272: Software Engineering

Instructor: Tevfik Bultan

Lecture 0: Introduction

Computer Science vs. Software Engineering

- Degree: Computer Science

  UCSB CS website: *This program **introduces students to core concepts and cutting-edge topics in computer science**. The program provides students with **hands-on experience and a depth of understanding of computer theory, systems, and applications** that prepares them for successful careers in computer science and to participate in the next-generation of technological advances.*

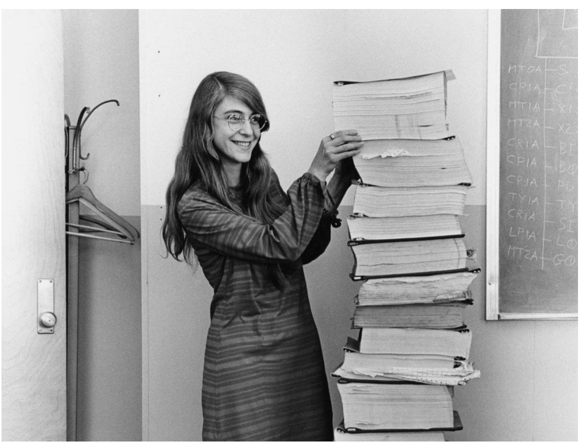
- Job: Software Engineer

  ChatGPT: *A software engineer is a **professional who applies engineering principles to the design, development, maintenance, testing, and evaluation of software** and systems that make computers or anything containing software, such as chips, work. They use various programming languages, tools, and techniques to create software solutions that meet the needs of users or clients. Software engineers play a crucial role in the technology industry, contributing to the development of software applications, operating systems, databases, and more.*

Software engineering is 55 years old!

- In 1968 a seminal NATO Conference was held in Garmisch, Germany

Attendees of the 1968 Garmisch conference



Margaret Hamilton Director of Software Development for the NASA's Apollo mission

Purpose: to look for a solution to ***software crisis***

– 50 distinguished computer scientists, programmers and industry leaders got together to ***look for a solution to the difficulties in building large software systems***

– Considered to be the birth of "***software engineering***" as a research area

Software’s Chronic Crisis

Large software systems often:

- Do not provide the desired functionality
- Take too long to build
- Cost too much to build
- Require too much resources (time, space) to run
- Cannot evolve to meet changing needs

Software engineering as a remedy: *a systematic, disciplined, quantifiable approach to the production and maintenance of software.*

Software's chronic crisis

- A quarter century (1994) after the Garmisch conference, an article in Scientific American declared:

> **Software's Chronic Crisis**
>
> TRENDS IN COMPUTING by W. Wayt Gibbs, staff writer.
> Copyright Scientific American; September 1994; Page 86
> *Despite 50 years of progress, the software industry remains years-perhaps decades-short of the mature engineering discipline needed to meet the demands of an information-age society*

Software's chronic crisis

- Another quarter century later:

- This is a photo of the navigation system of my car
  - ***It crashes and reboots while I am driving!***

Software's chronic crisis

We are still looking for a:

systematic, disciplined, quantifiable approach to the production and maintenance of software

which ensures:

safety, dependability, security, reliability, availability, usability, efficiency, scalability, and maintainability

of software systems.

Software Failures

- There is a long list of failed software projects and software failures
- You can find a list of famous software bugs at:
  http://www5.in.tum.de/~huckle/bugse.html
- I will talk about two famous and interesting software bugs

Ariane 5 Failure

- A software bug caused European Space Agency's Ariane 5 rocket to crash 40 seconds into its first flight (***cost: half billion dollars***)

- The bug was caused because of a software component that was being reused from Ariane 4
- A software exception occurred during execution of a data conversion from 64-bit floating point to 16-bit signed integer value
  - The value was larger than 32,767, the largest integer storable in a 16 bit signed integer, and thus the conversion failed and an exception was raised by the program
- When the primary computer system failed due to this problem, the secondary system started running.
  - The secondary system was running the same software, so it failed too!

