Software Engineering Introduction / Motivation

CIS350

Erik Fredericks // frederer@gvsu.edu

Adapted from materials provided by Byron DeVries, Jagadeesh Nandigam, Betty H. C. Cheng

Overview

- Welcome!
- Syllabus
- SE

What we *will* cover

Practical software engineering

Models and processes to get you prepared for the **real world**

Teamwork!

What we will *not* cover

In-depth programming

- For programming tasks, I'm going to make the assumption that you know a base level of programming

What do you want to cover?

Knowledge you are **assumed to have**

Ability to program in **some language**
- You will be working on a real project!

Ability to work **on a team**
- You're an adult, I also expect maturity and even effort-sharing

Ability to **talk to me if you have problems**
- Don't be nervous, I am very approachable!

How class will *generally* work

Our classes are 3 hour-sessions per week
- Classes will be a mix of:
    - Lecture
    - Discussions
    - Labwork
        - Guided/self-guided work

Subject to change based on material, naturally
- But, I want you to get **practical experience**

You will develop a semester-long project

Goals of this project:

1) Have something portfolio-worthy at the end

2) Develop it like a real-world project
   a) Project roles (ish)
   b) Software artifacts
   c) etc.

Project steps

1) Form teams and roles

2) Submit a project proposal

3) Setup a GitHub repository **and** website (github.io)

4) Project checkpoints

5) Project delivery
   a) Evaluate other teams' artifacts

Project?

I want you to create something that is **interesting to you**

Are you a software developer with latent tendencies for video game design?
- Make a video game!

Aspiring app developer?
- Make that app!

No clue!?!?
- Let's talk

What is important?

The software engineering **process**

Meaning:
- Artifacts
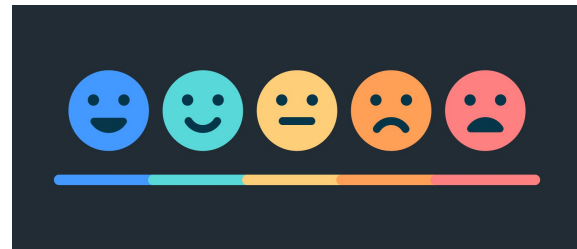- Project checkpoints
- Presentations

Team?

Teams of **2 to 4 people**
- Sorry, SE kind of necessitates being in a group

Things you'll have to do:
- Requirements specifications
- Artifacts
- Project checkpoints
- Presentations
- Etc.

Feedback touchpoints

Periodically I will ask for your feedback on the class itself

WE ARE IN A NEW ERA!

I am going to ask you (for participation points):
- What is working for you
- What is not working for you
- What can we do to address that

My goal is to make sure you are successful and happy with your learning experience

Our tech stack

Class is **synchronous in-person**, meaning:
1) Class runs at specific days/times
2) Office hours are in-person (can be online) as well

Class website: Blackboard // https://gvsu-cis350.github.io/gvsu-cis350/
Async Chat: Discord
Term projects: GitHub

If you want to get a hold of me for questions:
● Ping me in Discord
● Email me
● Visit office hours (virtual or in-person (generally))

Syllabus

As always, the syllabus is **worth reading**

Important topics like:
- When is my final exam?
- What is the grading breakdown?
- What time does this class meet?
- Where can I find the nifty textbook?
  - hehehhehehe
    - heh

Questions so far?

So...

What programming languages do you know?

And so...

What do **you** think of when you hear software engineering?

What is software engineering?

The study of systematic and effective processes and technologies for supporting all software lifecycle activities activities in order to support:

- Improved quality
- Reduced cost

Historical perspective

- **1940s**: Computers Invented

- **1950s**: Assembly Language, Fortran

- **1956**: First working silicon transistor

- **1960s**: COBOL, ALGOL, PL/1, Operating Systems

- **1969**: First Conference on Software Engineering

- **1970s**: Multi-User Systems, Databases, Structured Programming

Historical perspective

- **1980s**: Networking, Personal Computing, Embedded Systems, Parallel Architectures

- **1990s**: Information superhighway, distributed systems, OO in widespread use

- **2000s**: Virtual Reality, Voice Recognition, Video Conferencing, Global Computing, Pervasive Computing

- **2010s**: Electronic Health Records, Autonomous Vehicles, Awareness of Security Issues

