Software Engineering (CS 208)

Course Introduction and Motivation

Dr. Puneet Gupta

Logistics

Instructor: Dr. Puneet Gupta
E-mail: puneet@iiti.ac.in
Office: Room 411, POD-1A (Scandium Building)

TAs: Anup Kumar Gupta, Trishna Saikia, Ashutosh Dhamania

Prefix email subject by CS208

Marks distribution

Mid-sem examination	**30%**
Practical evaluation	**30%**
End-sem examination	**40%**

What about attendance?
LaTeX should be used for preparing tutorial solutions.

Exams will be conducted online by looking at the current scenario. Mode of examination will be decided later.

Logistics

Lab

- Form groups of maximum size of 5 students
- A list of projects will be provided
- Can propose and work on your own project idea after discussing with me
- Takeaway from previous courses: Better to perform all the actions before the deadline to avoid last minute problems. Deadlines will be communicated later.

Reading material from the relevant sources will be provided.

Kindly beware of the different notations, definations and terminologies prevalent in the literature of software engineering.

Possible to create tutorials into doubt clearing sessions or regular lectures.

Cheating and plagiarism without proper credit to the original source(s) will lead to strict punishments.

Objectives of Studying CS-208

- Understand and apply Software Engineering concepts
- Building complex software systems when requirements are frequently changing
- Designing high quality software system within time, budget and other constraints.
- **Acquire technical knowledge**
- Acquire managerial knowledge
- Understand the Software development life-cycle

Acquire Technical Knowledge

Learn about modeling at different phases of software lifecycle:

- Requirements Elicitation
- (Requirements) Analysis
- Architectural Design
- Object/Component Design
- Coding
- Testing

Learn about Traceability among Models

Programs versus Software Products

Programs	***Software Products***
• Usually small in size	• Large in size
• Author is sole user	• Multiple users
• Single developer	• Team of developers
• Lacks proper user interface	• Well-designed interface
• Lacks proper documentation	• Well documented & user-manual prepared
• Ad hoc development.	• Systematic development

Program: Single use single developer
Software: Multi-user multi-developer

What is a Software Product?

Software Products or software systems are delivered to the customer with the proper documentation that that describe how to install and use the system.
In certain cases, software products may be part of system products where hardware, as well as software, is delivered to a customer.

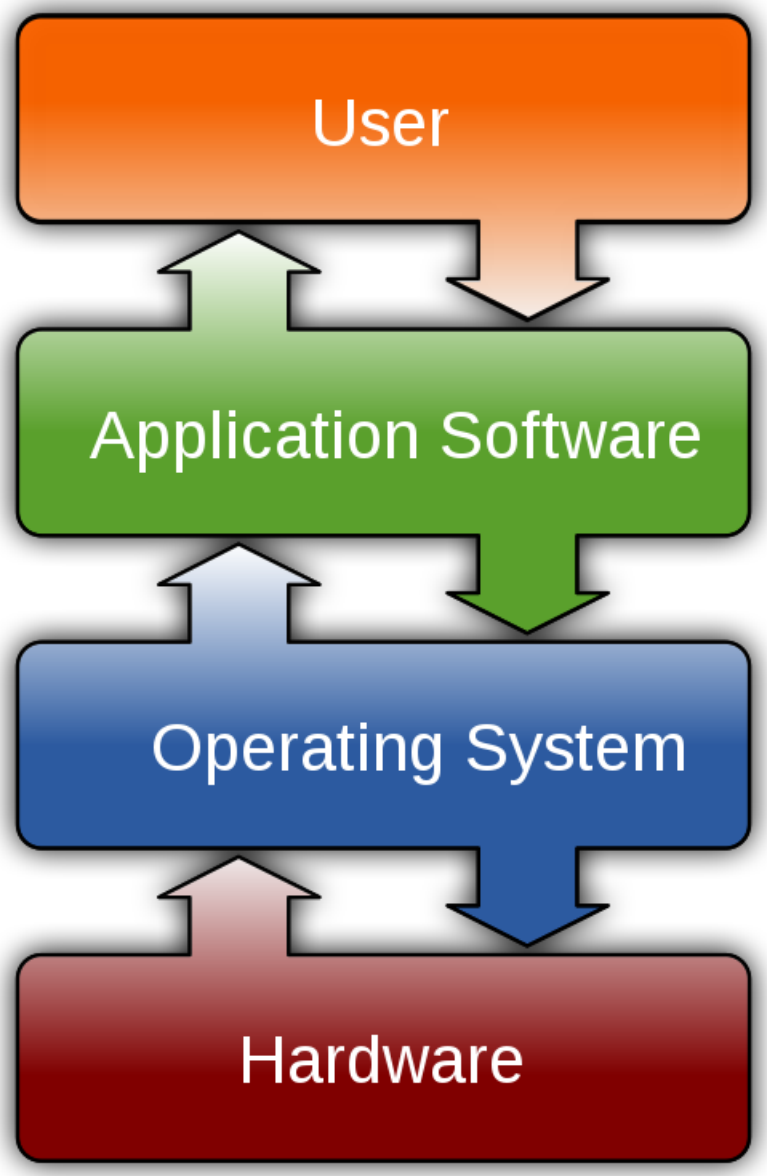
It consists of the following:

- ✓ Multiple separate programs
- ✓ Configuration files which are used to set up these programs
- ✓ System documentation which describes the structure of the system
- ✓ Developer and User documentation which explains how to use the system
- ✓ Web sites for users to download recent product information
- ✓ And so on...