Ariane 5 Failure

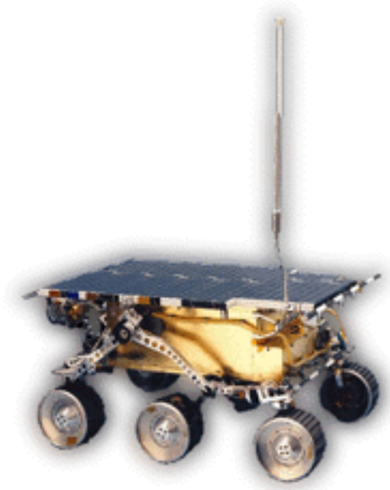
- The programmers for Ariane 4 had decided that this particular velocity figure would never be large enough to raise this exception.
  - Ariane 5 was a faster rocket than Ariane 4!
- The calculation containing the bug actually served no purpose once the rocket was in the air.
  - Engineers chose long ago, in an earlier version of the Ariane rocket, to leave this function running for the first 40 seconds of flight to make it easy to restart the system in the event of a brief hold in the countdown.

- You can read the report of Ariane 5 failure at:

  http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html

Mars Pathfinder

- A few days into its mission, NASA's Mars Pathfinder computer system started rebooting itself
  - Cause: Priority inversion during preemptive priority scheduling of threads



- Priority inversion occurs when
  - a thread that has higher priority is waiting for a resource held by thread with a lower priority
- Pathfinder contained a data bus shared among multiple threads and protected by a mutex lock
- Two threads that accessed the data bus were: a high-priority bus management thread and a low-priority meteorological data gathering thread
- Yet another thread with medium-priority was a long running communications thread (which did not access the data bus)

Mars Pathfinder

- The scenario that caused the reboot was:
  - The meteorological data gathering thread accesses the bus and obtains the mutex lock
  - While the meteorological data gathering thread is accessing the bus, an interrupt causes the high-priority bus management thread to be scheduled
  - Bus management thread tries to access the bus and blocks on the mutex lock
  - Scheduler starts running the meteoroloVgical thread again
  - Before the meteorological thread finishes its task yet another interrupt occurs and the medium-priority (and long running) communications thread gets scheduled
  - At this point high-priority bus management thread is waiting for the low-priority meteorological data gathering thread, and the low-priority meteorological data gathering thread is waiting for the medium-priority communications thread
  - Since communications thread had long-running tasks, after a while a watchdog timer would go off and notice that the high-priority bus management thread has not been executed for some time and conclude that something was wrong and reboot the system

Disastrous consequences: Security

- Facebook data leak
- Microsoft software misconfiguration
- SolarWinds hack

Newsweek
TECH & SCIENCE
October 4, 2021

1.5 Billion Facebook Users' Personal Information Allegedly Posted for Sale

CNN BUSINESS
August 24, 2021

Data leak exposes tens of millions of private records from corporations and government agencies

