CSE 470 : Software Engineering

Afrina Khatun
Lecturer
CSE, BRACU

Marks Distribution

- Attendance : 5
- Quiz: 10
- Mid Term Exam : 25
- Project and Assignments : 20
- Final Exam : 40

Reference Books

1. Ian Sommerville, "Software Engineering", Addison Wesley, 8th edition, 2007.

2. Roger S Pressman, Roger Pressman,"Software Engineering: A Practitioner's Approach", McGraw-Hill, 7th edition, 2010.

3. \\tsr\Fall-2019\CSE\AKH\CSE470\

Ground Rules

- Makeup **MID** and **FINAL** will be conducted as BRACU policy.
- **No makeup quiz**.
- **Best (n-1) quiz** will be considered.
- **Assignment/ Report Submission** needs to be submitted by given **deadline**.
- Disciplinary action will be taken in case of **Cheating**.
- You need to maintain **class behavior**.
- You are **encouraged** to ask questions in class.
- You can **reach me** by phone, email etc. in case of important (valid and logical) query.
- Visit me in my **consultation hour** (available in tsr) for any query, discussion or even for gossiping.

Goals of Software Engineering

- To produce software that is **absolutely correct**.
- To produce software with **minimum effort**.
- To produce software at the **lowest possible cost**.
- To produce software in the **least possible time**.
- To produce software that is **easily maintained** and **modified**.
- To maximize the **profitability** of the software production effort.

- In practice, none of these ideal goals is completely achievable. The challenge of software engineering is to see how close we can get to achieving these goals.

- The *art* of software engineering is balancing these goals for a particular project.

Real-life Software

Inevitably most software systems are LARGE Systems.

This means:

- Many people involved in design, building and testing,
- Team effort, not individual effort
- Many millions of $ spent on design and implementation
- Millions of lines of source code
- Lifetime measured in years or decades
- Continuing modification and maintenance

Example — Eclipse

Eclipse is a popular software development environment for Java.

Some interesting characteristics of a recent release:

- Lines of source code > 1,350,000.
- Effort(person-years) > 400.
- Classes 17,456.
- Inheritance relations 15,187.
- Methods 124,359.
- Object instantiations 43,923.
- Fields 48,441.
- Call relations 1,066,838.
- Lifetime bugs > 40,000.
- Est. Development cost > $ 54,000,000.

Why Is Software Engineering Important

Cost of getting software *wrong* is often terrible.

- **Bankruptcy** of software producer.
- Injury or loss of human life **broken software can KILL people.**
- Software producer profitability depends on producing software efficiently and minimizing maintenance effort. **Software reuse is an economic necessity.**
- Immense body of old software (legacy code or dusty decks) that must be rebuilt or redesigned to be usable on modern computer systems.
- **Very, very few contemporary systems work correctly when first installed**. We need to do much better.
- **Over $600,000,000,000 spent each year on producing software**.