Costs: Hardware vs Software
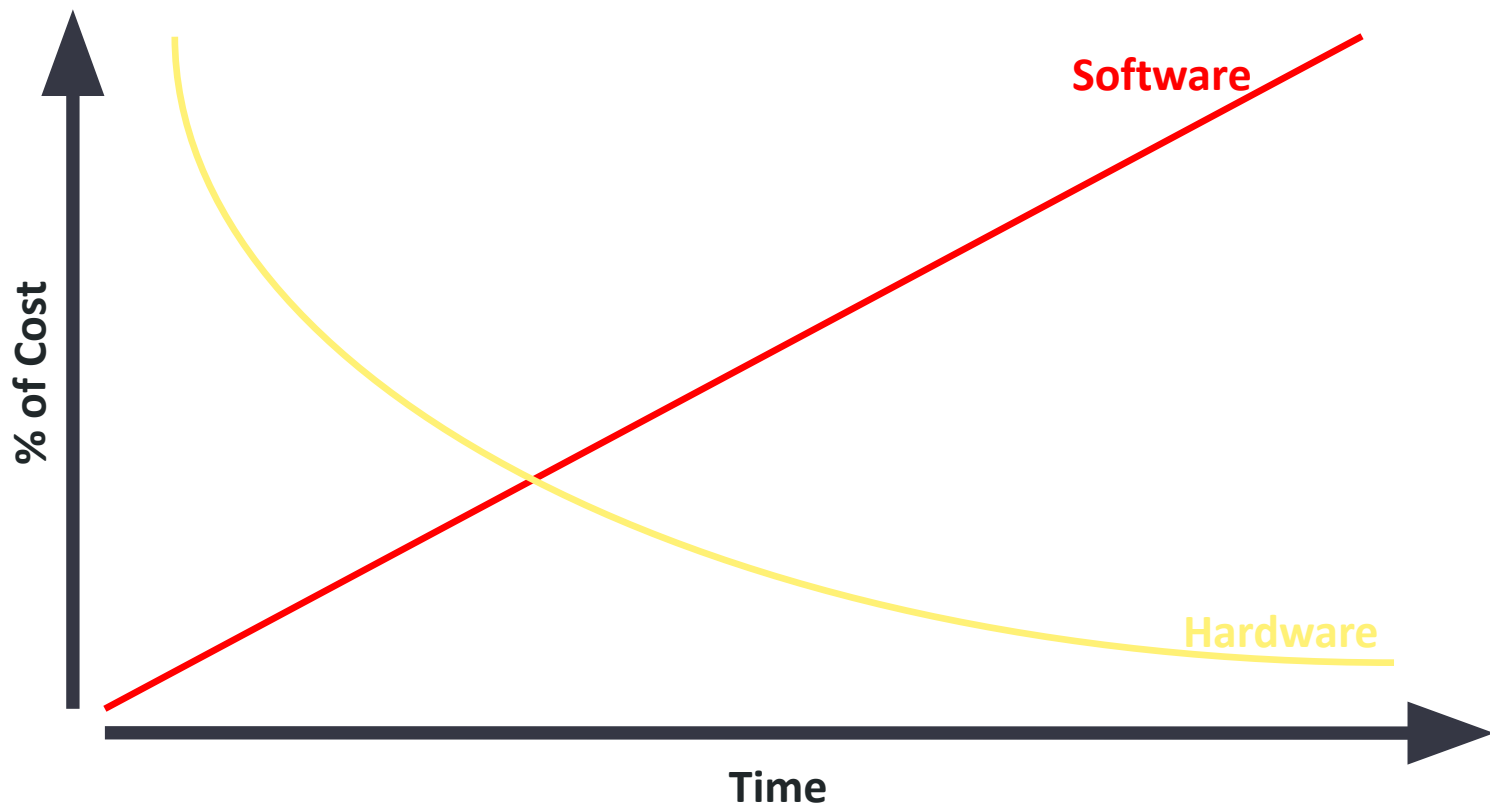
Sizes of software applications

- **Large**: 5 years, 100s of programmers, 1M LOC:
    - Example: MS Word, Excel

- **Very Large**: 10 years, 1000s of programmers, 10M LOC:
    - Example: Air Traffic Control, Telecommunications, Space Shuttle

- **Very, Very Large**: 15+ years, 1000s of programmers, 35M LOC:
    - Example: Year 2000 (Y2K) Updates

- **Ultra-Large Scale**: ? years, ? distributed programmers:
    - 1000s of heterogeneous sensors and decision / processing units
    - Distributed, heterogeneous systems and control
    - Examples: Intelligent transportation systems, health care systems, full Internet of Things (IoT)

Why is software so hard & expensive?

Isn't software free?
Growth in the number of:
- Features
- Inputs
- Outputs

A single increase causes **dimensional growth**

Sizes of software applications

- **Trivial**: 1 month, 1 programmer, 500 LOC:
    - Example: Introductory Programming Project

- **Very Small**: 4 months, 1 programmer, 2000 LOC:
    - Example: Course Project

- **Small**: 2 years, 3 programmers, 50K LOC:
    - Example: Nuclear power plant controller, Pace maker

- **Medium**: 3 years, 10s of programmers, 100K LOC:
    - Example: Optimizing Compiler

What…is the problem?

Software is difficult to maintain ➜ why?