Software: Computer programs and associated documentation such as requirements, design models and user manuals.
New software can be created by developing new programs, configuring generic software systems or reusing existing software.

How user interact with computer?

A diagram showing how the user interacts with application software on a typical desktop computer.The application software layer interfaces with the operating system, which in turn communicates with the hardware. The arrows indicate information flow.



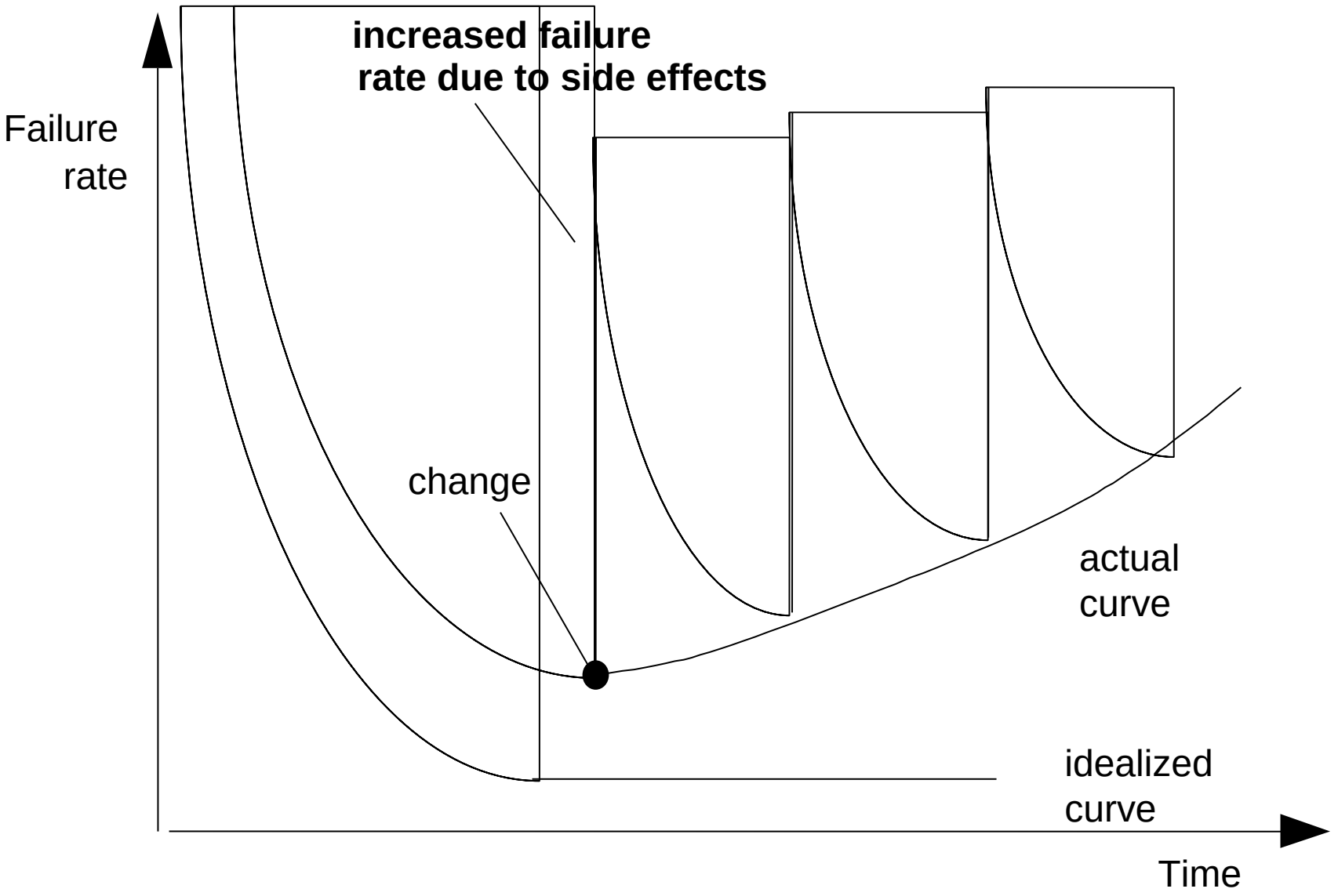
Source: https://en.wikipedia.org/wiki/Software

Software Vs Hardware

- You can’t see, touch, or feel software
- Software is *only* engineered, not manufactured
- Software doesn’t wear out
- Software is complex
- Software can behave like an aging factory

Tradeoff between hardware and software

Wear vs. Deterioration



Roger S. Pressman, Software Engineering: A Practitioner's Approach, Fourth Edition 1997

Categories of software

Software products may be developed for a particular customer or may be developed for a general market.