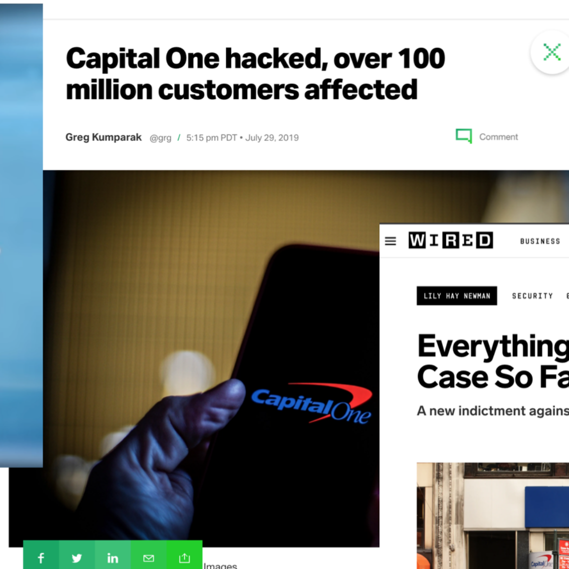
CNN BUSINESS
December 19, 2020

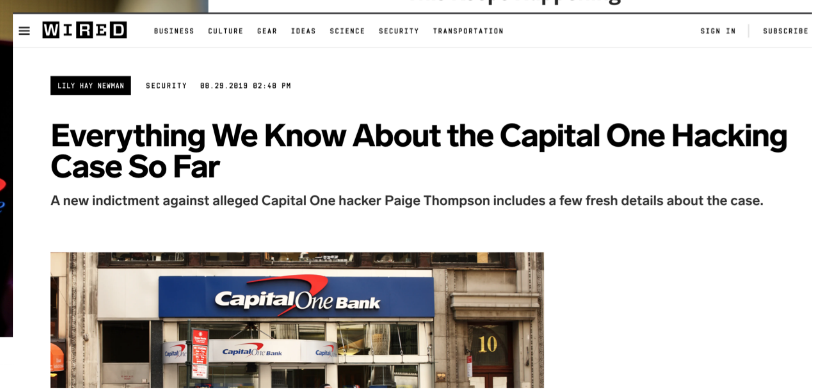
Massive SolarWinds hack has big businesses on high alert

Access Control Data Breaches

User must manually write policies

- Easy to write incorrect/overly permissive polic
- Leads to unintended access to secure data





Side-channels

Meltdown

Reading kernel memory from user space

Spectre

Exploiting speculative execution

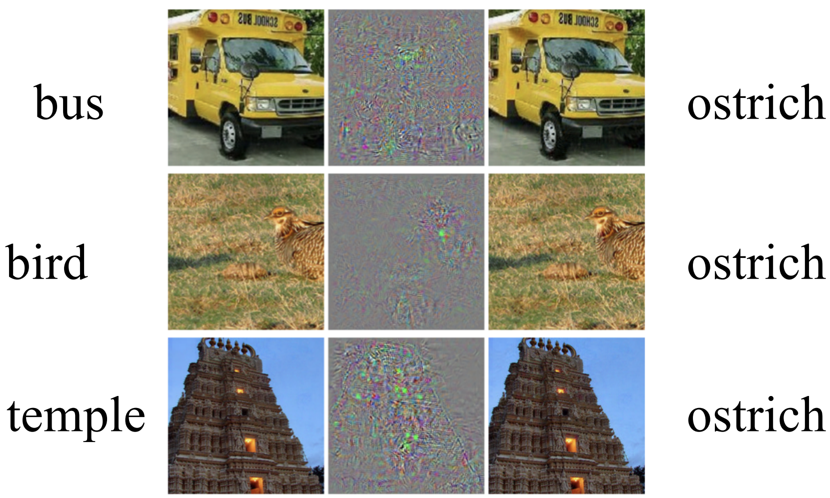
Confidentiality and side-channels

- ***Confidentiality***: A program that manipulates secret information should not reveal/leak that information

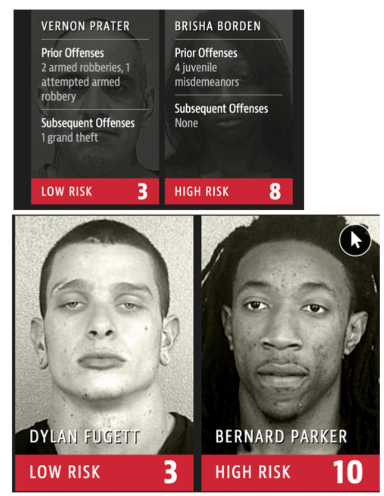
- ***Side-channel attacks*** recover secret information from programs by observing non-functional characteristics
  - execution time, memory usage, memory accesses, network communication, energy consumption, etc.

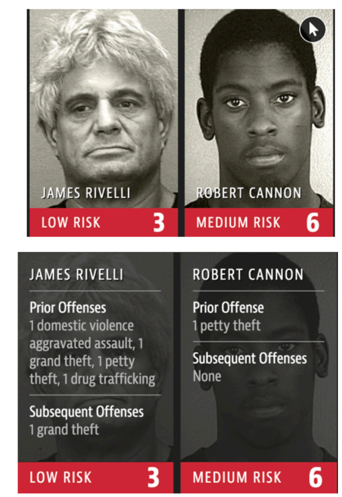
New problems!

robustness failure
in image classification



fairness failure
in recidivism risk assessment





Software’s Chronic Crisis

- These are not isolated incidents:
  - An IBM survey of 24 companies developing distributed systems:
    - 55% of the projects cost more than expected
    - 68% overran their schedules
    - 88% had to be substantially redesigned

