SCS1302:
Introduction to Software Engineering

Bachelor of Computer Science

1st Year

Semester 1

Prof. K. P. Hewagamage

UCSC

Ground Rules…

- Attendance – Theory and Practical
- UGVLE Course Page
  - Add a profile picture (Get to know members in the course)
  - Course Related Announcements
  - Participate activities
- How to ask questions
  - Direct Interaction (F2F)
  - slido.com (online interaction)

Course Aim:

- To provide a broad understanding of the software engineering process, concepts and the systematic development and management of software projects.
- explain the software engineering principles and techniques that are used in developing quality software products
- apply software engineering principles and techniques appropriately to develop a moderately complex software system

Learning Outcomes

After successfully completing this subject, students should be able to:

- **LO1**. Explain the software engineering principles and techniques in developing quality software products, such as the software development lifecycle, software quality, and software testing
- **LO2**. Apply software engineering principles and techniques appropriately in developing a moderately complex software system
- **LO3:** Use software engineering tools and technologies to support the development process
- **LO4:** Apply ethical and professional principles to software engineering practice.

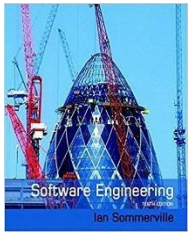
Outline of Syllabus

1. Introduction
2. Software Processes
3. Requirement Engineering
4. Agile Software Development
5. System modeling
6. Architectural design
7. Design and implementation
8. Software Testing
9. Software evolution

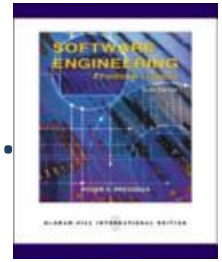
Software engineering

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenditure on software represents a significant fraction of GNP in all developed countries.

Main References

1. 

Software Engineering by Ian Sommerville, 10th edition, Addison-Wesley, 2015.

2. 

Software Engineering: A practitioner's approach by Roger S. Pressman, 9th edition, McGraw-Hill International edition, 2020.

UCSC

Software Engineering

1. Introduction to Software and Software Engineering

Intended Learning Outcomes

- Identify the problems associated with developing software
- Describe the need for a managed approach to software development
- Be able to define the term ‘Software Engineering’
- Identify software quality attributes and their classification

What is Software?

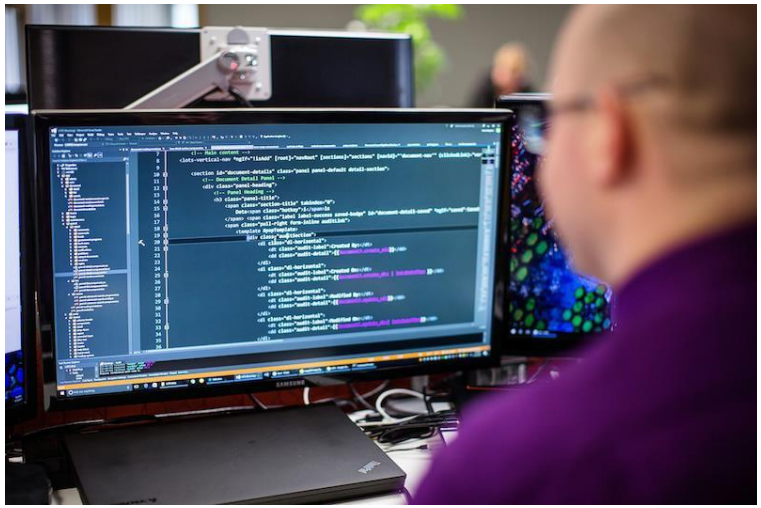
- Software is any set of machine-readable instructions to the computer's processor to perform specific operations.
- As a product, a Software system is a collection of intercommunicating components, programs, configuration files, system documentation and user documentation
- Software products may be developed for a particular customer or may be developed for a general market.
  - **Generic** - developed to be sold to a range of different customers
  - **Bespoke (custom)** - developed for a single customer according to their specification

Define - Software

Software is:

1) *Instructions (computer programs) that when executed provide desired features, function, and performance;*
2) *Data structures that enable the programs to adequately manipulate information.*
3) *Documentation that describes the operation and use of the programs.*

Software Projects vs. Other Projects



Software Projects vs. Other Projects

- The main difference in software engineering compared to other engineering disciplines are as follows;
  - It is difficult for a customer to specify requirements completely.
  - It is difficult for the developer to understand fully the customer needs.
  - Software requirements change regularly.
  - The environments to which software is made change regularly.
  - Software is primarily intangible; much of the process of creating software is also intangible, involving experience, thought and imagination (a brain product).
  - Because of intangibility it is difficult to specify software.
  - Customers and developers have communication gaps.

Software Projects vs. Other Projects

- It is difficult to test software exhaustively.
- Not like other engineering disciplines that have narrow scope, software can be developed in diverse contexts ranging from embedded systems to giant manufacturing systems.
- The replication of software is trivial and as a consequence software is elastic and gradually becomes more complex as changes are made over time.
- A small change to a few lines of code could do a big change in the overall behavior of the system.
- Finally, as a discipline, software development is so young that measurable, effective techniques are not yet available, and those that are available are not well-calibrated.

Questions about software

- Why does it take so long to get software finished?
- Why are development costs so high?
- Why can't we find all errors before we give the software to our customers?
- Why do we spend so much time and effort maintaining existing programs?
- Why do we continue to have difficulty in measuring progress as software is being developed and maintained?

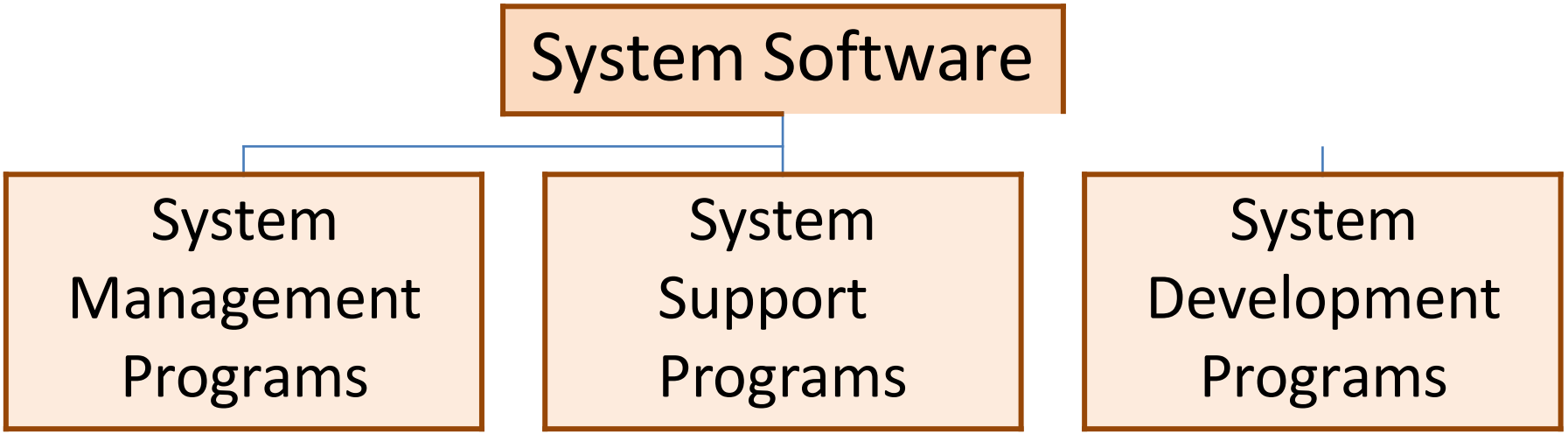
Main Types of Software

- There are two main types of software;