Software schedule and cost estimation is difficult

Too many software projects fail*, including:
- Ariane Missile,
- Denver Airport Baggage System
- Therac-25

Budget, schedule, or functional issues that result in cancellation of the project

Let's look at a few case studies:

Case Study : Ariane 5

Ariane 5 rocket -- created by European Space Agency
Required **10 years** and **$7 billion** to build

In under a minute after launch, the rocket exploded

As a result of...
    Trying to put a 64-bit number into a 16-bit variable

A bug and a crash, J. Gleick, New York Times, Dec 1996

Case Study : Toyota Prius Braking

Delay in brake software caused vehicles to brake slower than expected

If traveling 60mph, the brakes didn't trigger until ~90ft beyond initial brake press

Disconnect in the anti-lock brake system (ABS) -- < 1 second lag

CNN Feb 4, 2010

Case Study: Mars Orbiter

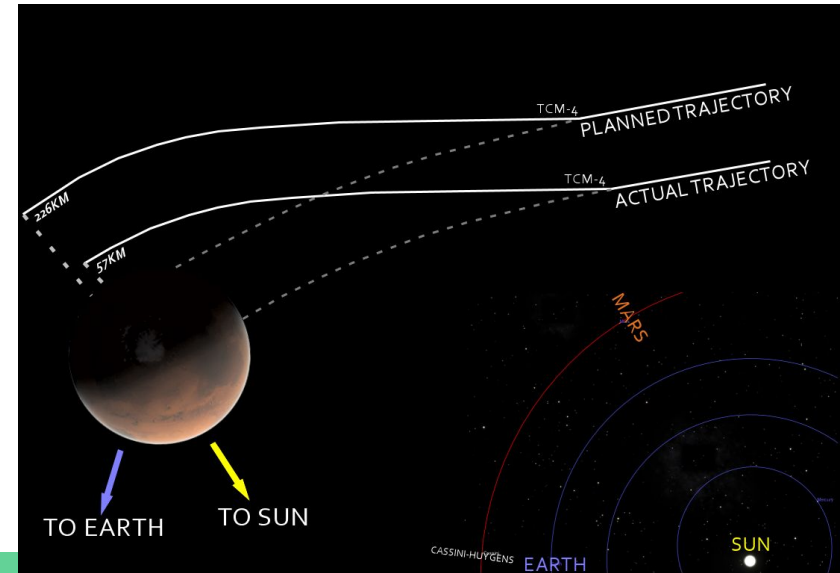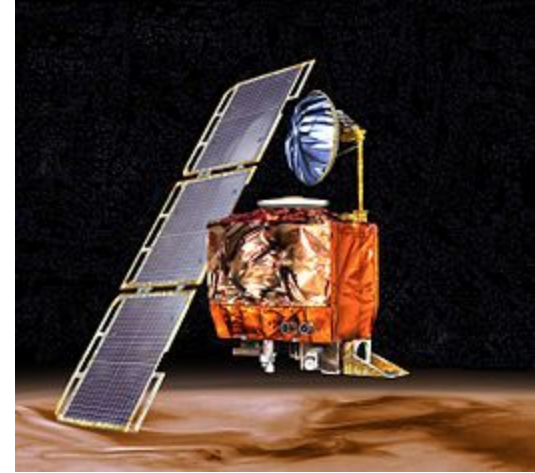https://en.wikipedia.org/wiki/Mars_Climate_Orbiter

Satellite for Mars to study climate/surface (launched 1998)

Crashed to the surface of Mars after a successful launch and slightly-odd inconsistencies in trajectory

Loss of $327.6 million (in 1998)

Cause?
- One team used metric units, one team used imperial units

Case Study : Inadvertent Overdosing of Radiation

Let's talk about the **Therac-25**
- Radiation therapy machine (chemotherapy)
- Iteration over the well-used Therac-6
  - Now completely software-controlled
    - …. Including safety mechanisms
    - Hey, its probably cheaper to have everything be software-controlled, right?

- Machine was extensively tested and verified over several years
  - A single programmer wrote all the code
  - Unit testing and general SW testing was minimally done

Leveson 1993
http://www.cs.umd.edu/class/spring2003/cmsc838p/Misc/therac.pdf

Therac-25

11 units were installed in the US and Canada
- **Six** massive overdoses of radiation were reported

Why?
- Program faults from the Therac-20 software codebase that were not previously detected
  - Well, why weren't they detected?
    - Because the **hardware interlocks** prevented the machine from ever entering a state
    - Now that it was completely SW-controlled, well, no safety mechanism existed to prevent overdose

Therac-25

Effectively this problem results from **race conditions** and **concurrency errors**

But, the human element is also to blame
- Medical staff initially did not believe that their machine was causing the problems
- Improper training led staff/nurses to use shortcuts or rapid key entries that caused the machine to go into the wrong power mode
- From Wiki (https://en.wikipedia.org/wiki/Therac-25#Additional_reading):
  - The failure only occurred when a particular nonstandard sequence of keystrokes was entered on the VT-100 terminal which controlled the PDP-11 computer: an "X" to (erroneously) select 25 MeV photon mode followed by "cursor up", "E" to (correctly) select 25 MeV Electron mode, then "Enter", all within eight seconds.[5] This sequence of keystrokes was improbable, and so the problem did not occur often and went unnoticed for a long time.[3]

What could have been done to prevent this?

