Introduction to Software Engineering

ECSE-321

Unit 1 - Introduction

Engineering

- **Per *Merriam-Webster Collegiate Dictionary*,**
  - a. the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people
  - b. the design and manufacture of complex products
  - c. the discipline dealing with the art or science of applying scientific knowledge to practical problems (WorldNet ®)

Software Engineering

- **Per *ECSE-321:***
  - a. the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people
  - b. the design and manufacture of complex products
  - c. the most important of all engineering disciplines

What is the problem?

- LARGE software projects
- How large are we talking about?
  - Typically, more than 100,000 LOC (lines of code)
- Large projects
  - Large budgets
  - Large teams
  - Years of development
- Concern?

Large Projects

Driven by commercial viability

Need to meet certain "expectations"

To be successful:

- Reach market before competing products
- Need to have ***key*** features than others
- Need to have ***more*** features that others
- Etc.

Problems with Large Projects

- Large projects (more than 500000 LOC) is a risky undertaking
- What is the risk?
  - 65% of large projects are cancelled before completion
  - Lost investment
  - Average cancelled projects in US is about a year behind and over budget by 200%
- Cancelled projects amounted to $14billion in 1993

More on Large Projects

- Of completed projects
  - 2/3 experience schedule delays and cost overruns
  - 2/3 experience low reliability and quality problems in the first year of deployment

Why Projects Fail?

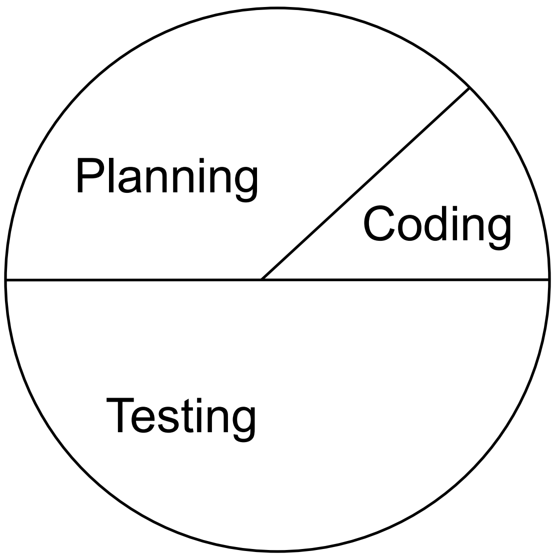
- Referred to “*death march projects*” by Nancy Leveson
  - Feature creep
  - Thrashing
  - Integration problems
  - Overwriting source code
  - Constant re-estimation
  - Redesign and rewriting during testing
  - No documentation of design decisions
  - Etc.

Types of “Death March Projects”

- Mission Impossible
  - Likely to succeed, happy workers
- Ugly
  - Likely to succeed, unhappy workers
- Kamikaze
  - Unlikely to succeed, happy workers
- Suicide
  - Unlikely to succeed, unhappy workers

More on the “Software Project Problem”

Development Costs



What are the “Other” Problems?

- So far we discussed the issues in getting things “done”
- Completing the project
- What are the problems after completing the project?

Some "Other" Problems

- Safety of software systems
- Software is controlling many mission critical systems
  - Medical equipment
  - Real-time systems (e.g., car stability control)
  - Large-scale infrastructure (e.g., power Grids)
- How do we ensure software is safe?
- Failure modes of software + hardware needs to be examined?

Why Software Engineering?

- Definitely, SE can improve the worst case scenario
  - Worst software developed under a well regimented SE process is going to be better than a worst software developed under a ad-hoc process
  - Best software.. cannot say much!

Why Software Engineering?

- Use well known engineering principles to design, develop, maintain high quality software systems
- Is SE going to always result in high quality software?
- Probability NOT! Why?

Why *Not* Software Engineering?

- Software is often linked to spontaneous thinking
- Can SE mean end of “hacking” culture?
- Software hackers beware!
  - Go the way of quack doctors!
  - Extreme scenario - need to be certified professional before releasing a software
  - Each piece of software needs to be certified?

Why is SE hard?

- "Curse of flexibility"
  - Software usually takes all the ***slack***
  - Software engineers usually save hardware engineers' ___
- Complexity management
- Lack of historical usage information
- Large discrete state spaces

Typical Computing Model

General Purpose Machine + **Software** = **Special Purpose Machine**

- Machines physically impossible become feasible
- Changes without retooling – runtime reconfiguration
- Abstract specification with implementation details

Curse of Flexibility

- Software takes care of the “rest” and acts as a glue
- Not physically constrained to help
  - Limit the scope
  - Control the complexity
- Flexible – start working before the problem is fully understood (e.g., early stage simulators)

Efforts to Characterize SE

- Often SE is compared to Civil engineering
  - E.g., constructing a building
- Good analogy
  - Size matters: dog house versus skyscraper
  - Team effort with careful project planning
  - Difficulties with design changes
  - Other relevant elements: building structure, scaffolding, architecture, components, etc.