GENERIC

- developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
- referred to as commercial off-the-shelf (COTS) software

BESPOKE (custom)

- Developed for a single customer or organization according to their specification.
- Product specification controlled by the product developer
- Examples – embedded control systems, air traffic control software, traffic monitoring systems.
- Also known as services. Usually not developed from scratch. Often generic software is tailored. Eg. Fees collection systems.

Horizontal product (like OS, MS word) Vs vertical product (like banking systems)

Categories of software

Open-source software

- Developed and maintained by a team of volunteers
- May be downloaded and used free of charge
- Examples:
  - Linux operating system
  - Firefox web browser
  - Apache web server

Type of software

1. System software: such as compilers, editors, file management utilities
2. Application software**:** stand-alone programs for specific needs.
3. Engineering/scientific software: Characterized by algorithms such as automotive stress analysis, molecular biology, orbital dynamics etc
4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
5. AI software uses non-numerical algorithm to solve complex problems like Robotics, expert system, pattern recognition game playing

And so on…

Importance of software

Why Software is Important?

- ✓ The economies of ALL nations are dependent on software. More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment,)
- ✓ Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.

Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs. (will be discussed later.)

Some steps of software designing

Fundamental principles applicable to all types of software systems are:

Communication is the key

- Properly plan the development process
- Have clear ideas of aim and timelines
- Different type of software require different processes and quality attributes: Understand them

Understanding software attributes

- Software should behave as expected, without failures.
- Manage security issues
- Software should be efficient and should not waste resources.

- Manage software building process such that it can be delivered within budget and on-time.

- Use existing resources in the best possible way. Reuse existing software rather than write new software.

What are the attributes of good software?

- Efficiency: The software should not make wasteful use of system resources such as memory and processor cycles.
- Maintainability: It should be possible to evolve the software to meet the changing requirements of customers. It describes how easily a system can be modified.
- Dependability: It is the flexibility of the software that ought to not cause any physical or economic injury within the event of system failure. It includes a range of characteristics such as reliability, security and safety.
- In time: Software should be developed well in time.
- Within Budget: The software development costs should not overrun and it should be within the budgetary limit.
- Functionality: The software should exhibit the proper functionality, i.e, it should perform all the intended task.
- Adaptability: The software system should have the ability to get adapted to a reasonable extent with the changing requirements.
- Acceptability: Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.
- and so on….

What are the attributes of good software?

Transitional attributes are important when the software is moved from one platform to another:

- ✓ Portability: Capability of a program to be executed on various types of data processing systems without converting the program to a different language and with little or no modification
- ✓ Interoperability: Capability of two or more functional units to process data cooperatively
- ✓ Adaptability and so on...

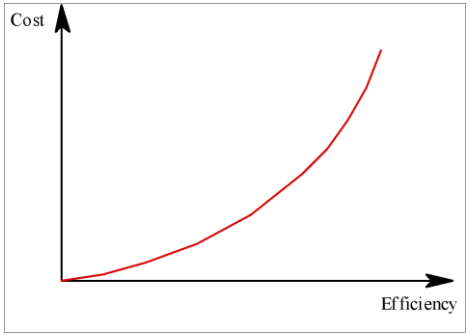
Difference between interoperability and portability from cloud perspective:

- ✓ Interoperability means the ability of two cloud systems to talk to another, i.e. to exchange messages and information in a way that both can understand.
- ✓ Data portability means the ability to move data (files, documents, database tables, etc.) from one cloud system to another, and have that data usable in the other system.
- ✓ Application Portability means the ability to move executable software from one cloud system to another, and be able to run it correctly in the destination system.

What are the attributes of good software?

Maintenance attributes are required to test how well a software has the capabilities to maintain itself in the ever-changing environment:

- ✓ Maintainability
- ✓ Adaptability
- ✓ Scalability and so on...



- The relative importance of these characteristics depends on the product and the environment in which it is to be used.
- In some cases, some attributes may dominate. Like, key attributes may be dependability and efficiency in safety-critical real-time systems.
- Costs tend to rise exponentially if very high levels of any one attribute are required.

What are the attributes of good software?

- From the Users Perspective
  - Functionality
  - Reliability
  - Efficiency
  - Usability
  - And so on...