What does this tell us?

Not that the engineers were incompetent. A **lot** of effort went into designing and testing these systems.

It tells us that testing is **never** complete and **it is impossible to consider every single conceivable issue**

Could a Lint test have solved the Ariane-5 explosion? Possibly
Could extensive human testing have solved the <X> catastrophe? Maybe, maybe not.

Therefore, it is important to verify and test as much as possible prior to release!

So software engineering…

Historically, probably some of the driest classes you will get (unless if you have one of our nifty professors, naturally)

But that's all stuff that you do in the working world, right?
Specifications, testing, verification. That's the bland side of programming.

Where's the excitement? Where's the sex appeal?

Well…

This is also why I feel that software engineers should have the same requirement as Mechanical Engineers to be tested for a Professional Engineer license.

People's lives can and will be at stake as a result of your programming and design processes.

Software Engineer's Responsibilities

Assume you programmed the Therac-25

You are **directly (or indirectly) responsible** for causing severe radiation burns, at least one of which resulted in a breast amputation

The patient died on November 3, 1985, of an extremely virulent cancer. An autopsy revealed the cause of death as the cancer, but it was noted that had she not died, a total hip replacement would have been necessary as a result of the radiation overexposure. An AECL technician later estimated the patient had received between 13,000 and 17,000 rads.

Software/Quality Engineer's Responsibilities

Now that (I assume) you're scared straight

Be responsible when you program. It's not a game. People's lives will be affected by your actions.

/end soapboxing

So now that we're all mortified…

In this class, we will consider the real-world implications of software engineering concepts in terms of real-world concerns

- So let's talk about SE!

Why do we need SE?

To **predict** time, effort, and cost

To **improve** software quality

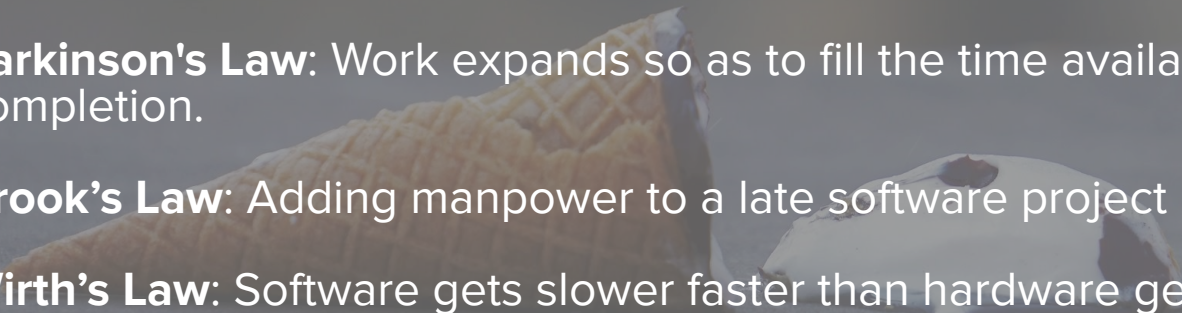To **improve** maintainability

To **meet** increasing demands

To **lower** software costs

To successfully **build** large, complex software systems

To **facilitate** group effort in developing software

When software engineering is done *incorrectly*

each of these goals can be subverted.

Some of the "laws"

- **Murphy's Law**: If anything can go wrong, it will.

- **Hofstadter's Law**: It always takes longer than you expect, even when you take into account Hofstadter's Law.

- **Parkinson's Law**: Work expands so as to fill the time available for its completion.

- **Brook's Law**: Adding manpower to a late software project makes it later.

- **Wirth's Law**: Software gets slower faster than hardware gets faster.