Software's Chronic Crisis

- Software product size is increasing exponentially
  - faster, smaller, cheaper hardware
- Software is everywhere: from TV sets to cell-phones to watches to cars
- Marc Andreessen: "Software is Eating the World"
- Software is in safety-critical systems
  - cars, airplanes, nuclear-power plants
- We are seeing more of
  - distributed systems
  - embedded systems
  - real-time systems
    - These kinds of systems are harder to build
- Software requirements change
  - software evolves rather than being built
- Novel problems related to security arms race and increasing use of AI

Summary

- Software's chronic crisis: Development of large software systems is a challenging task
  - Large software systems often: Do not provide the desired functionality; Take too long to build; Cost too much to build Require too much resources (time, space) to run; Cannot evolve to meet changing needs

- Software engineering focuses on addressing challenges that arise in development of large software systems using a systematic, disciplined, quantifiable approach

No Silver Bullet

- In 1987, in an article titled:
  "No Silver Bullet: Essence and Accidents of Software Engineering"
  Frederick P. Brooks made the argument that there is no silver bullet that can kill the werewolf software projects

- Following Brooks, let's philosophize about software a little bit

Essence vs. Accident

- Essence vs. accident in software development
  - We can get rid of accidental difficulties in developing software
  - Getting rid of these accidental difficulties will increase productivity

- For example using a high level programming language instead of assembly language programming
  - The difficulty we remove by replacing assembly language with a high-level programming language is not an essential difficulty of software development,
    - It is an accidental difficulty brought by inadequacy of assembly language for programming

Essence vs. Accident

- Essence vs. accident in software development
  - Brooks argues that software development is inherently difficult
    - "*The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms and invocations of functions.* ***This essence is abstract in that such a conceptual construct is the same under many different representations****. ... The hard part of building software is the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation*."
    - Even if we remove all accidental difficulties which arise during the translation of this conceptual construct (design) to a representation (implementation), still at its essence software development is difficult

Inherent Difficulties in Software

- Software has the following properties in its *essence*:
  - Complexity
  - Conformity
  - Changeability
  - Invisibility
- Since these properties are not *accidental* representing software in different forms do not effect them

- The moral of the story:
  - Do not raise your hopes up for a silver bullet, there may never be a single innovation that can transform software development as electronics, transistors, integrated-circuits and VLSI transformed computer hardware

Complexity

- Software systems do not have regular structures, there are no identical parts

- Identical computations or data structures are not repeated in software

- In contrast, there is a lot of regularity in hardware
  - for example, a memory chip repeats the same basic structure millions of times

Complexity

- Software systems have a very high number of discrete states
  - Infinite if the memory is not bounded

- Elements of software interact in a non-linear fashion

- Complexity of the software increases much worse than linearly with its size

Complexity

- Consider a plane that is going into a wind-tunnel for aerodynamics tests
  - During that test it does not matter what is the fabric used for the seats of the plane, it does not even matter if the plane has seats at all!
  - Only thing that matters is the outside shape of the plane
  - This is a great abstraction provided by the physical laws and it helps mechanical engineers a great deal when they are designing planes
- Such abstractions are available in any engineering discipline that deals with real world entities
- Unfortunately, software engineers do not have the luxury of using such abstractions which follow from physical laws
  - Software engineers have to develop the abstractions themselves (without any help from the physical laws)

Conformity

- Software has to conform to its environment
  - Software conforms to hardware interfaces not the other way around
- Most of the time software systems have to interface with an existing system
- Even for a new system, the perception is that, it is easier to make software interfaces conform to other parts of the system

Changeability

- Software is easy to change, unlike hardware
- Once an Intel processor goes to the production line, the cost of replacing it is enormous (the Pentium FDIV bug in 90s cost Intel half billion dollars)
- If a Microsoft or Apple product has a bug, the cost of replacing it is negligible.
  - Just ask users to update their software

Changeability is not an Advantage

- Although it sounds like, finally, software has an advantage over hardware, the effect of changeability is that there is more pressure on changing the software