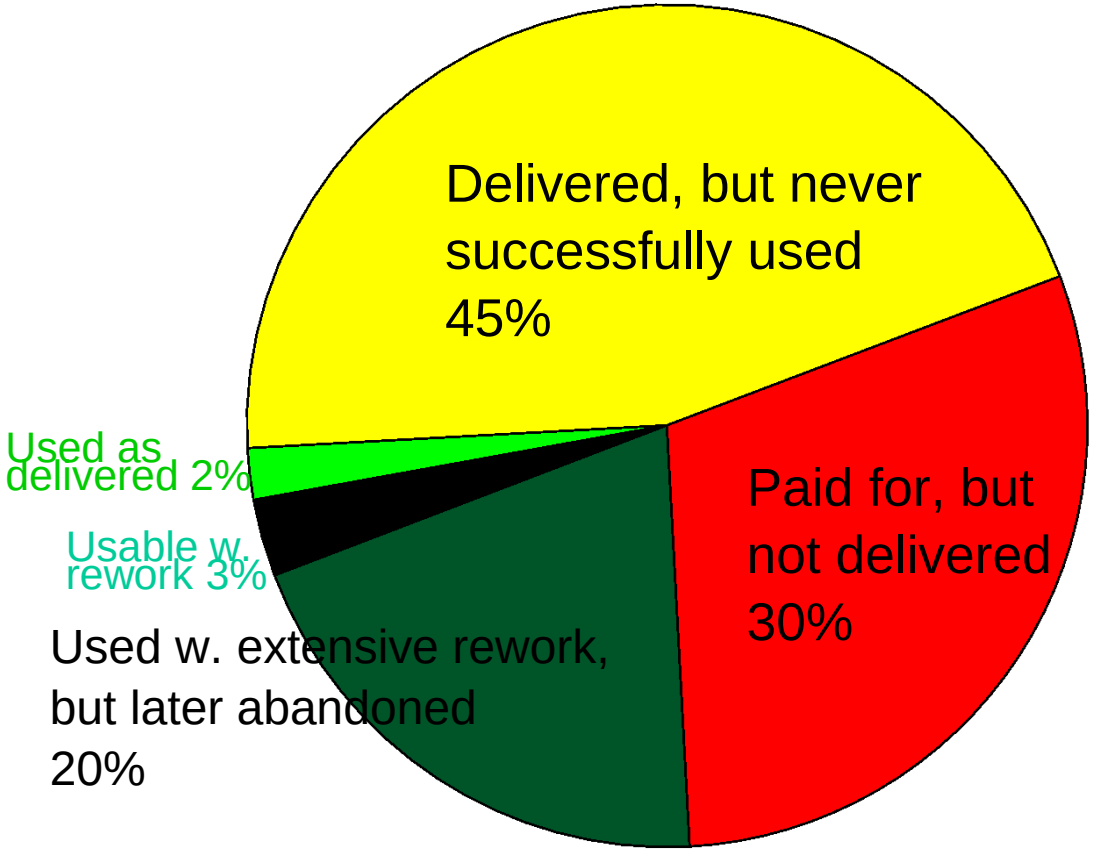
- From the Developers Perspective
  - Functionality
  - Understandability
  - Testability
  - Maintainability
  - Adaptability
  - And so on….

What is a software process?

- A set of activities whose goal is the development or evolution of software
- Generic activities in all software processes are:
  - Specification - what the system should do and its development constraints
  - Development - production of the software system
  - Validation - checking that the software is what the customer wants
  - Evolution - changing the software in response to changing demands

Software Crisis

9 software projects totaling $96.7 million: Where The Money Went
[Report to Congress, Comptroller General, 1979]



When it happen:

- fail to meet user requirements.
- frequently crash.
- expensive.
- difficult to alter, debug, and enhance.
- often delivered late.
- use resources non-optimally.

Take a look at the Standish Report (The “Chaos” Report)

Software Crisis

Figures in the late 90s, $250 billion annually in US is invested over 175,000 projects with varying complexity and try to incorporate lessons from past

Incredible pressure to develop systems that are:

- On time,
- Within budget,
- Meets the users' requirements

Result:

- At most 70% of projects completed
- Over 50% ran over twice the intended budget
- $81 billion dollars spent in cancelled projects!!

Getting better, but we need better tools and techniques!