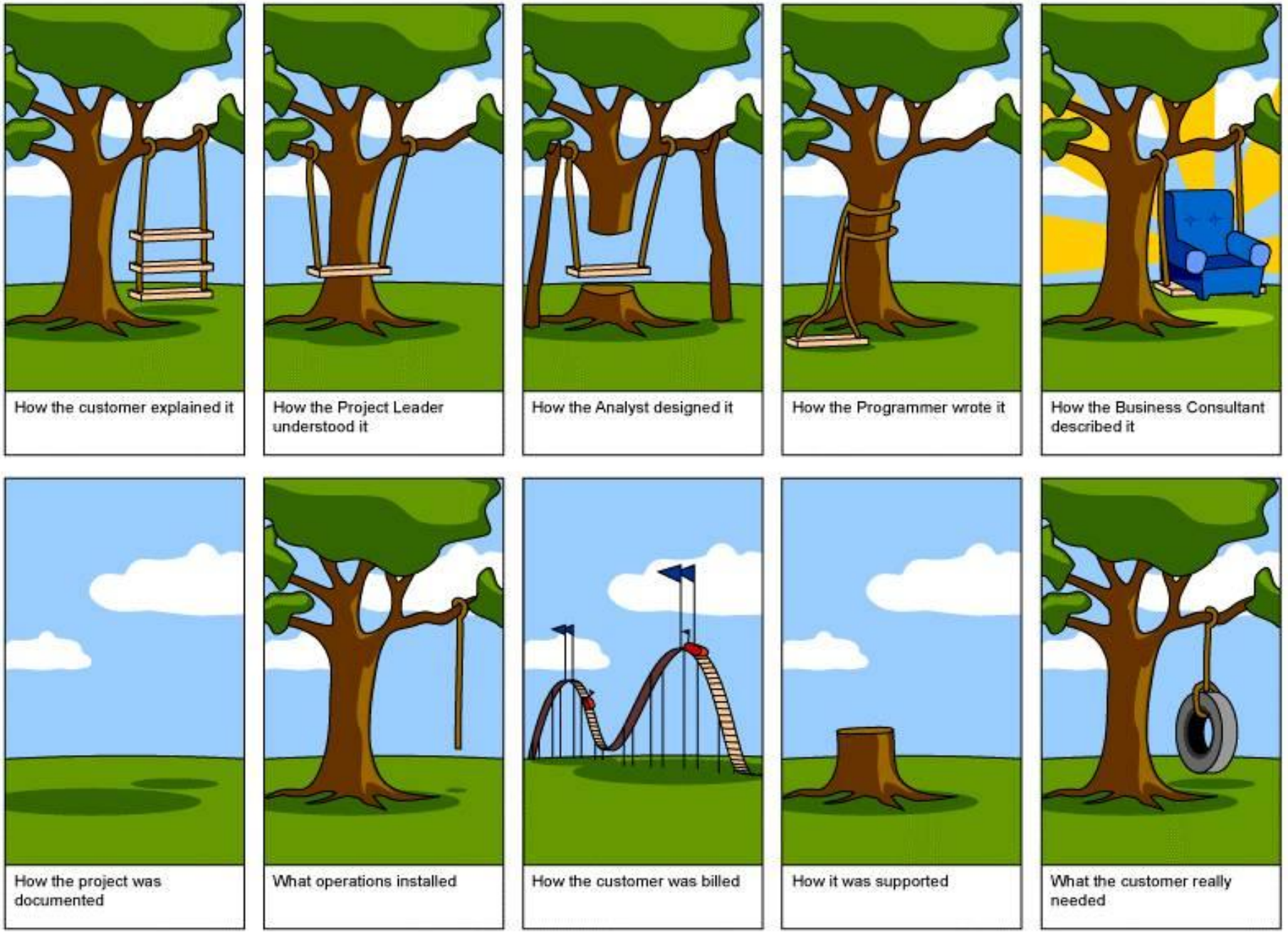
Software Horror Stories

- Bank of America spent $23,000,000 on a 5-year project to develop a new accounting system. Spent over $60,000,000 trying to make new system work, finally abandoned it. Loss of business estimated in excess of $1,000,000,000.

- The B1 bomber required an additional $1,000,000,000 to improve its air defense software, but the software still isn't working to specification.

- Ariane 5, flight 501.

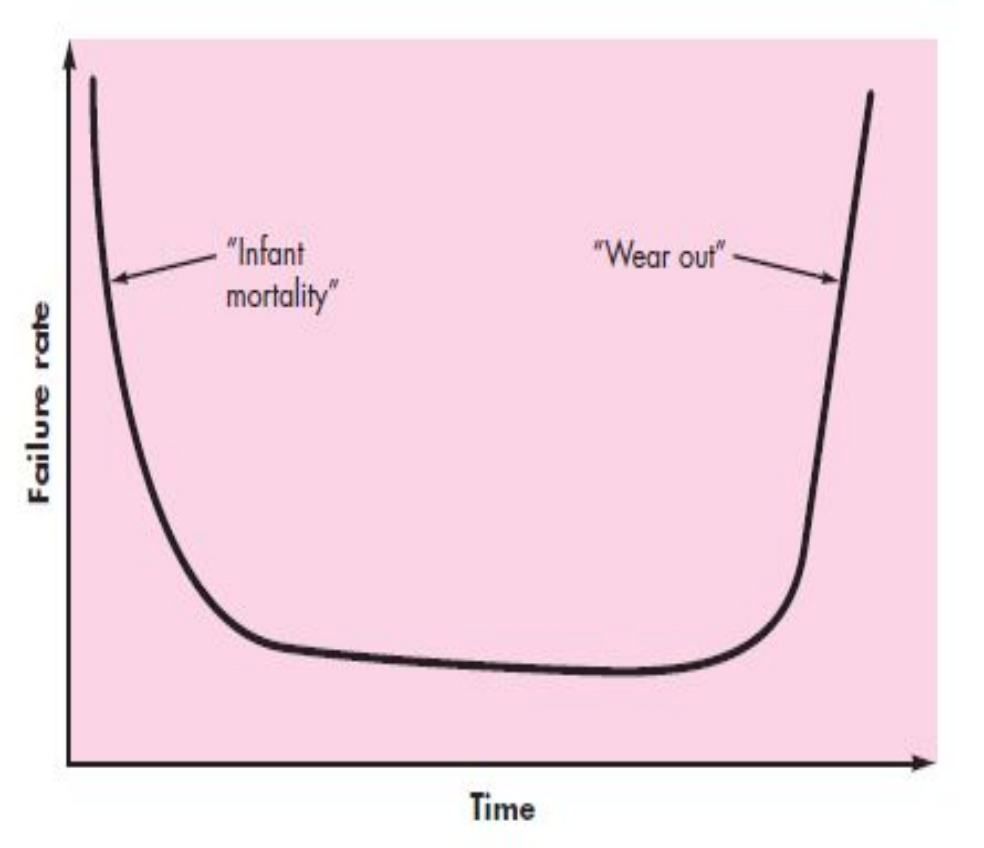
The loss of a $500,000,000 spacecraft was ultimately attributed to errors in requirements, specifications and inadequate software reuse practices.


