation of the design into program code
Testing and Verification	Detecting and fixing errors, and demonstrating the correctness of the program
Postdelivery Maintenance	Correct defects reported by users, modify or enhance functionality

Software Process

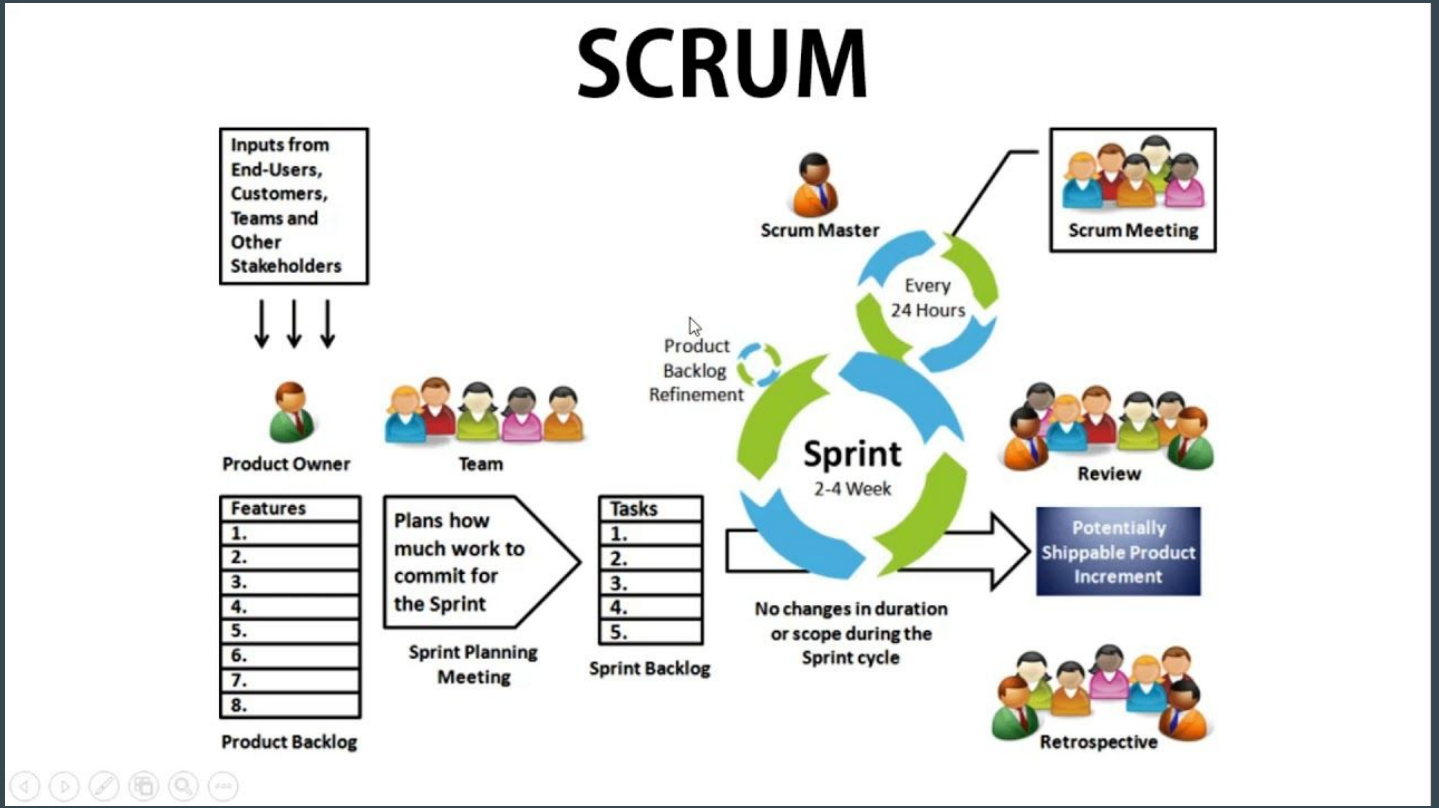
- Software Process
  - A standard sequence of steps for the development or maintenance of software
    - May include use of specific tools or techniques
    - Metrics used to measure process effectiveness
- Waterfall Process
  - Development activities conducted in order previously presented
  - Each activity produces a document or product that is the input for the next activity
- SCRUM Process
  - Focus on individuals and interactions over processes and tools
  - Working software over comprehensive documentation

Waterfall Process



Winston Royce, "Managing the Development of Large Software Systems",
Technical Papers of Western Electric Show and Convention (WesCon), August 25-28, 1970.

SCRUM (Agile Process)



Program Specification

- Start with the problem description (initial requirements)
- Ask questions!!
  - What are the inputs, outputs, and key operations that must be performed?
  - What assumptions are you allowed to make?
  - What should the program do if an error occurs?
  - Are there any performance constraints?
  - Etc.
- Describe the interactions between the user and the software (scenarios)
  - Normal interactions and their variations
  - Exception scenarios (desired error handling)
- How detailed should the specifications be?
  - Detailed enough that a programmer not familiar with the project can follow them to produce the product (why?)

Program Design

- Abstraction
  - Model of a complex system including only key details
- Information Hiding
  - Hiding data and function details to limit access to implementation details
- Step-Wise Refinement
  - Iterative, incremental approach to problem solving
  - Top-Down: defer details as long as possible
  - Bottom-Up: start with details and work towards big-picture
  - Round-Trip Gestalt: Object-Oriented Design

Classical vs. Object-Oriented Design

- Classical Design
  - Focused on actions (functions or operations) to be performed
  - Functional decomposition
- Object-Oriented Design
  - Objects model tangible items or events in the system
    - Attributes
    - Methods
  - Equal emphasis on data and operations
  - Iterative and incremental development
    - Identify candidate classes
    - Define external relationships and interfaces
    - Address internal details
    - Repeat and refine until complete

Software Correctness

- Testing
  - Executing the program using various data designed to discover errors
- Debugging
  - Process of removing known errors
- Test Plan
  - Document summarizing the tests planned for a program or module, their purposes, inputs, expected outputs, and criteria for success
- Driver
  - Program written specifically to directly test a subprogram or module used in bottom-up integration
- Stub
  - Dummy code used as a stand-in to facilitate top-down integration and testing

Software Correctness Proofs

- Assertion
  - A logical proposition that can be true or false
- Precondition
  - Assertions that must be true upon entry into a function in order for the postconditions to be guaranteed
- Postcondition
  - Assertions which state what results are expected at the exit of a function assuming the preconditions are true

Testing Hierarchy

- Developer Testing
  - Informal unit testing performed by developer
- Unit Testing
  - Formal testing of each module
- Integration Testing
  - Systematic, incremental integration and testing of individually tested modules instead of “big bang” approach
- Acceptance Testing
  - Process of testing the system in its real environment with real data
- Regression Testing
  - Re-execution of program tests after modifications have been made to ensure that the program still works correctly

Testing Strategies

- Black-Box Testing
  - No knowledge of program internals
  - Tests derived from specifications
  - Apply inputs, examine outputs
  - Data coverage
- Clear-Box Testing
  - Internal structure of code known
  - Select inputs to exercise each statement and branch path
  - Code coverage
- An effective, efficient Test Plan typically combines both Black-Box and Clear-Box tests

Metric-Based Testing

- Measurable factors used to evaluate thoroughness of testing

Question?

What are some metrics that could gauge the thoroughness of testing?

Verification vs. Validation

- Program Verification
  - Has the current workflow been completed correctly?
  - “Are you building the program right?”
- Program Validation
  - Does program solve the actual problem?
  - “Are you building the right program?”