1. System Software
   - computer software designed to operate and control the computer hardware and to provide a platform for running application software

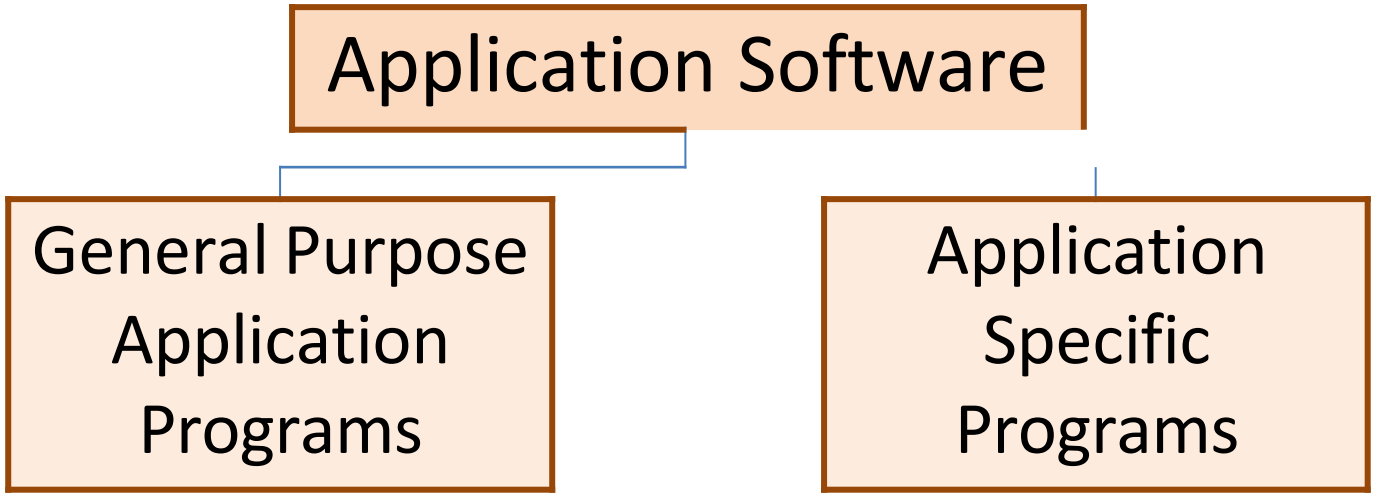
2. Application Software
   - set of one or more programs designed to carry out operations for a specific application

System Software



UCSC

Application Software



- Word Processing
- Electronic Spread Sheets
- Database Managers
- Graphics Software
- Integrated Packages

- Accounting, General Legers etc.
- Marketing-Sales Analysis etc.
- Manufacturing-Production Control etc.
- Finance-Capital Budgeting etc.

Common Software Types

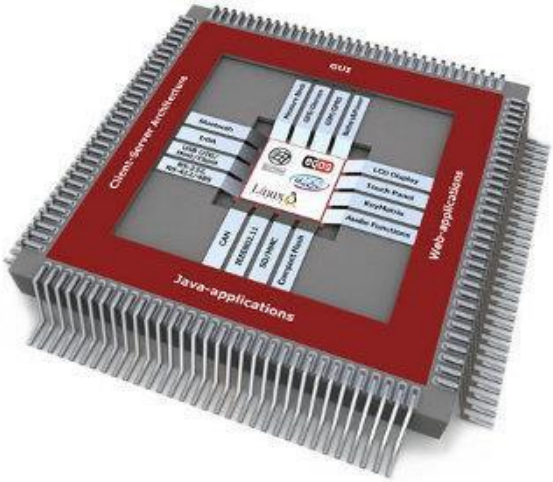
- **System Software:**
  - System software is a collection of programs is written to service the other programs.

Eg: Operating system component, drivers, telecommunication proces s

Common Software Types

- **Business software:**

  management information system software that access one or more large database containing business information



- **Embedded software:**

  Embedded software resides in read-only memory and is used to control product and system for the customer and industrial markets

UCSC

Common Software Types

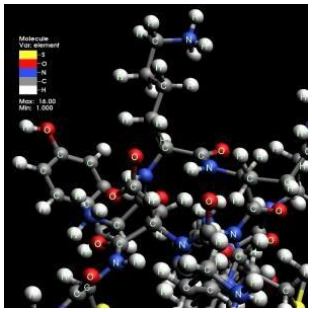
- **Web-based software:**

  The network becomes a massive computer providing an almost unlimited software resources that can be accessed by anyone with a modem.

- **Artificial intelligence software:**

  AI software makes use of non-numerical algorithms to solve the complex problems that are not amenable to computing or straightforward analysis.

Common Software Types



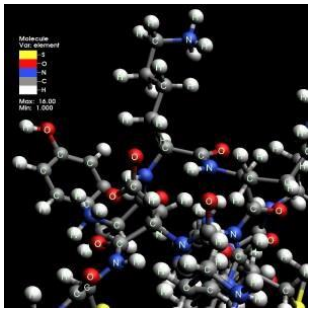
- **Personal computer software:**

  Personal computer software market has burgeoned over the past two decades.

  Word processing, spreadsheets, computer graphic, multimedia management

Common Software Types

- **Scientific or Engineering Software :**
- Used for modeling, simulation, or analysis in fields like physics, chemistry, and engineering.



- **Personal computer software:**

  Personal computer software market has burgeoned over the past two decades.