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

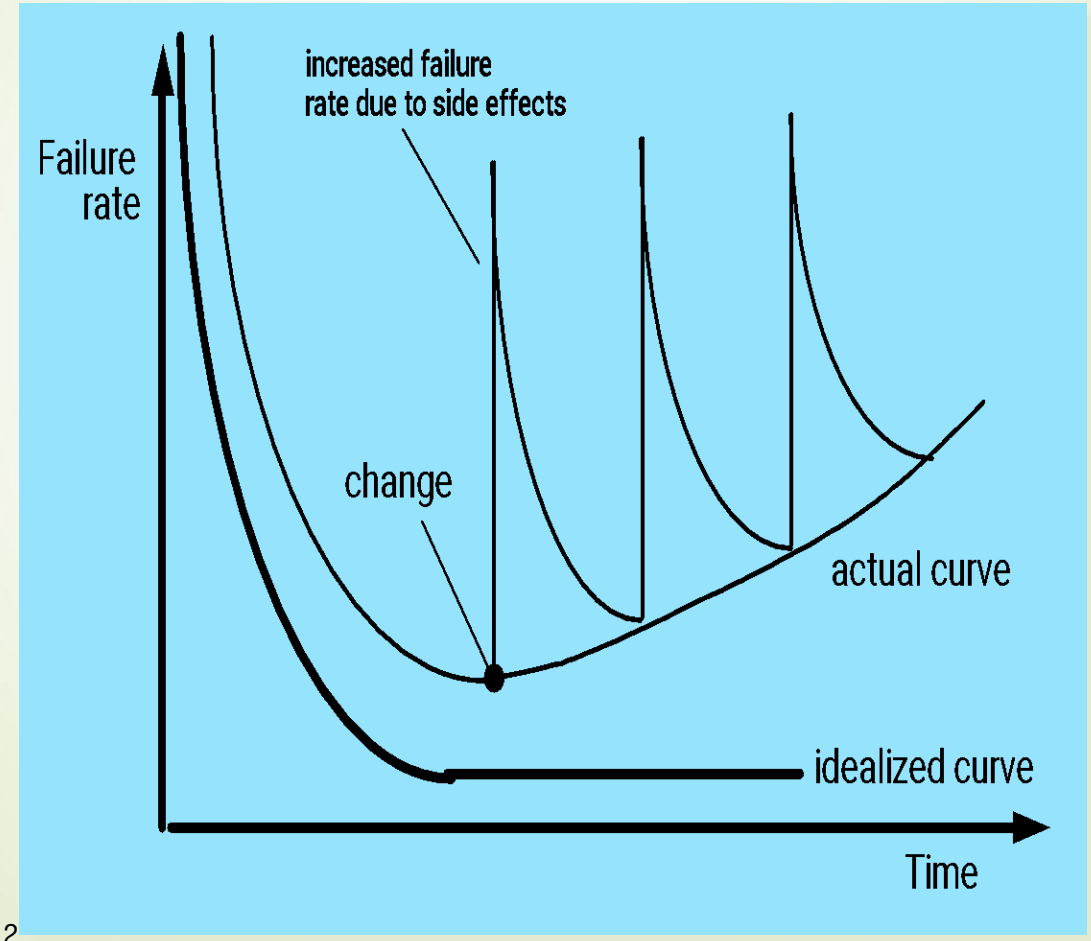
What Is Software ?