SE versus Civil Engineering

- Civil engineering is guided by the laws of physics
- Software engineering lacks the underlying laws
- Civil engineering relies on components a lot
- Software engineering is making some progress in that direction with the advent of “service” orientation

How do we characterize Software?

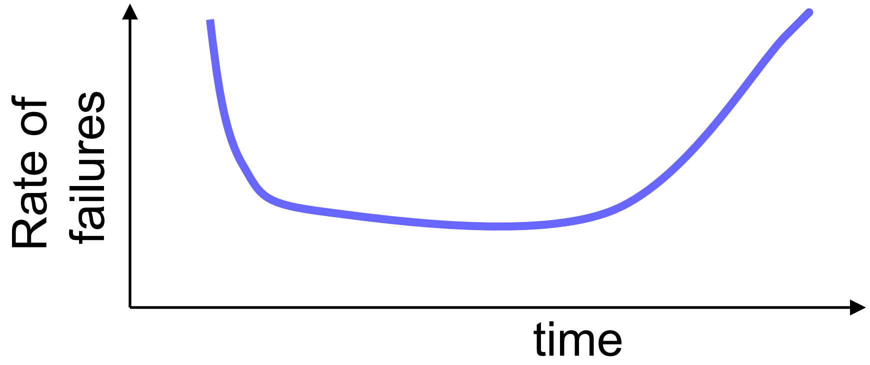
- Can consider some axes of variabilities:
  - Size
  - How humans interact with it
  - Requirements (changes in requirements)
  - Need for reliability
  - Need for security
  - Portability
  - Cost

Categorize the following Software Systems

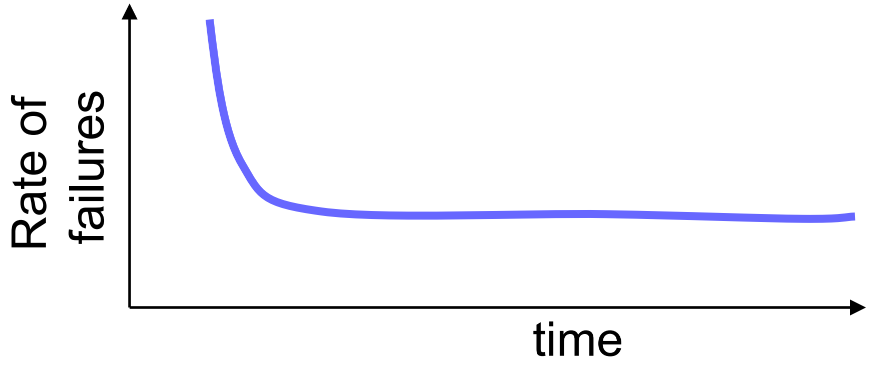
	Open office	Car Stability Controller	Social Network Portal
Size			
Interactivity			
Requirements			
Reliability			
Security			
Portability			
Cost			