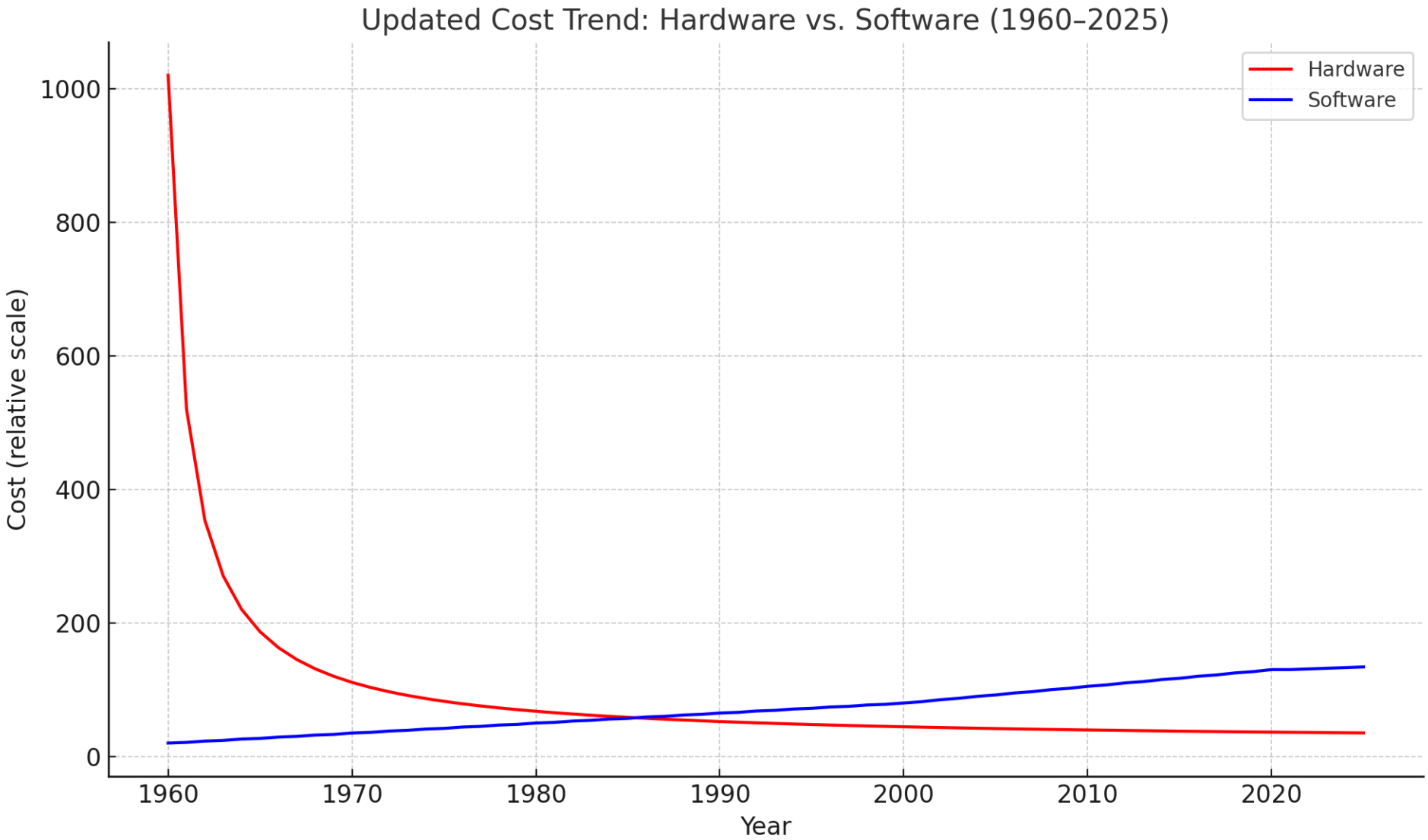
  Word processing, spreadsheets, computer graphic, multimedia management

Software Application Domains

- System software.
- Application software.
- Engineering/Scientific software.
- Embedded software.
- Product-line software.
- Web/Mobile applications.
- AI software (robotics, neural nets, game playing).

What is the legacy software?

Cost of Hardware vs. Software



What is the trend after 2020?

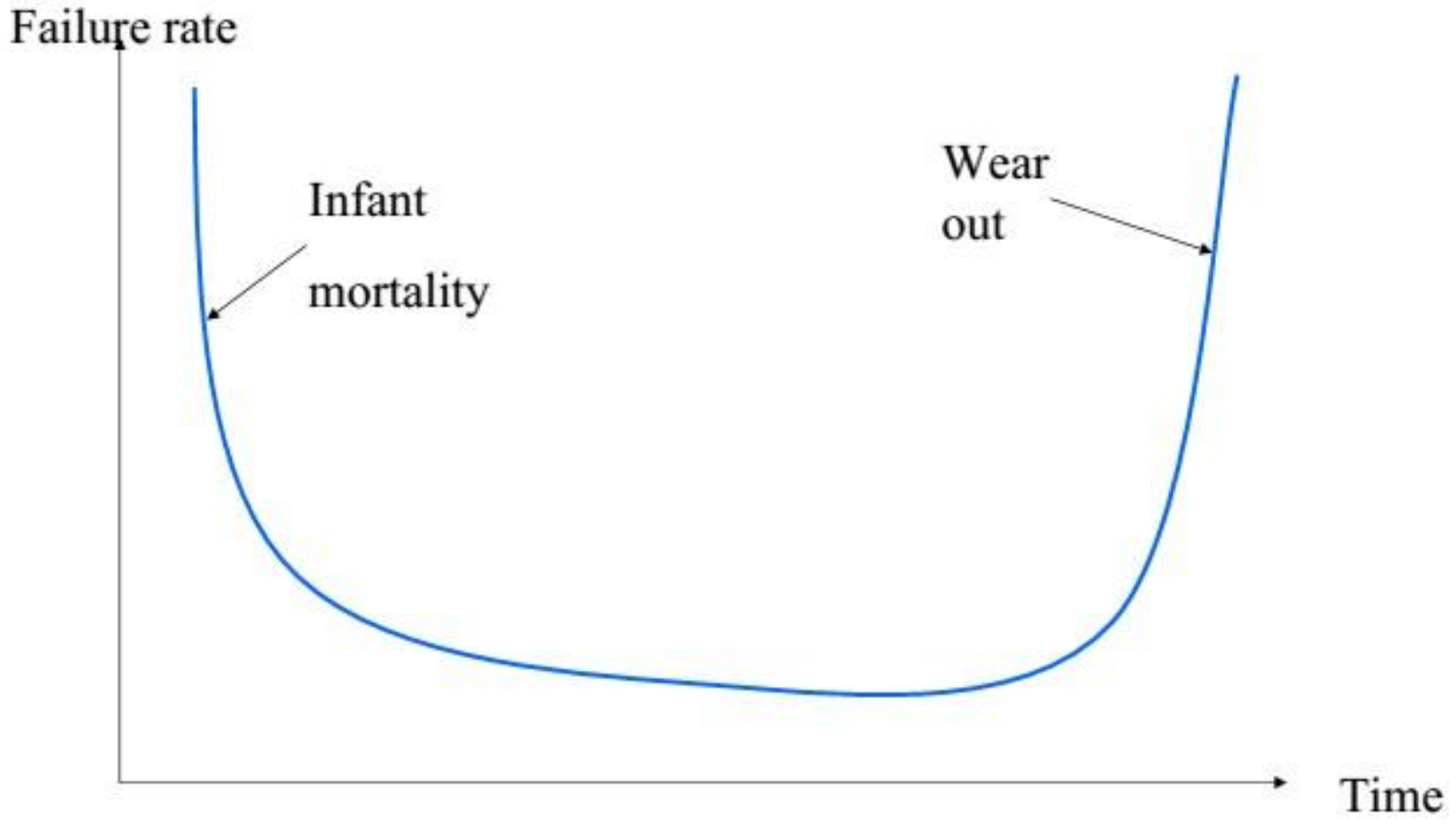
Software Costs : Why Software is More Expensive

- Software now dominates total system cost
  - For most modern systems, especially PCs and enterprise setups, the software licenses, services, and support outweigh the hardware costs.
- Maintenance > Development
  - Maintaining software (fixes, updates, upgrades, compatibility) can cost **2–5 times** more than the original development — especially in systems with long life cycles.
- Software Engineering is Cost Engineering
  - A major goal of software engineering is to make software that's reliable, maintainable, and scalable —