Reasons for Software Crisis

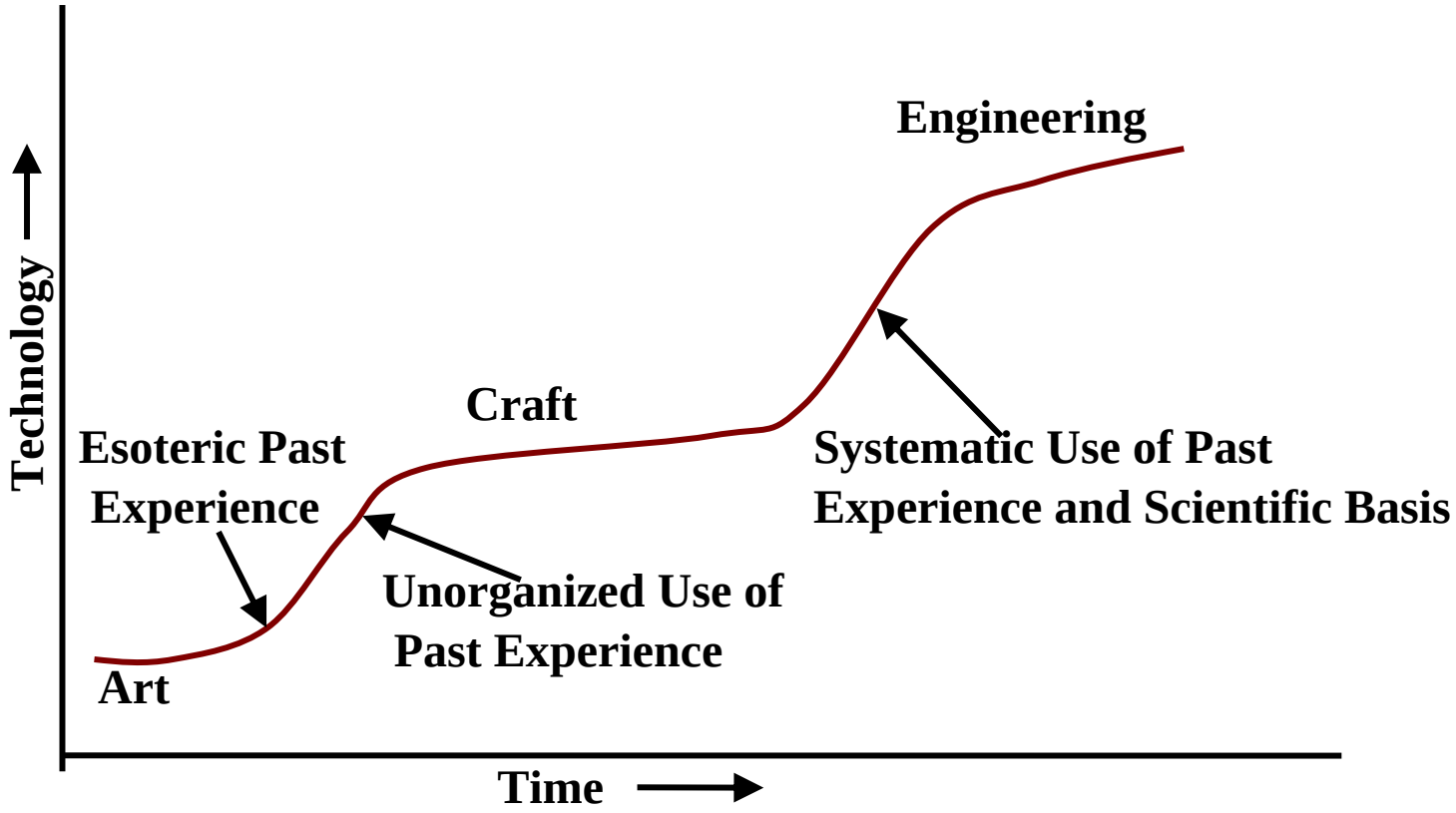
- (In)security: Allowing ourselves to become too dependent on software (and hardware) that was never designed to be robust or secure
- Over complexity: Competition for more features, ease of use, and integration are making products too large to comprehend and maintain
- Software patents: These legal "land mines" are beginning to choke the software industry
- Improper testing: Bugs spotted after deployment
- Maintenance ripple effect in evolutionary softwares
- Cost overruns: Faults in understanding clear functionality
- Team mismanagement
- Improper team selection
- Improper skill assessment of higher authorities
- Communication gap between customer and manager
- and so on....

Software failures

- **Therac-25 (1985-1987)**: six people overexposed during treatments for cancer
- **Taurus (1993)**: the planned automatic transaction settlement system for London Stock Exchange cancelled after five years of development
- **Ariane 5 (1996)**: rocket exploded soon after its launch due error conversion (16 floating point into 16-bit integer)
- **The Mars Climate Orbiter** assumed to be lost by NASA officials (1999): different measurement systems (Imperial and metric)
- **And so on...**

All these can be prevented by utilizing software engineering

Engineering Practice



- Heavy use of past experience: Past experience is systematically arranged.
- Theoretical basis and quantitative techniques provided.
- Many are just thumb rules.
- Tradeoff between alternatives: Revisit it later....
- Pragmatic approach to cost-effectiveness

What is software engineering?

“Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software.’’ [From Wikipedia]

“The establishment and use of sound engineering principles in order to obtain economically built software that is reliable and works efficiently on real machines.” [Classic Definition (1969)]

“Software Engineering: (1) The application of a systematic, disciplines, quantifiable approach to the development, operation, and maintenance of software; that is the application of engineering to software. (2) The study of approaches as in (1).” [IEEE Definition (1993)]

“A discipline whose aim is the production of fault-free software, delivered on-time and within budget, that satisfies the user’s needs. Furthermore, the software must be easy to modify when the user’s needs change.” [Schach]

“A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers.” [Ghezzi, Jazayeri, Mandrioli]

“Multi-person construction of multi-version software.” [Parnas]

Software Engineering: Definition

Software Engineering is a collection of techniques, methodologies and tools that help with the production of:

- a high quality software system
- with a given budget
- before a given deadline

while change occurs.

The process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints.

Constraints like meets user requirements, adaptable for easy maintenance, require less resources, and so on...

Analysis of the Definition

Systematic development and evolution

An engineering process involves applying well understood techniques in a organized and disciplined way

Many well-accepted practices have been formally standardized e.g. by the IEEE or ISO

Most development work is *evolutionary*

Large, high quality software systems

Software engineering techniques are needed because large systems cannot be completely understood by one person

Teamwork and co-ordination are required

Key challenge: Dividing up the work and ensuring that the parts of the system work properly together

The end-product that is produced must be of sufficient quality

Cost, time and other constraints

- Finite resources
- The benefit must outweigh the cost
- Others are competing to do the job cheaper and faster
- Inaccurate estimates of cost and time have caused many project failures

Why Software Engineering?