Failures

- HW failure curve

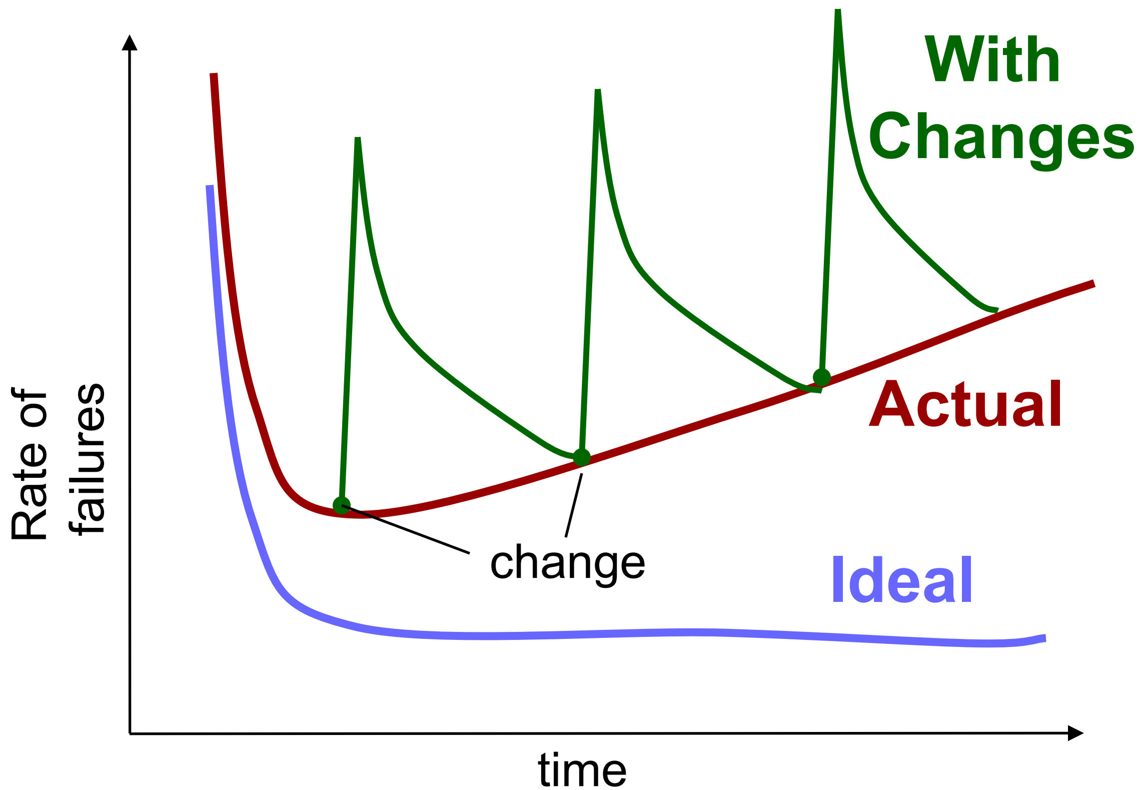


- SW (ideal) failure curve



Software failures

- Real world failure curve



Software Engineering Myths: Management

- We have books with rules. Isn’t that sufficient?
  - Which rules are important? (This is a general problem with certification)
- If we fall behind, add more programmers
  - Adding people to an already late project makes it later!
- We can outsource it
  - If we don’t know how to do it in-house, it is harder to do it with outsiders

Software Engineering Myths: Customer

- We can refine the requirements later
  - A recipe for disaster
- The good thing about software is that is easier to change
  - It can cost more to change later

Software Engineering Myths: Practitioner

- Lets write the code, so we will be done faster
  - Sooner you begin coding, the longer it will take to finish
- Until I finish, I cannot assess the quality
  - Software and design reviews are more effective than testing
- There is no time for software engineering
  - Is there time to redo the software?

Recap: What is SE for?

- We want to build a large system
- How will we know the system works?
- How do we develop the system efficiently?
  - Minimize time
  - Minimize dollors
  - Minimize ...?

How do we know the software works?

- How do we know a given behaviour is a bug?
  - Have some separate specification of what the program must do
  - We need to define the requirements for "working" before start coding

Teams and Specifications

- Do we really need to write specifications?
- Informally, people can
  - Discuss what to do
  - Divide up the work
  - Implement incompatible components
  - Surprised when it does not just work together

What can we do?

- Write specifications:
  - Write down exactly what it is supposed to do
  - Make sure all team members understand it
  - Keep the specifications up to date
- Still.. we could have problems
  - Ambiguities and contradictions can occur
  - They lead to bugs
  - Problems can be reduced

Summary #1: Importance of Specifications

- Specification allows us to:
  - Check whether software works
  - Build software in teams
- Actually checking that software works is hard
  - Code reviews
  - Static analysis tools
  - Testing and more testing
  - ...

How do we code efficiently?

- We want to minimize the development time
  - Reach market first!
- Coding faster...
  - hire more programmers
  - parallelize the programming process!

Parallel Development

- How many programmers can we keep busy?
  - As many as there are independent tasks
- People can work on different modules
  - Thus we get parallelism
  - And save time
- What are the pitfalls?

Few words about Parallel Processing

Few words about Parallel Processing

Few words about Parallel Processing

Pitfalls of Parallel Development

- Problems are the same as in parallel computing
- More people = more communication
  - expensive, harder to manage
- Individual tasks cannot to be too small
- We need to take care of sequential constraints

Interfaces

- Chunks of work must be independent
  - Put them together to form the final system
- We need well defined interfaces between components
- Interfaces must not change much

Defining Interfaces

- What are interfaces?
  - Specifications between components that are supposed to be independent

Software Architecture

- To define interfaces, we must decompose a system into separate pieces
- How to do this?

Decomposition can be driven by

- What the system does
- How we build it
- Who builds it

Summary #2: Decomposition

- Efficient development requires
  - Decomposing the system into pieces
  - Good interfaces between pieces
- Pieces should be large
  - Don’t try to break into too small pieces
- Interfaces are specifications of boundaries