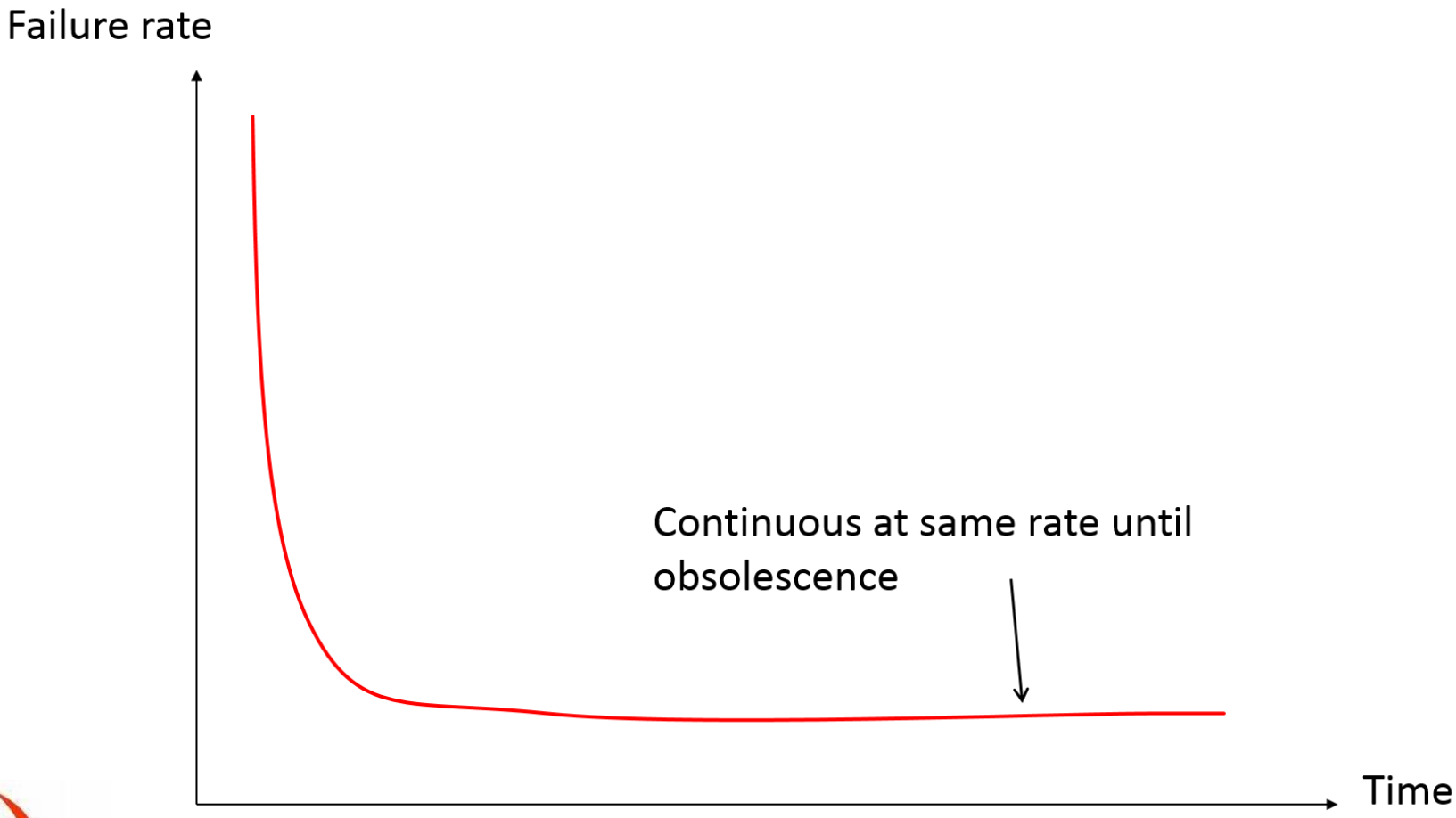
Failure Curves for Software vs. Hardware

- Software is engineered, not manufactured — no physical degradation
- Software doesn’t “wear out,” but it can still fail, especially when changed
- Hardware physically ages, while software logically degrades

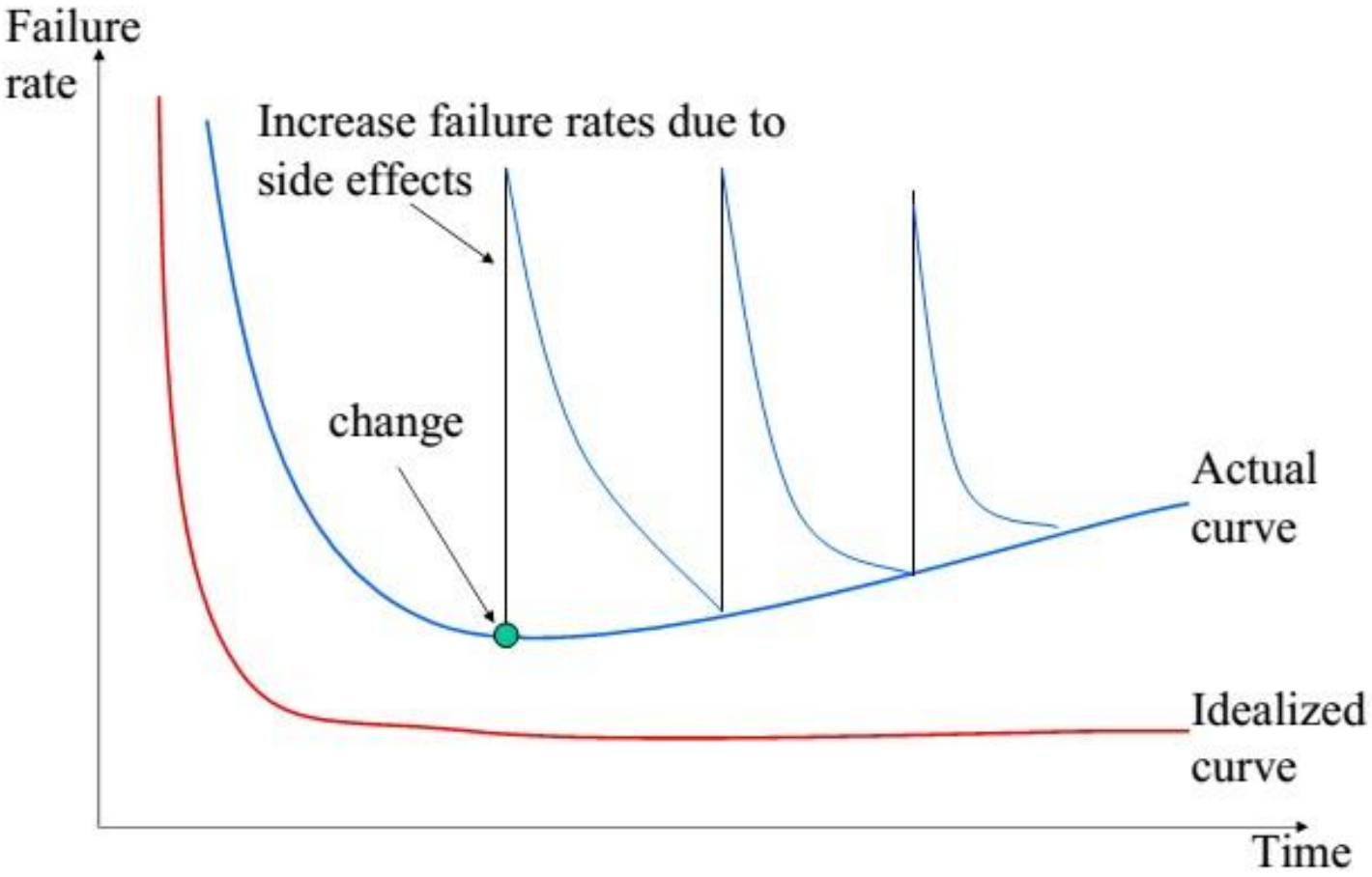
Failure “Bathtub” curve for hardware



Failure curve for software



Failure curve for software: in reality



Why Software Projects Fail

- Increasing System Complexity

Modern software must:

- Support **larger and more complex systems**
- Be delivered **faster**
- Enable **new capabilities** once thought impossible

Bigger dreams bring bigger risks.