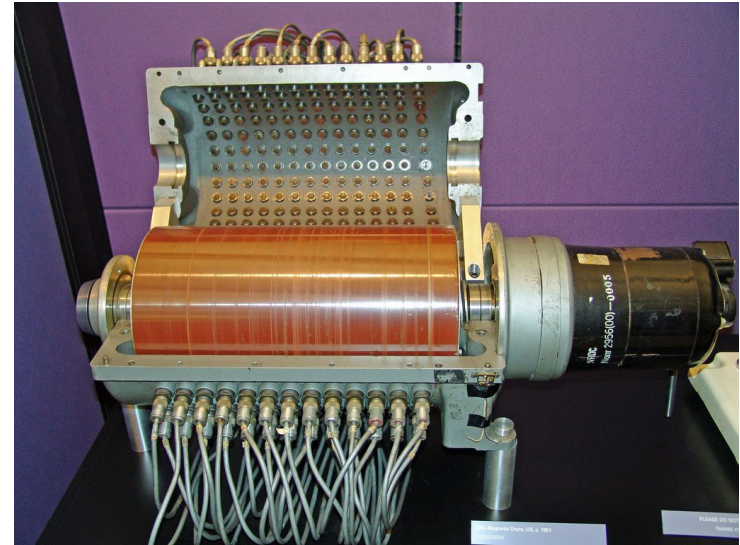# Shaw's Model of Engineering Evolution*
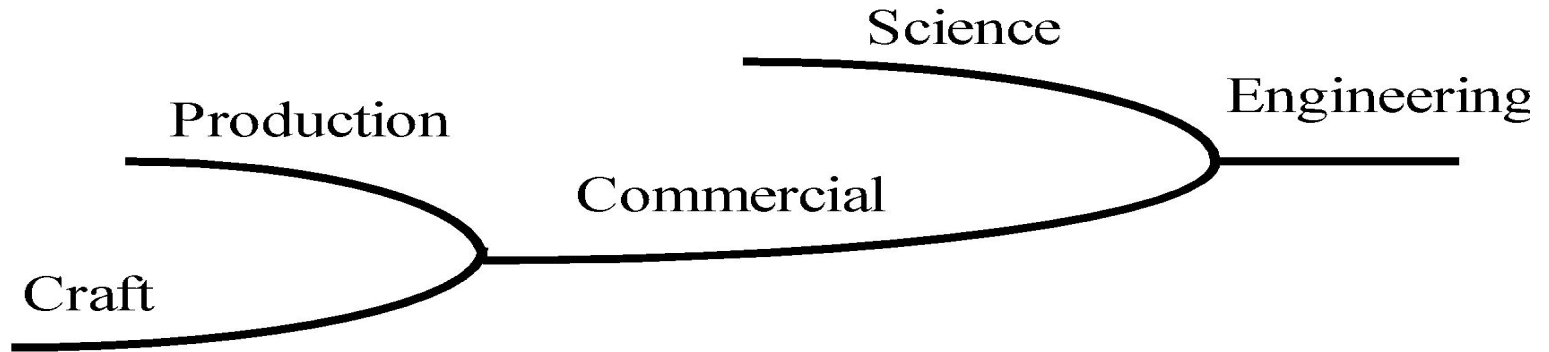
*[Shaw-IEEE-Computer90]

Craft Characteristics

- Virtuosos and talented amateurs
- Intuition and brute force
- Haphazard progress
- Casual transmission of knowledge
- Extravagant use of available materials
- Manufacture for use rather than sale

Examples: Woodworking, Artists

Commercial Production Characteristics

- Skilled crafts
- Established procedure
- Pragmatic refinement
- Training in specific domain (e.g., mechanics-- automotive technicians, structures-- construction worker, electricians)
- Economic concern for cost and supply of materials
- Manufacture for sale

Examples: (Small Scale) Furniture Manufacturing, Automotive Components

Professional Engineering Characteristics

- Educated professionals
- Analysis and theory
- Progress relies on science
- Educated professional class
- New applications enabled through analysis
- Market segmentation by product variety

Examples: Civil Engineering (bridges, buildings), Automotive Engineering (electronics, mechanical engineering)