- Programs
  - Source code
  - Data structures
- Documents
  - Requirements and specification documents
  - Design documents
  - Test suites and test plans

Hardware Failure Curve



Software failures



What Is Good Software?

- **Correct, correct, correct**
- **Maintainable** and easy to modify
- Well modularized with well-designed interfaces
- **Reliable and robust**
- Has a **good user interface**
- **Well documented**
  - Internal documentation for maintenance and modification
  - External documentation for end users
- **Efficient**
  - **Not wasteful of system resources**, cpu & memory
  - **Optimized data structures and algorithms**

Goodness Goals Conflict

- **All goodness attributes cost $s to achieve**
- Interaction between attributes
  - **High efficiency may degrade maintainability, reliability**
  - More complex user interface may degrade efficiency, maintainability, and reliability
  - Better documentation may divert effort from efficiency and reliability
- **Software engineering management has to trade-off satisfying goodness goals**

Legacy Software

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology.
- software must be enhanced to implement new business requirements.
- software must be extended to make it interoperable with other more modern systems or databases.
- software must be re-architected to make it viable within a network environment.

Software Myths

- Management myths
  - We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know.
  - My people do have state-of-the-art software development tools
  - If we get behind schedule, we can add more programmers and catch up – Mongolian Horde Concept

Software Myths…

- Customer myths
  - A general statement of objectives is sufficient to begin writing programs…we can fill in the details later.
  - Project requirement continually change, but change can be easily accommodated because software is flexible.

Software Myths…

- Developer myths
  - Once we write the program and get it to work, our job is done.
  - Until I get the program "running" I really have no way of assessing its quality.
  - The only deliverable for a successful project is a working program.

Need Different Approaches for Developing Large Software

- Need formal **management** of software production process.
- Formal & detailed statement of requirements, specification and design.
- Much more attention to modularity and interfaces.
- *Must be separable* into manageable pieces.
- Need version control.
- More emphasis of rigorous and thorough testing.
- Need to plan for long term maintenance and modification.
- Need much more documentation, internal and external.

Why Is Software Development Hard?

- **Changing requirements** and specifications
- **Inability to develop complete** and correct requirements
- **Programmer variability** and unpredictability
- **Communication and coordination**
- **Imprecise and incomplete requirements and specifications**
- **Inadequate** software **development tools**
- **Inability to accurately estimate** effort or time required
- Overwhelming complexity of large systems, more than linear growth in complexity with size of the system
- Poor software development processes
- **Lack of attention to issues of software architecture**

What is Software Engineering?

The science (& art) of building *high quality* software systems

- On **time**
- On **budget**
- With **correct** operation
- With **acceptable performance**

Software Engineering:

The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of Software

SE Framework Activities

- Communication
  - Requirement collection
- Planning
  - Specification
- Modeling
  - Requirement Analysis
  - Design
- Construction
  - Code generation
  - Testing
- Deployment
  - Release
  - Maintenance

Major Software Production Tasks

- **Requirements analysis:**
  - Analyze software system requirements in detail
- **Specification**:
  - Develop a detailed specification for the software
- **Design**:
  - Develop detailed design for the software data structures, software architecture procedural detail, interfaces
- **Coding**:
  - Transform design into one or more programming language(s)
- **Testing**:
  - Test internal operation of the system and externally visible operations & performance
- **Release**:
  - Package and deliver software to users
- **Maintenance**:
  - Error correction and enhancement after system

The Essence of Practice

- Polya suggests:
  1. *Understand the problem* (communication and analysis).
  2. *Plan a solution* (modeling and software design).
  3. *Carry out the plan* (code generation).
  4. *Examine the result for accuracy* (testing and quality assurance).

Understand the Problem

- *Who has a stake in the solution to the problem?* That is, who are the stakeholders?
- *What are the unknowns?* What data, functions, and features are required to properly solve the problem?
- *Can the problem be compartmentalized?* Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?* Can an analysis model be created?

Plan the Solution

- *Have you seen similar problems before?* Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- *Has a similar problem been solved?* If so, are elements of the solution reusable?
- *Can subproblems be defined?* If so, are solutions readily apparent for the subproblems?
- *Can you represent a solution in a manner that leads to effective implementation?* Can a design model be created?

Carry Out the Plan

- *Does the solution conform to the plan?* Is source code traceable to the design model?
- *Is each component part of the solution provably correct?* Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

Examine the Result

- *Is it possible to test each component part of the solution?* Has a reasonable testing strategy been implemented?
- *Does the solution produce results that conform to the data, functions, and features that are required?* Has the software been validated against all stakeholder requirements?