- Lack of Software Engineering Discipline

Many teams:

- Write code without **structured engineering principles**
- Skip formal **design, documentation, and testing**
- Focus on speed over **quality and maintainability**

Result: Projects may be fast and cheap but are often **unreliable and hard to scale**.

What is Software Development?

- Software development is the process of designing, programming, testing, and maintaining software products. It involves more than just writing code — it includes planning, documentation, and ongoing improvement.

Development Methodology?

- A **software development methodology** is a structured approach that guides how software is planned, developed, and delivered. It defines the **phases**, **practices**, and **tools** used throughout the project.
- Examples: Waterfall, Agile, DevOps

- The development process is often viewed as a **life cycle**, from initial concept to retirement of the software. Using the right methodology helps manage complexity, improve quality, and ensure project success.

Software Development Life Cycle

- SDLC: Software Development Life Cycle has the following stages of software development:
  1. Requirements Gathering and Specification
  2. System Design
  3. Implementation
  4. Testing and Integration
  5. Deployment
  6. Maintenance

Members of a Software Project Team

Role	Responsibility
1. Project Manager	Oversees planning, timelines, budget, and team coordination
2. Systems Analyst	Gathers requirements, defines system specifications
3. Designer	Creates system architecture and user interface design
4. Programmer / Developer	Writes, builds, and integrates code
5. Tester (QA Engineer)	Tests software to ensure quality and functionality
6. Technical Clerk / Support Staff	Manages documentation, updates records, assists with administrative tasks

Project Success Criteria

- Deliver the software to the customer at the agreed time.
- Keep overall costs within budget.
- Deliver software that meets the customer's expectations.
- Maintain a happy and well-functioning development team.

What Defines Project Success?

- Deliver on Time

  Ensure the software is delivered to the customer at the a greed schedule or milestone.

- Stay Within Budget

  Manage resources and scope to prevent cost overruns a nd optimize value.

- Meet Customer Expectations

  Deliver a product that is functionally complete, bug-free, and aligned with the customer's goals.

- Maintain a Healthy Team

  A happy, motivated, and collaborative team I ncreases l ong-term success and sustainability.

Why Software Projects Fail

Planning Failures

?

Communication Issues

?

Process & Technical Weaknesses

?

External Risk Factors

?

“Failure rarely comes from a single mistake — it’s usually a combination of bad planning, poor communication, weak technical discipline, and pressures from outside the team.”

Software Quality Management

Why quality management is important in software ?

How quality is managed in the software?

What are product qualities and process qualities?

Software Quality Management

Important?

How is Quality Managed?

Product vs. Process Qualities

Type	Description	Examples
Product Quality	Characteristics of the delivered software	Usability, efficiency, reliability
Process Quality	Characteristics of the development process	Adherence to standards, documentation, version control

"Think of product quality as how good the cake is, and process quality as how clean and organized the kitchen was. You need both to get consistent results."

Real World Statistics

IBM Survey, 2000

- 55% of systems cost more than expected
- 68% overran the schedules
- 88% had to be substantially redesigned

Bureau of Labour Statistics (2001)

- for every 6 new systems put into operation, 2 cancelled
- probability of cancellation is about 50% for large systems
- average project overshoots schedule by 50%

Software Project Success & Failure Rates

📌 **Successful** = On time, on budget, with full scope

⚠️ **Challenged** = Delivered but over budget, late, or missing features

❌ **Failed** = Cancelled or never used

Latest Reality (2023–2025 range)

Status	Approx. %
Successful	35–45%
Challenged	40–50%
Failed	10–20%

UCSC

What Changed Since Early 2000s?

- **Better tools** (e.g., Jira, Git, CI/CD)
- **Agile and DevOps adoption**
- **Improved communication (Slack, Zoom)**
- **Cloud platforms reducing infrastructure risks**

These advances **increased success** and **reduced failure**, but challenges like scope creep, unrealistic expectations, and unclear requirements still persist

Case Study -Development Failures

Over Budget

Home Office IT project millions over budget

Home Office (UK) IT project run by Bull Information Systems is expected to blow its budget by millions of pounds and is hampered by a restrictive contract, according to a leaked report. The National Audit Office Report is expected to reveal damning evidence that the project to implement two systems – the National Probation Service Information System, and the Case Record and Management System will cost 118m pounds by the end of the year, 70% over its original budget.

www.computing.co.uk/News/111627

Over Schedule

New air traffic system is already obsolete

National Air Traffic Services (*Nats*) is already looking at replacing the systems at its new control center at Swanwick in Hampshire, even though the system doesn't become operational until next week. This project is six years late and 180m pounds over budget. Swanwick was originally meant to be operational by 1997, but problems with the development of software by Lockheed Martin caused delays, according to *Nats*.

www.vnunet.com/News/1128597

UCSC

Software Crisis

- The term was coined in early 70's
- The causes of the software crisis were linked to the overall complexity of hardware and the software development process.
- The crisis manifested itself in several ways:
  - Projects running over-budget.
  - Projects running over-time.
  - Software was very inefficient.
  - Software was of low quality.
  - Software often did not meet requirements.
  - Projects were unmanageable and code difficult to maintain.
  - Software was never delivered.

Understanding the Software Crisis

The term “Software Crisis” was coined in the **early 1970s** (formally raised at the 1968 NATO Conference) to describe the growing problems in software development as systems became larger and more complex.

- Hardware was advancing rapidly (cheaper, faster), but software couldn’t keep up.
- Developers lacked structured methods to manage large codebases.
- There was no formal software engineering discipline yet — just ad hoc coding.

Symptoms of the Software Crisis

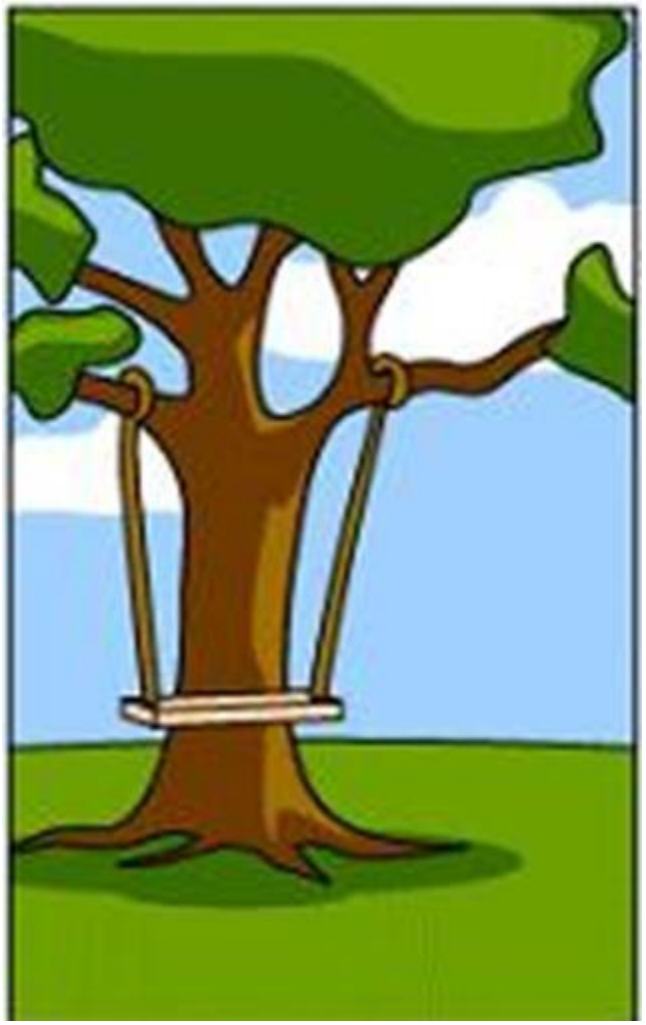
Symptom	Impact
Over-budget and over-time projects	Financial and resource strain
Inefficient and buggy software	Poor performance, user frustration
Failure to meet user requirements	Abandoned or rejected systems
Difficult to maintain and scale code	Long-term instability
Complete project failure (not delivered)	Total loss of investment

The crisis exposed the need for **software engineering as a discipline**, just like civil or electrical engineering — with processes, testing, modeling, and design principles.