- Since software is easy to change software gets changed frequently and deviates from the initial design
  - adding new features
  - supporting new hardware

Changeability

- Conformity and Changeability are two of the reasons why reusability is not very successful in software systems

- Conformity and Changeability make it difficult to develop component based software, components keep changing

Invisibility

- Software is invisible and un-visualizable
- Complete views can be incomprehensible
- Partial views can be misleading
- All views can be helpful

- Geometric abstractions are very useful in other engineering disciplines
  - Floor plan of a building helps both the architect and the client to understand and evaluate a building
- Software does not exist in physical space and, hence, does not have an inherent geometric representation

Invisibility

- Visualization tools for computer aided design are very helpful to computer engineers
  - Software tools that show the layout of the circuit (which has a two-dimensional geometric shape) makes it much easier to design a chip

- Visualization tools for software are not as successful
  - There is nothing physical to visualize, it is hard to *see* an abstract concept
  - There is no physical distance among software components that can be used in mapping software to a visual representation

Summary

- According to Brooks, there are essential difficulties in software development which prevents significant improvements in software engineering:
  - Complexity; Conformity; Changeability; Invisibility

- He argues that an order of magnitude improvement in software productivity cannot be achieved using a single technology due to these essential difficulties

How Do We Build Software?

Let' s look at an example:

- Sometime ago I asked our IT folks if they can do the following:
  - Every year all the PhD students in our department fill out a progress report that is evaluated by the graduate advisors. We want to make this online.
- After I told this to our IT manager, he said “OK, let's have a meeting so that you can explain us the functionality you want.”
- We scheduled a meeting and at the meeting we went over
  - The questions that should be in the progress report
  - Type of answers for each question (is it a text field, a date, a number, etc?)
  - What type of users will access this system (students, faculty, staff)?
  - What will be the functionality available to each user?

Requirements Analysis and Specification

- This meeting where we discussed the functionality, input and output formats, types of users, etc. is called requirements analysis
  – During requirements analysis software developers try to figure out the functionality required by the client

- After the requirements analysis all these issues can be clarified as a set of **Requirements specifications**
  – Maybe the IT folks who attended the requirements analysis meeting are not the ones who will develop the software, so the software developers will need a specification of what they are supposed to build.

- Writing precise requirements specifications can be very challenging:
  – Formal (mathematical) specifications are precise, but hard to read and write
  – English is easy to read and write, but ambiguous

Design

- After figuring out the requirements specifications, we have to build the software
- In our example, I assume that the IT folks are going to talk about the structure of this application first.
  - There will be a backend database, the users will first login using an authorization module, etc.
- Deciding on how to modularize the software is part of the **Architectural Design.**
  - It is helpful (most of the time necessary, since one may be working in a team) to document the design architecture (i.e., modules and their interfaces) before starting the implementation.
- After figuring out the modules, the next step is to figure out how to build those modules.
- **Detailed Design** involves writing a detailed description of the processing that will be done in each module before implementing it.
  - Generally written in some structured pseudo-code.

Implementation and Testing

- Finally, the IT folks are going to pick an implementation language (PHP, python, Java, etc.) and start writing code.
- This is the **Implementation** phase:
  – Implement the modules defined by the architectural design and the detailed design.

- After the implementation is finished the IT folks will need to check if the software does what it is supposed to do.
- Use a set of inputs to **Test** the program
  – When are they done with testing?
  – Can they test parts of the program in isolation?

Maintenance

- After they finished the implementation, tested it, fixed all the bugs, are they done?

- No, I (client) may say, “I would like to add a new question to the PhD progress report” or “I found a bug when I was using it” or “You know, it would be nice if we can also do the MS progress reports online” etc.
  - The difficulty of changing the program may depend on how we designed and implemented it:
    - Are the module interfaces in the program well defined? Is changing one part of the code effect all the other parts?