Civil Engineering Evolution
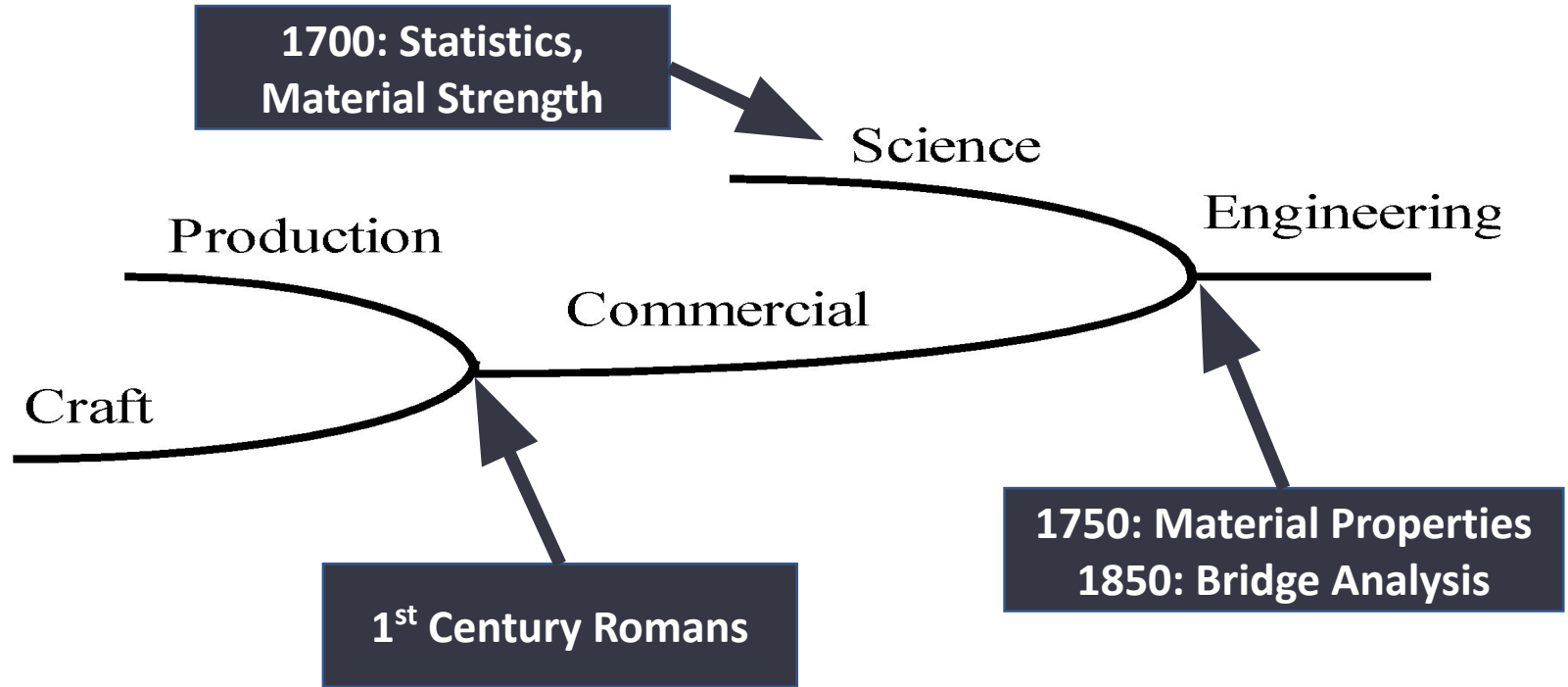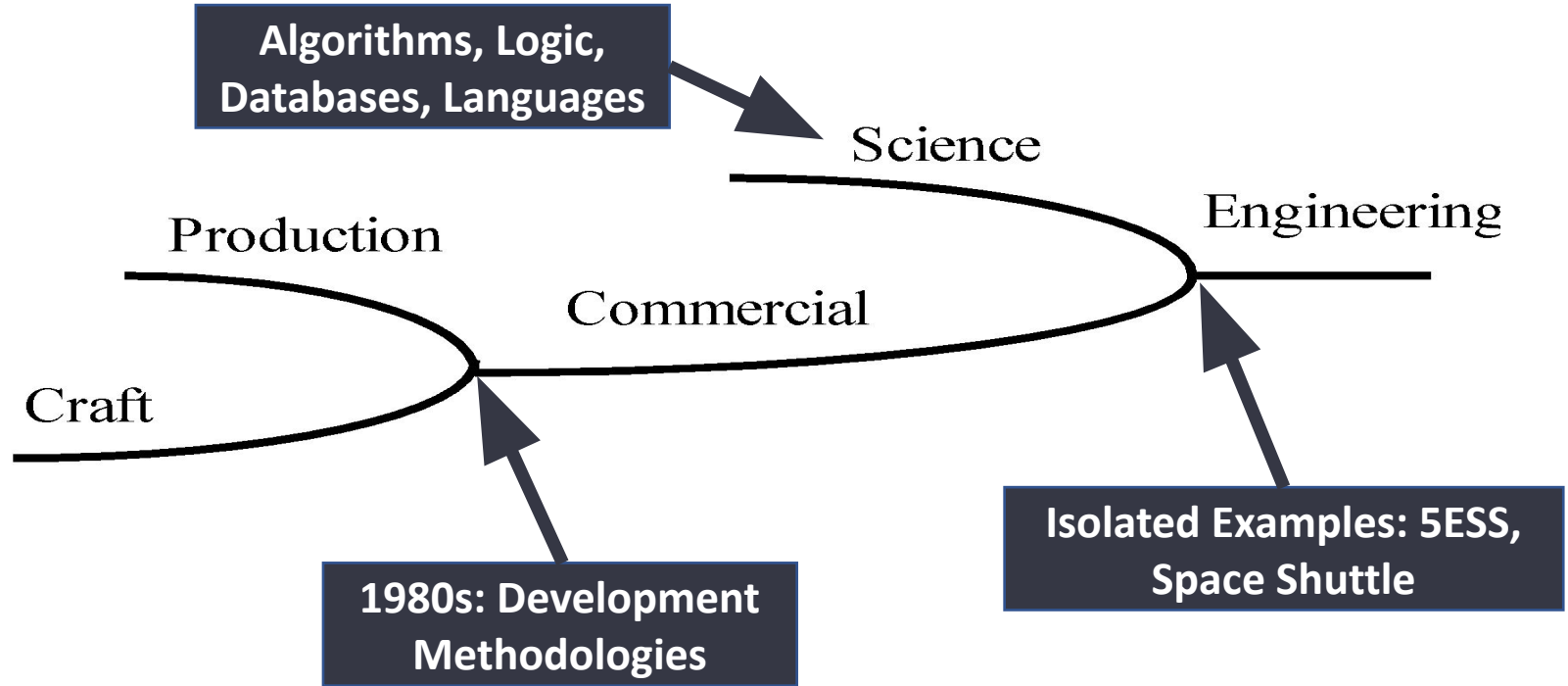
Civil engineering: Bridges