A typical software project:

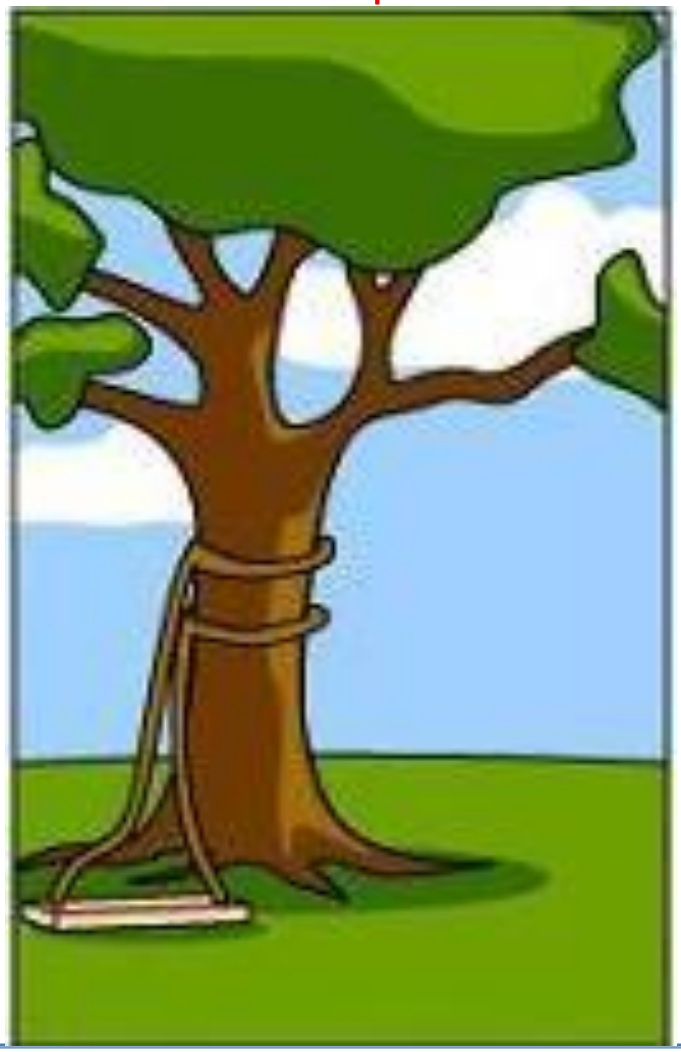
How the Customer Described it

How the Business Analyst Understood it



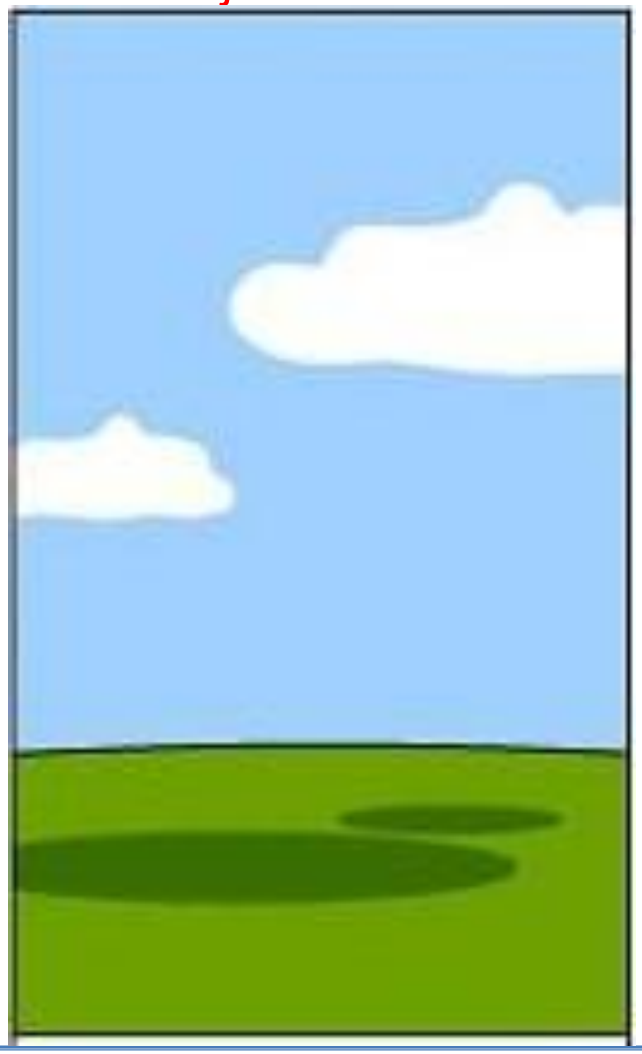
How the Architect Designed it

How the Developers coded it



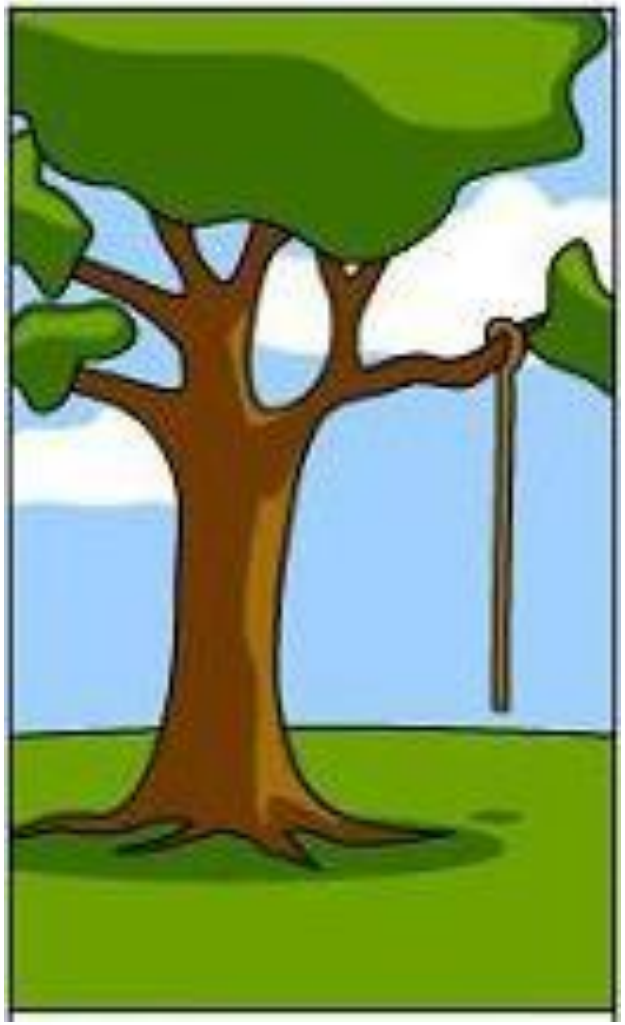
How the Marketing team described it to the customer