- **Software development is hard** !
- **Important to distinguish "easy" systems** (*one developer, one user, experimental use only*) **from "hard" systems** (*multiple developers, multiple users, products*)
- **Experience with "easy" systems is misleading**
  - ***One person techniques do not scale up***
- **Analogy with bridge building:**
  - **Over a stream = easy, one person job**
  - **Over a river … ? (*the techniques do not scale)***
- The problem is ***complexity***
- Many sources, but ***size*** is key:
  - **UNIX contains 4 million lines of code**
  - **Windows 2000 contains 10^8 lines of code**

Software engineering is about managing this complexity.

Why Software Engineering?

- To acquire skills to develop complex software systems
  - Exponential growth in complexity and difficulty level with size.
  - The ad hoc approach breaks down when size of software increases
  - **How to break large projects into smaller and manageable parts?**
- **It is usually cheaper, in the long run, to use software engineering rather than just write the programs. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.**
- **Learn effective techniques of specification, design, interface development, testing, project management, etc. Implement them to produce reliable and trustworthy systems economically and quickly. Remeber, more and more, individuals and society rely on software systems and thus, software crisis needs to be mitigated.**

What are the key challenges facing software engineering?

- Heterogeneity - platforms and execution environments
- Delivery – faster time to market
- Trust –includes reliability, security
- Shifts in economics of computing
  - Lower hardware costs and greater development and maintenance costs.
- Shifts in technology
  - Extensive networking
  - Availability and adoption of OO technology
  - Graphical user interfaces
- Budgets and costs
- Maintaining quality
- Legacy systems: Old, valuable systems must be maintained and updated
- Scale: Single user to global community.

Software Engineering: A Problem Solving Activity

Analysis: Understand the nature of the problem and break the problem into pieces

Synthesis: Put the pieces together into a large structure

For problem solving we use:

- Techniques (methods): Formal procedures for producing results using some well-defined notation
- Methodologies: Collection of techniques applied across software development and unified
- Tools: Instrument or automated systems to accomplish a technique

Large programming projects suffer management problems different in kind than small ones, due to division of labor.
Critical need is the preservation of the conceptual integrity of the product itself through designing appropriate guidelines and strictly following them.

Factors affecting the quality of a software system

Complexity:

The system is so complex that:

- no single programmer can understand it anymore
- introduction of one bug fix causes another bug

Change:

The “Entropy” of a software system increases with each change: Each implemented change erodes the structure of the system which makes the next change even more expensive (“Second Lehman's laws of software evolution”).

As time goes on, the cost to implement a change will be too high, and the system will then be unable to support its intended task. This is true of all systems, independent of their application domain or technological base.