Basis in Theory (or two theories):
- Statistics: Composition of Forces
- Material Strength: Force required to bend a beam

Theories preceded *use in* Civil Engineering by 150 years

The underlying production practices preceded *engineering* (i.e., a practical application of theories) by 1700 years!

Software Engineering Evolution*

*Circa 1990 as described in [Shaw-IEEE-Computer90]

Two **pillars** of SE education

Basis in:
- Production processes and process frameworks
- Rigorous theories addressing design problems that attend to the various phases of these processes

This course:
- Organized around the first "pillar"
- structured so that process issues will motivate introduction of theoretical content

The graduate courses cover additional theoretical components.

Software engineering phases

Definition: What are we building?

Development: How are we building it?

Verification: Did we build it right?

Maintenance: Managing change and errors.

Umbrella Activities: Common across phases.

Definitions

Requirements definition and analysis:

The developer must understand:
- Application Domain
- Required Functionality
- Required Performance
- User Interface

Definitions

Project Planning:
- Allocate Resources
- Estimate Costs
- Define Work Tasks
- Define Schedule

Systems Analysis / Allocate System Resources to:
- Hardware
- Software
- Users

Development (1/2)

Software Design:
- User Interface Design

- High-Level Design:
    - Define Modular Components
    - Define Major Data Structures

- Detailed Design:
    - Define Algorithms
    - Define Procedural Detail

Development (2/2)

Implementation:
- Develop Code for each Module
- Unit Test Creation

Integration:
- Combine Modules
- System Test Creation

Verification

Release Verification:
- Unit Test Execution
- System Test Execution
- Exploratory Testing

Maintenance

Types of Maintenance:

- **Correction**: Fix software defects

- **Adaption**: Accommodate Changes
    - New Hardware
    - New Company or Legal Policies