How the Project was Documented



UCSC

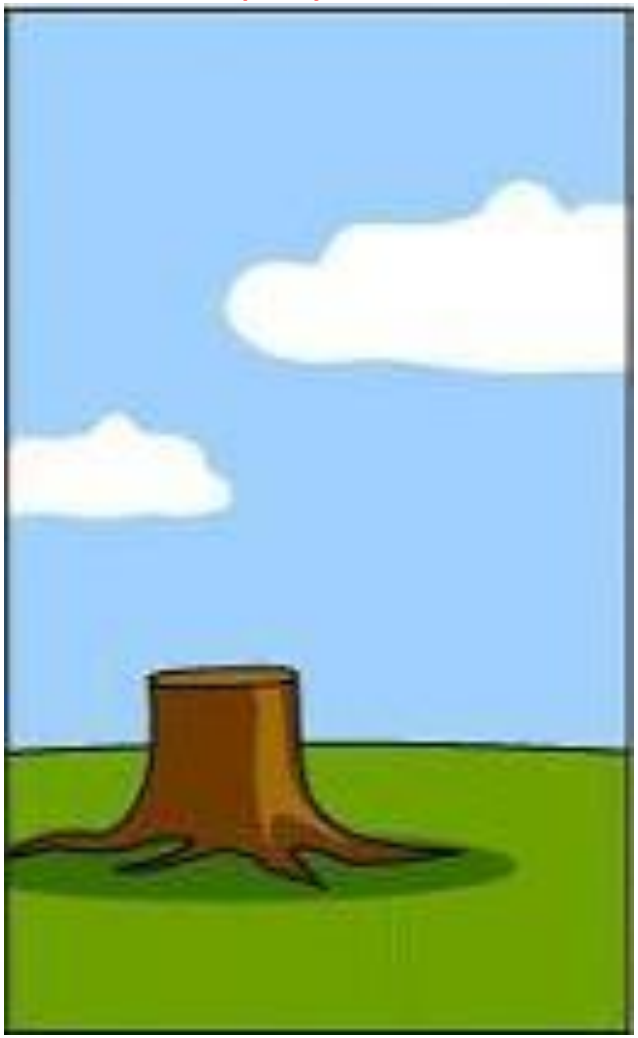
How it was Deployed at Customer Site



How the Customer was Billed

UCSC

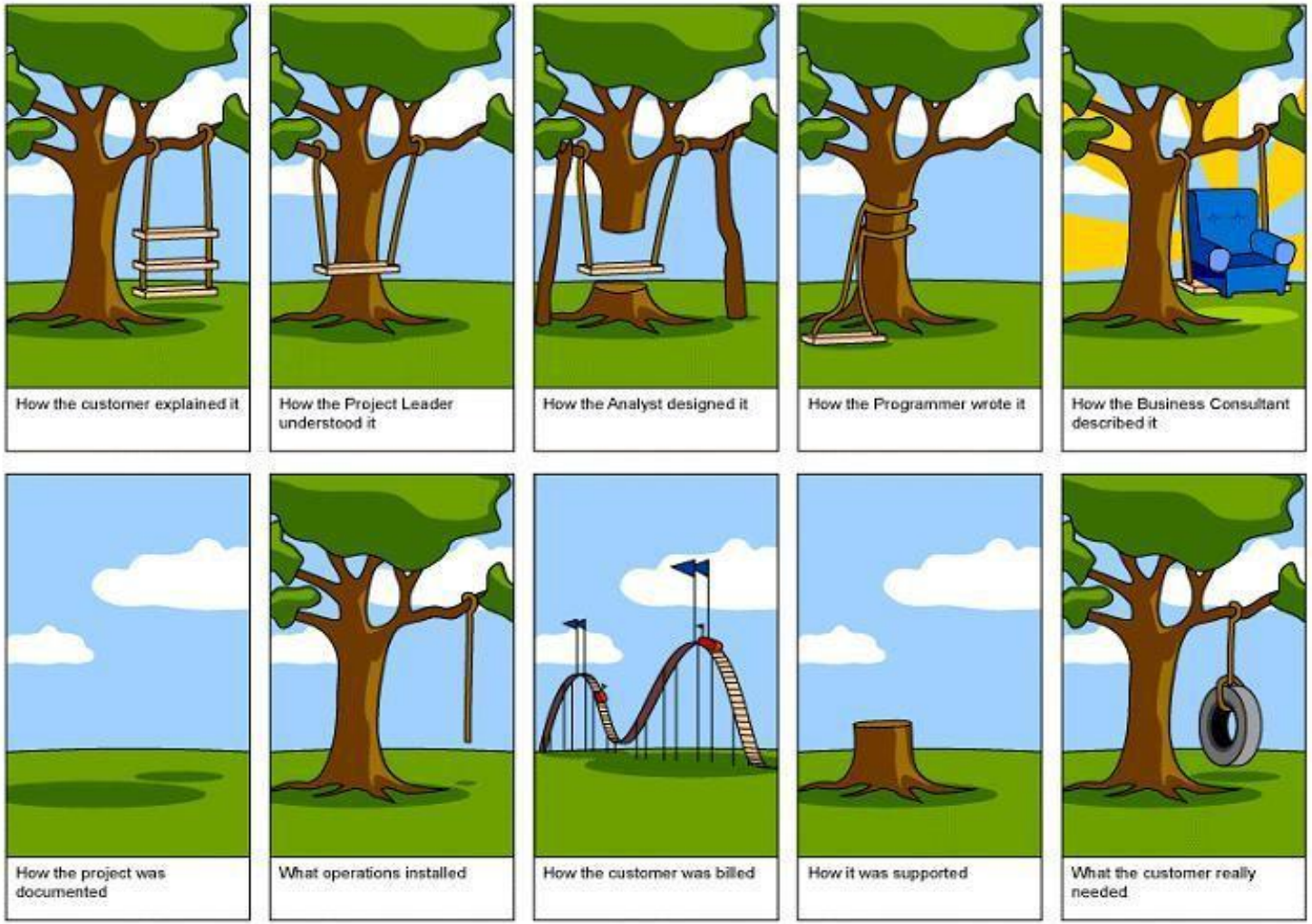
How it was Supported after Deployment



What Customers Really wanted!

UCSC

How the Swing Project happened!!!



UCSC

The Solution: Software Engineering

- Software engineering is concerned with theories, methods and tools for professional software development
- Greater emphasis on systematic , scientific development (engineering) of Software products
- Usage in computer assistance in software development (CASE)

Software Engineering —
The Solution to the Software Crisis

What Is Software Engineering?
A discipline that applies **scientific principles**, **engineering practices**, and **tool support** to develop **high-quality, maintainable** software.

Area	What It Contributes
🎯 **Clear Requirements**	Understanding user needs precisely through formal/semi-formal methods
🧪 **Prototyping**	Demonstrating early system versions to reduce risks and gain feedback
🛠️ **Engineering Process**	Structured models (e.g., Waterfall, Agile) for planning and control
🔍 **Quality Focus**	Emphasis on readable, error-free code that meets expectations
💻 **Tool Support (CASE tools)**	Automated support for design, testing, documentation, and maintenance

Software Engineering – Definition

Standard Definition (IEEE 610.12):

"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software."