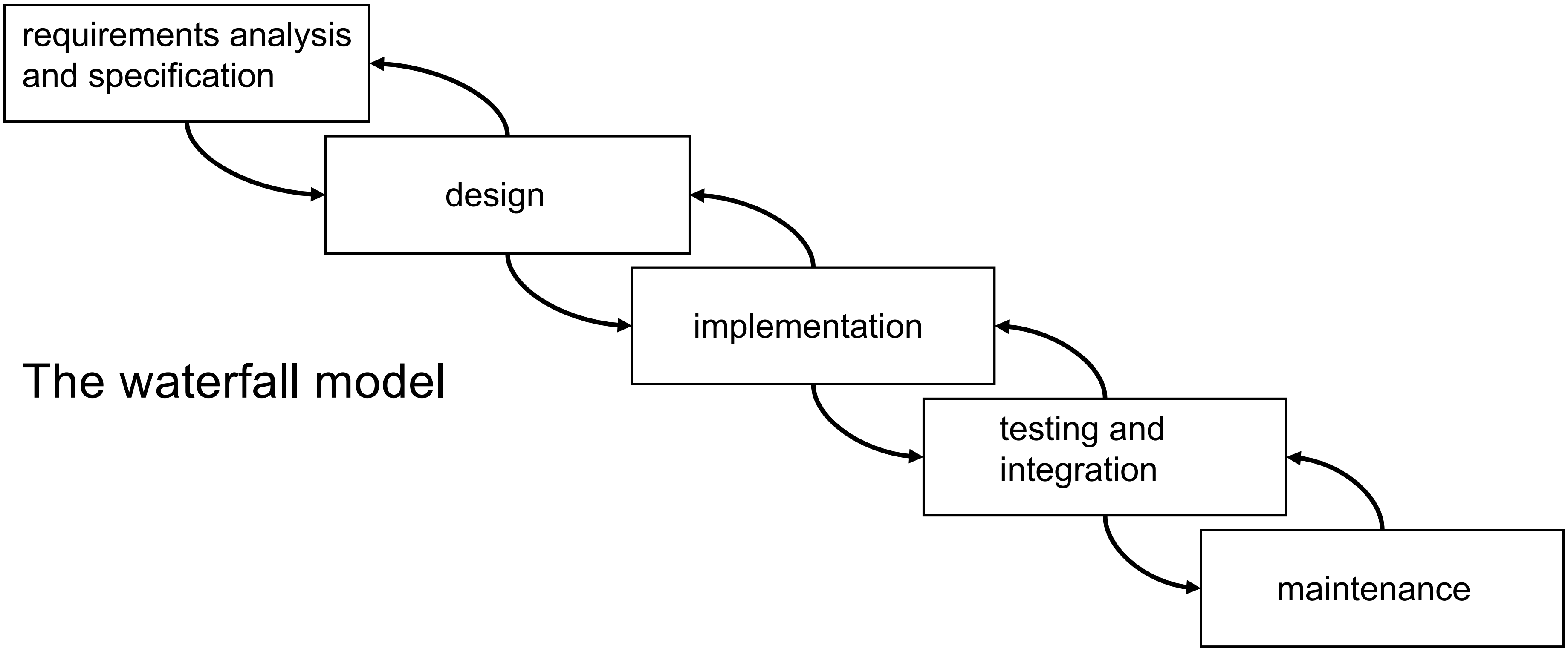
- This is called the **Maintenance** phase where the software is continually modified to adopt to the changing needs of the customer and the environment.

Software Process

- Then there is the question of how to organize the activities we mentioned before (requirements analysis, design, implementation, testing).