- **Prevention**: Make more maintainable

- **Enhancement**: Add Additional Functionality

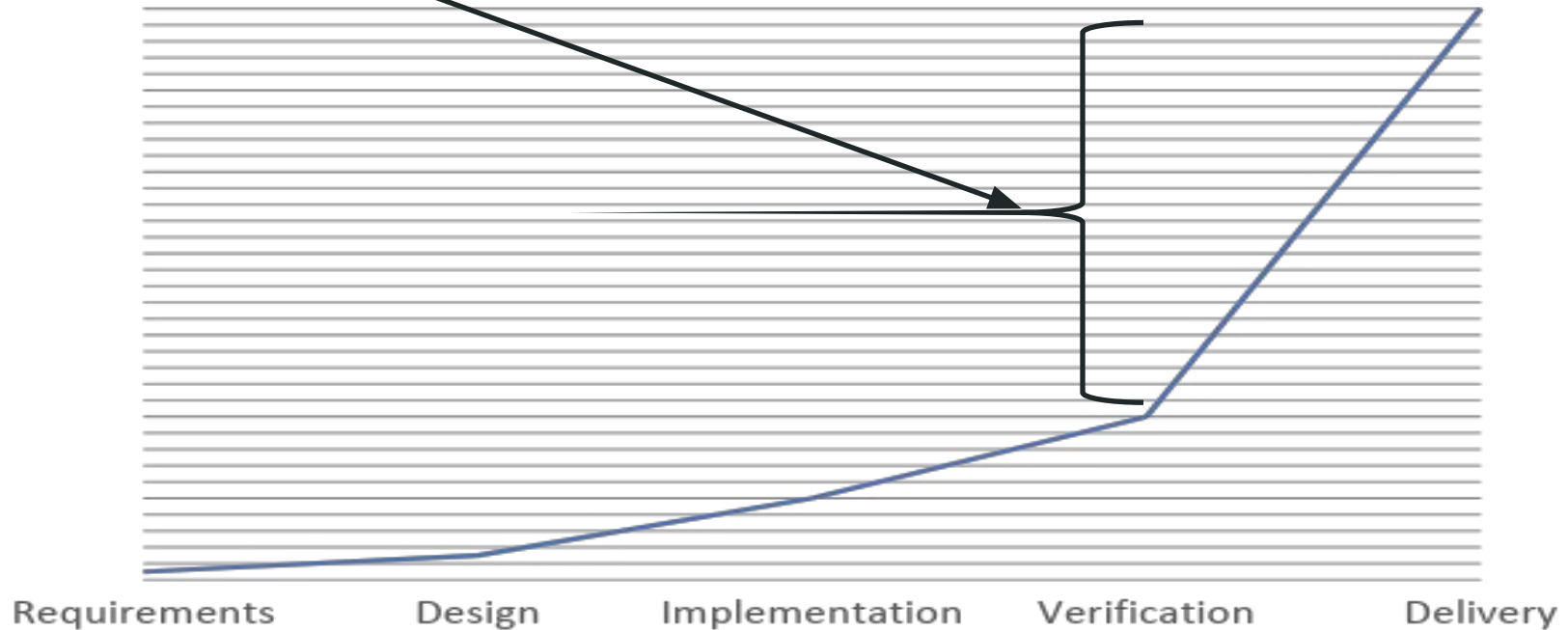# Umbrella activities

Types of Umbrella Activities:

- **Reviews**: Assure quality and correctness

- **Documentation**: Support understanding or certification

- **Version Control**: Track changes

- **Configuration Management**: Ensure integrity of releases

Portion of Software Engineering Costs

■Definition ■Development & Verification ■Maintenance
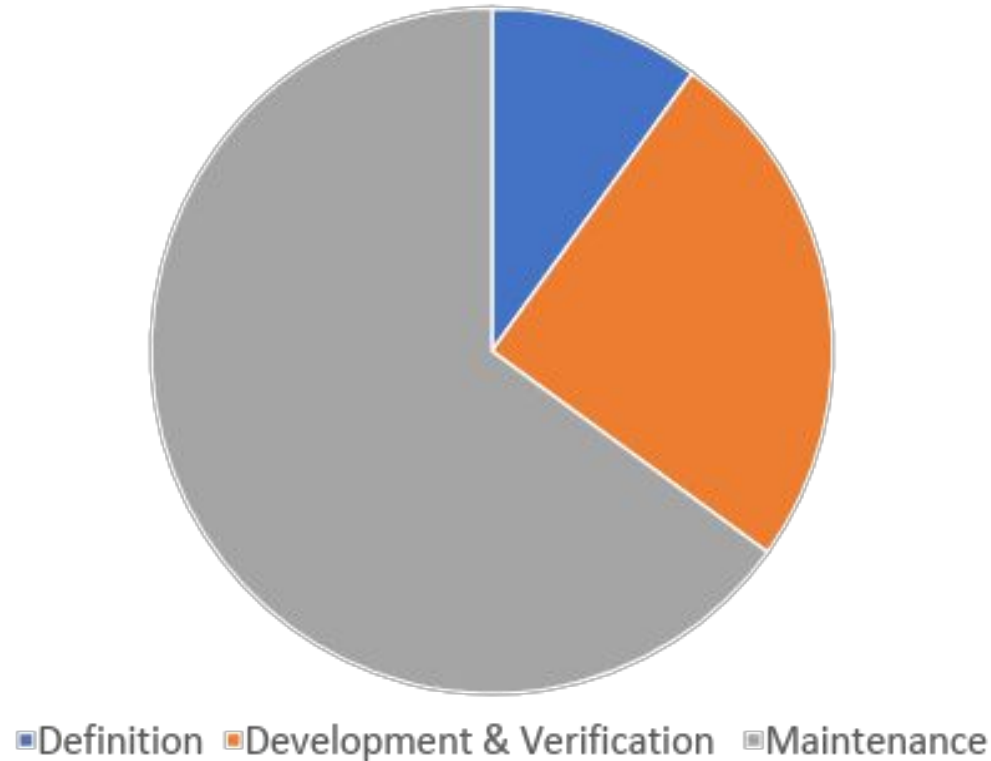
Proportional Cost of Fixing Software Errors*

The payoff of Software Engineering

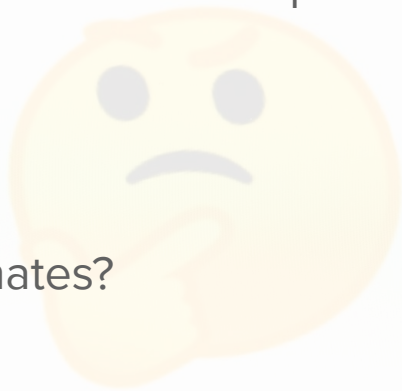Requirements   Design   Implementation   Verification   Delivery

*In proportion to fixing software errors during requirements

ChatGPT Discussion

First off, how many of you use ChatGPT to help with assignments?
      (no judgements, yet)

How do you use it?

And do you know that it hallucinates?
      (here, have some paper)

ChatGPT Discussion

So should we use it or not use it?

Well, the genie's out of the bottle and it is probably here to stay (unless if that lawsuit is successful...)
- Just be aware that its output *may* not be completely correct

...and don't copy and paste it directly into your answers either ಠ_ಠ

HOMEWORK

Homework posted -- ethics reflections and project team formation.