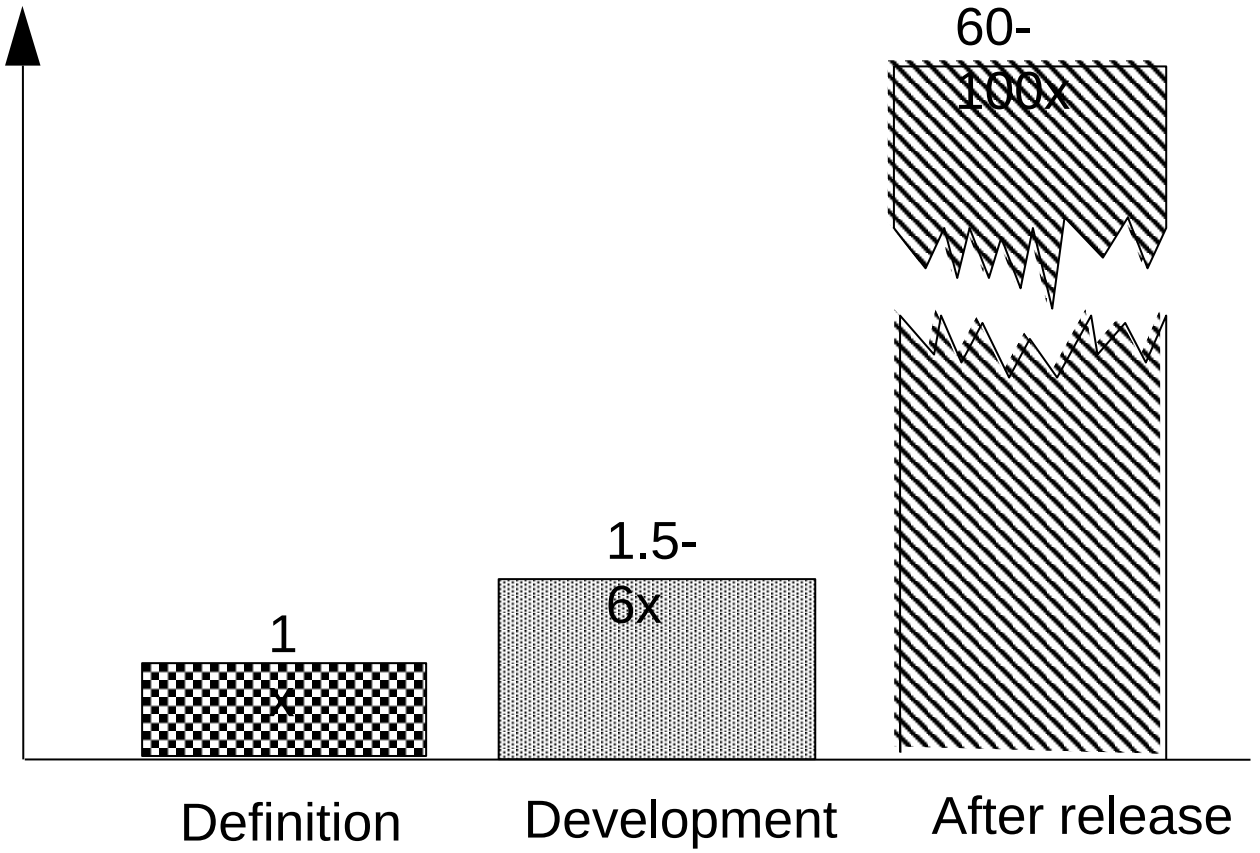
Brooks’s Law

Adding personnel to a late project makes it later

Postdelivery maintenance is extremely Costly

- To correct a fault early
  - Usually just a document needs to be changed
- To correct a fault later stages
  - Change the code and the documentation
  - Test the change itself
  - Perform testing
  - Reinstall the product on the client's computer(s)

The Cost of Change



An error that cost $40 to correct in design phase will cost $30 to correct in specification phase, and $2000 to correct in maintenance phase [Schach 2002]

Takeaway: We must try to find errors in early phases by conducting reviews and if possible indulge the user in every step by periodically validating the work.

Image credit: Roger S. Pressman, Chapter 1 Page 19, Software Engineering: A Practitioner's Approach, Fourth Edition 1997

What are software engineering methods?

- Structured approaches to software development which include system models, notations, rules, design advice and process guidance.
- Model descriptions
  - Descriptions of graphical models which should be produced;
- Rules
  - Constraints applied to system models;
- Recommendations
  - Advice on good design practice;
- Process guidance
  - What activities to follow.

Software Engineering vs. System Engineering

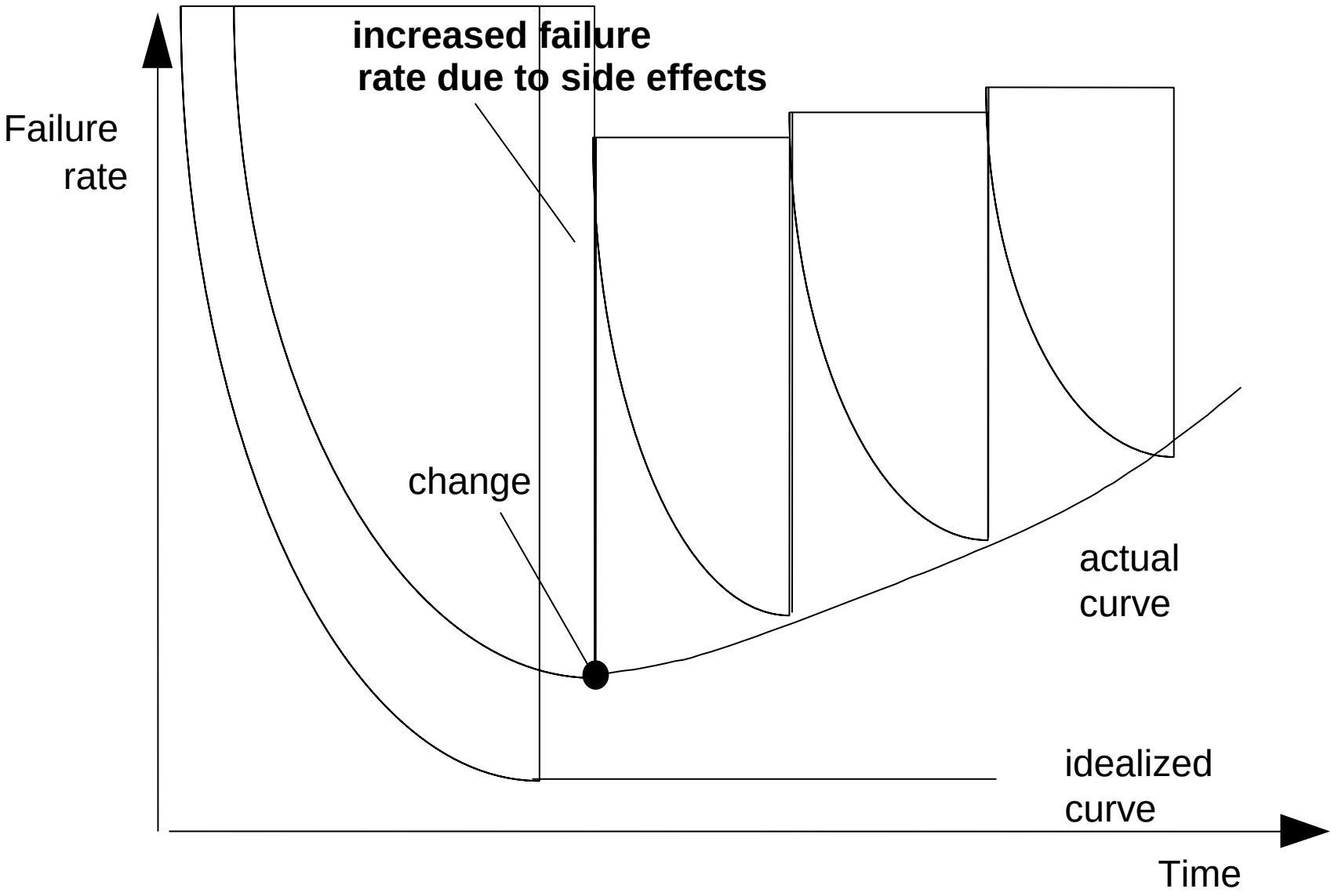
- System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering.
- Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system.