- Systematic = planned and structured
- Disciplined = follows defined principles and standards
- Quantifiable = measurable outcomes (not guesswork)

Software engineering is the science and art of building reliable, high-quality software in a structured and predictable way.

- Prevents software crises
- Reduces cost, bugs, and delays
- Makes code maintainable, reusable, and scalable
- Ensures professional practice (like civil/mechanical engineering)

Software Engineer vs Developer

Software Engineer vs Developer – What's the Difference?

Role	Focus	Scope	Thinking Style
Software Developer	Focuses on coding and building features	Project/task level	**Tactical** – implement what's given
Software Engineer	Applies engineering principles to system design	System-wide / architectural	**Strategic** – plan, model, optimize

<u>Full Stack Developer?</u>

The left image shows a **Full Stack Developer** — someone who codes both frontend (UI/UX) and backend (servers, APIs). That's a **specialized developer** role — not necessarily an engineer, but highly skilled.

Top Skills for Software Engineers

1. Problem Solving

2. Attention to Detail

3. Clear Communication

4. Continual Learning

5. Teamwork

6. Empathize with End Users

UCSC

What makes a great engineer beyond just coding?

1. 🧠 **Problem Solving**

“At the core of engineering is problem solving — not just writing code, but finding the *right* solution for the *right* problem.”

2. 🔬 **Attention to Detail**

“One semicolon in the wrong place can crash a program. Precision is power.”

3. 🗣️ **Clear Communication**

“You’ll often explain technical ideas to non-technical people — clarity builds trust.”

4. 📚 **Continual Learning**

“Tech changes fast. You don’t stop learning after graduation. Every tool or language you know today might be outdated tomorrow.”

5. 🤝 **Teamwork**

“Modern software is built in teams — collaboration tools, code reviews, pair programming are all part of it.”

6. ❤️ **Empathize with End Users**

“User-centered thinking helps you build systems people actually want to use.”

Why Is Software Different from Other Engineering Fields?

#	Challenge	Why It Matters / Real-World Analogy
1	**Hard to specify completely**	Customers don’t know what they want until they see it
2	**Hard to fully understand user needs**	Developers and users speak “different languages”
3	**Requirements change frequently**	Like building a house where the design keeps evolving
4	**Software is intangible**	You can’t “see” a bug like a broken bridge — it’s mental
5	**Exhaustive testing is nearly impossible**	You can’t test all paths, inputs, and edge cases manually

Cost Distribution in the Software Development

Maintenance typically consumes **60–80%** of the total life cycle cost.

Phase	Approximate Cost %
Development (requirements → delivery)	**20–40%**
Maintenance (post-delivery)	**60–80%**

Source: IEEE Standard 1219, various Standish Group & Capers Jones reports

Why Maintenance Costs More:

1. 🛠️ **Bug fixes** after deployment
2. 🔄 **Feature changes** due to evolving needs
3. ⚙️ **Adapting** to new hardware, OS, browsers, etc.
4. 📉 Poor documentation or code quality increases cost over time
5. 👥 Often handled by different teams unfamiliar with original design

"The moment you ship your software is when the real cost begins. Writing code is just the beginning — keeping it running, secure, relevant, and bug-free is where the real money goes."

Why Software Development Is So Challenging?

1. Inherent Software Complexity
2. Constant Change & Uncertainty
3. Human Factors & Communication Barriers
4. Project Management Challenges
5. Project-Specific Characteristics That Complicate Things

Professional & Ethical Responsibility in Software Engineering

- Software engineering goes beyond technical skills — it's about trust, safety, and public good.
- Engineers must act **honestly**, **transparently**, and with **accountability**.
- **Ethics > Law** — Just because something is legal doesn't make it ethical.

"What you build could affect thousands — or millions. That's why professionalism matters."

UCSC

Common Ethical Responsibilities of Software Engineers

Ethical Area	What It Means
🔒 **Confidentiality**	Respect sensitive data — even without an NDA
🧠 **Competence**	Don’t accept work you’re not qualified to do
📘 **Intellectual Property**	Respect copyright, licensing, and patents
🖥️ **Computer Misuse**	Don’t abuse access (e.g., installing malware, misusing employer resources)

“Ethics is doing the right thing — even when no one is watching.”

UCSC

ACM/IEEE Code of Ethics

- Created by two major global organizations: **ACM** and **IEEE-CS**
- Provides a shared professional standard
- Contains **8 guiding principles**, e.g., public interest, lifelong learning, quality of life, leadership, competence

"The Code of Ethics helps align engineers' actions with society's expectations."

The ACM/IEEE Code of Ethics

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

Ethical principles

1. **PUBLIC** - Software engineers shall act consistently with the public interest.

2. **CLIENT AND EMPLOYER** - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3**. PRODUCT** - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4**. JUDGMENT** - Software engineers shall maintain integrity and independence in their professional judgment.

5. **MANAGEMENT** - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. **PROFESSION** - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. **COLLEAGUES** - Software engineers shall be fair to and supportive of their colleagues.

8**. SELF** - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

Core Principles of Software Engineering

These principles apply to **all types of software**, no matter the tools, languages, or methodologies.

1. Use a Defined Process

Software must be developed using a **well-managed, structured** process.

Different processes fit different systems — no one-size- fits-all.

2. Ensure Dependability & Performance

Every system must be:

Reliable (no unexpected failures)

Secure (especially in today's world)

Efficient (good performance, even under load)

Core Principles of Software Engineering …

3. Start with Clear Requirements

Know what to build before writing code. Requirements engineering helps avoid wasted effort and poor alignment.

4.Reuse When Possible

Don’t reinvent the wheel — reuse components, libraries, and code whenever appropriate.

Modern Software is Built for the Web & the Cloud

Web-Based Systems

- The **Web is the new platform**: Most apps run via browsers, not local installs.
- Web-based apps = **accessible anywhere**, **cross-platform**, and **easier to update**.

Web Services

- Applications expose **functionality via APIs** over the web (e.g., REST, GraphQL).
- Enables **integration** with other services (payment, maps, social logins).

Cloud Computing

- Software runs on **remote servers**, not personal machines.
- Examples: Google Docs, Zoom, Canva, Microsoft 365.
- Cloud platforms = AWS, Azure, Google Cloud.

Pay-as-you-go Model (SaaS)

- Users **don't buy software outright** — they **subscribe** (e.g., Netflix, ChatGPT).
- Businesses benefit from **scalability, lower upfront costs, and remote access**.

"Today, software is a *service*, not a product. You don't install Photoshop — you access it. That's web-based software engineering."

Web-based Software Engineering

- Web-based systems are **complex, distributed systems** running over the internet.
- Despite their complexity, they follow the **same software engineering principles** (requirements, design, implementation, testing, and maintenance).

- "Web-based systems demand fast, scalable, and modular design, but they are built on the same core engineering values that apply to all software systems — with more agility, service reuse, and richer user interfaces."

Key Engineering Practices

1. Software Reuse

Web systems often **reuse components** (libraries, APIs, services) instead of building everything from scratch.

2. Incremental & Agile Development

Requirements change frequently. So, these systems are built **iteratively** using **Agile methods**.

3. Service-Oriented Architecture (SOA)

Components are designed as **web services** that can function independently (e.g., login service, payment gateway).

4. Rich Interfaces

Technologies like **AJAX**, **React**, and **HTML5** allow creation of **dynamic and interactive UIs** that work inside the browser.

FAQs

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and 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 more general process.

FAQs

Question	Answer
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service- based systems. Web-based systems development has led to important advances in programming languages and software reuse.

UCSC