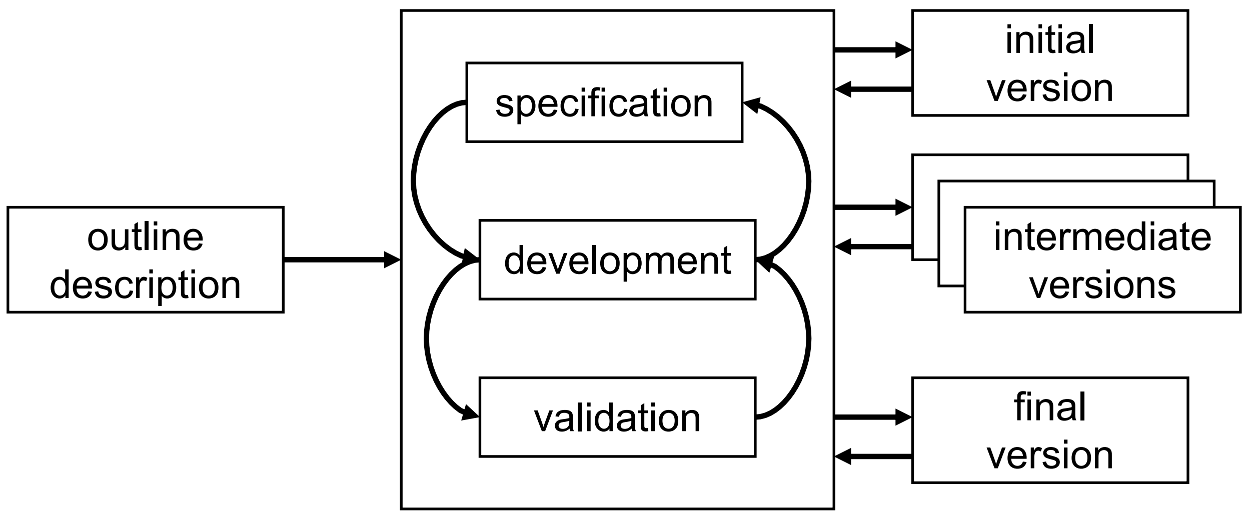
- There have been significant research on how to organize these activities
  – Waterfall model, spiral model, agile software development, extreme programming, Scrum, etc.

Software Engineering Process



The waterfall model

Software Engineering Process



Agile/Evolutionary/Incremental Software Development Process

Summary

- Software development involves multiple activities:
  - Requirements analysis and specification
  - Architectural design, detailed design
  - Implementation
  - Testing
  - Maintenance
  - Software development process

- There is active research in all of these areas in the software engineering community

This Course

- Software Engineering has been an active research area since its inception in 1968
- In this course we will have just a sampling of research in various areas of software engineering

Currently, most significant research results in software engineering are published in conferences:

- International Conference on Software Engineering (ICSE)
- ACM SIGSOFT International Conference on the Foundations of Software Engineering (FSE)
- IEEE/ACM International Conference on Automated Software Engineering (ASE)
- ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)

Top journals in software engineering are:

- IEEE Transactions on Software Engineering (TSE)
- ACM Transactions on Software Engineering and Methodology (TOSEM)

Active Research Areas In Software Engineering
(based on submissions to ICSE 2019)

Software testing	140	Program repair	19
Empirical software engineering	134	Distributed and collaborative software engineering	18
Software evolution and maintenance	117	Software reuse	18
Program analysis	115	Specification and modeling languages	17
Mining software engineering repositories	88	Refactoring	17
AI and software engineering	84	Recommendation systems	16
Security, privacy and trust	60	Requirements engineering	15
Tools and environments	58	Autonomic and (self-)adaptive systems	14
Validation and verification	54	Software process	14
Debugging	45	Cloud computing	14
Mobile applications	44	Software product lines	14
Human and social aspects of software engineering	39	Program synthesis	14
Program comprehension	35	Reverse engineering	13
Dependability, safety, and reliability	35	Software services	12
Fault localization	33	Software economics and metrics	12
Formal methods	29	Crowd sourced software engineering	11
Performance	27	Configuration management and deployment	11
Search-based software engineering	26	Component-based software engineering	9
Software modeling and design	24	Traceability	9
Software architecture	24	Software visualization	8
Programming languages	23	Human-computer interaction	8
Apps and app store analysis	22	Cyber physical systems	7
Agile software development	21	Green and sustainable technologies	5
Middleware, frameworks, and APIs	21	End-user software engineering	5
Parallel, distributed, and concurrent systems	20	Embedded software	4
Model-driven engineering	19	Ubiquitous/pervasive software systems	0

Papers we will discuss

- There are many active research conferences that focus on software engineering research and its sub-areas

- The premier professional organization in software engineering domain is ACM SIGSOFT
  - SIGSOFT sponsors conferences
  - It also gives distinguished paper and impact paper awards

- For this course I mostly selected papers that received awards and also some other impactful papers

- At the end of the class I hope that you will have a good understanding of the software engineering research and some of the major research contributions in this area