Revisiting: Software Vs Hardware

- You can’t see, touch, or feel software
- Software is *only* engineered, not manufactured
- Software doesn’t wear out
- Software is complex
- Software can behave like an aging factory

How these are performed?

Wear vs. Deterioration



Roger S. Pressman, Software Engineering: A Practitioner's Approach, Fourth Edition 1997

Engineering Process Model

- **Specification**: Set out the requirements and constraints on the system.
- **Design**: Produce a model of the system.
- **Manufacture**: Build the system.
- **Test**: Check the system meets the required specifications.
- **Install**: Deliver the system to the customer and ensure it is operational.
- **Maintain**: Repair faults in the system as they are discovered.

Software Engineering is Different

- Normally, specifications are incomplete.
- Very blurred distinction between specification, design and manufacture.
- No physical realization of the system for testing.
- Software does not wear out - maintenance does not mean component replacement.

Software Engineering Costs

- **Roughly 60% of costs are development costs, 40% are testing costs**. For custom software, evolution costs often exceed development costs

- **Costs vary depending on the type of system** being developed **and the requirements** of system attributes such as performance and system reliability

- **Distribution of costs depends on the development model that is used**

Software Engineering: Building Blocks

- Science: empirical studies; theories characterizing aggregate system behavior (e.g. reliability)
- Management: organizing teams, directing activities, correcting problems
- Human factors: user task understanding and modeling; knowledge of user interface design
- Engineering: tradeoffs, canonical solutions to typical problems
  - Tradeoffs and representative qualities
    - Pick any two:
      - Reliable, performance, cheap
      - Scalability, functionality, performance

Symptoms of Software Development Problems

- Inaccurate understanding of end-user needs
- Inability to deal with changing requirements
- Modules that don't fit together (integration)
- Software that's hard to maintain or extend (brittle)
- Late discovery of serious project flaws (integration)
- Poor software quality (architecture, risks unanticipated…)
- Process not responsive to Change
- Unacceptable software performance
- Team members in each other's way, unable to reconstruct who changed what, when, where, why (software architecture, …
- …and we could go on and on…

How to handle these problems?

- We need a process that
  - Will serve as a framework for large scale and small projects, that is, it is adaptive – embraces ‘change!’
  - Iterative (small, incremental ‘deliverables’)
  - Risk-driven (identify / resolve risks up front)
  - Flexible, customizable process (not a burden; adaptive to projects)
  - Architecture-centric (breaks components into ‘layers’ or common areas of responsibility…)
  - **Heavy** user involvement

- Identify best ways of doing things – a better process – acknowledged by world leaders…

Software Myths

Myth 1: Once we write the program and get it to work, our job is done.
Reality: the sooner you begin writing code, the longer it will take you to get done. 60% to 80% of all efforts are spent after software is delivered to the customer for the first time.

Myth 2: Until I get the program running, I have no way of assessing its quality.
Reality: technical review are a quality filter that can be used to find certain classes of software defects from the inception of a project.

Myth 3: software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.
Reality: it is not about creating documents. It is about creating a quality product. Better quality leads to a reduced rework. Reduced work results in faster delivery times.

Many people recognize the fallacy of the myths. Regrettably, habitual attitudes and methods foster poor management and technical practices, even when reality dictates a better approach.

Software Myths

- **Management myths**
  - *Standards and procedures for building software*
  - *Add more programmers if behind the schedule*
- **Customer myths**
  - *A general description of objectives enough to start coding*
  - *Requirements may change as the software is flexible*
- **Practitioner myths**
  - *Task accomplished when the program works*
  - *Quality assessment when the program is running*
  - *Working program the only project deliverable*

David Hooker's General Principles for Software Engineering Practice

Help you establish mind-set for solid software engineering practice (David Hooker 96).

1: *The Reason It All Exists: provide values to users*

2: *KISS (Keep It Simple, Stupid! As simple as possible)*

3: *Maintain the Vision (otherwise, incompatible design)*

4: *What You Produce, Others Will Consume* (code with concern for those that must maintain and extend the system)

5: *Be Open to the Future* (never design yourself into a corner as specification and hardware changes)

6: *Plan Ahead for Reuse*

7: *Think! Place clear complete thought before action produces better results.*

Thanks

Any Questions??