Software Engineering Approach

Lecture – 1

Introduction to Software Engineering Approach

(CSE 4658)

Course Details

- Credit: 4; Grading Pattern: 6
- Software Engineering

What Is Software Engineering? Working Well on Teams, Knowledge Sharing, Engineering for Equity, How to Lead a Team, Leading at Scale, Measuring Engineering Productivity, Style Guides and Rules, Code Review, Documentation, Testing Overview, Unit Testing, Testing Doubles, Larger Testing, Deprecation, Version Control and Branch Management, Code Search, Build Systems and Build Philosophy, Static Analysis, Dependency Management

Text: Software Engineering at Google Lessons Learned from Programming Over Time: Titus Winters, Tom Manshreck, and Hyrum Wright, O'REILLY

Evaluation

INTERNAL	
PARTICIPATION	5
WEEKLY ASSIGNMENTS	20
MID TERM	15
INTERNAL TOTAL	40
EXTERNAL	
FINAL ASSIGNMENT	40
ASSIGNMENT PRESENTATION	20
EXTERNAL TOTAL	60


AT&T 3G
7:42 PM
Saturday
31
Messages
Calendar
Photos
YouTube
Stocks
NYTimes
Calculator
App Store
Compass
Phone
Mail
Safari
iPod

Chapter-1

What Is Software Engineering?

Organization of Chapter-1

1. Software Engineering vs. Programming
2. Software Characteristics
3. Software Failure Rate
4. Software Engineering - defining
5. Time and Change
6. Hyrum's Law
7. Timeline of the developer workflow
8. Tradeoffs and Costs
9. Evolution of Pattern of Technology
10. Types of Software
11. Stakeholders in Software Engineering
12. Software Quality
13. Software Crisis

Software Engineering vs. Programming

- **Software engineering** is the comprehensive discipline of applying engineering principles to the design, development, testing, and maintenance of software systems.
- **Programming** is the specific act of writing, editing, and updating code to implement the designs of software engineering.
- A software engineer works on the big picture, acting as an architect who oversees the entire software lifecycle, whereas a programmer/developer focuses on the technical details of translating blueprints into functional code.

Software Engineering vs. Programming

Programming is about writing code. You take a task and write code to solve it.
Software engineering is when you take that piece of code and consider:

- How will this task evolve?
- How will this code adapt to those changes?
- What does this code encourage others to do?
- How does this code encourage other programmers to use it?
- How will I understand this code in 5 months?
- How will a busy team member jumping around grok this?
- What happens when the business becomes bigger?
- When will this code stop being good enough?
- How does it scale?
- How does it generalize?
- What hidden dependencies are there?
- considering the long-term effects of your code. Both direct and indirect.
- You always face the underlying concern of *"What's the expected lifespan of this code?"*.

Example: Online Food Ordering System

Programming mindset:

A developer writes a small program that:

- Lets a user select a food item from a menu.
- Calculates the total bill.
- Displays “Order placed” message.

This works perfectly for a single restaurant and a few users.

What will be the mindset of Software Engineer?

Software engineering mindset

Now consider scaling it into a **real-world system** like **Zomato or Swiggy**:

- **How will this task evolve?**
  Tomorrow, restaurants may want to add discounts, multiple payment options, or delivery tracking.
- **How will this code adapt?**
  Instead of hardcoding menu items, use a **database** with menu, restaurant, and order details.
- **What does this code encourage others to do?**
  Provide APIs so delivery apps, restaurants, and payment gateways can integrate easily.

Software engineering mindset - Example(Cont…)

- **How will I understand this code in 5 months?** Break logic into **modules** (user interface, payment, restaurant management, delivery tracking) and document workflows.
- **How will a busy team member grok this?** Write clear code with standard design patterns, unit tests, and sequence diagrams for new developers.
- **What happens when the business becomes bigger?** When thousands of users place orders simultaneously, code must handle concurrency, load balancing, and failover.

Software engineering mindset - Example(Cont…)

- **When will this code stop being good enough?**
  A single-server solution breaks when too many orders come in. Move to microservices and cloud infrastructure.
- **How does it scale / generalize?**
  Extend the code to handle not just food delivery, but also grocery delivery, medicine delivery, or event catering.
- **What hidden dependencies are there?**
  If code assumes “internet connection is always available” or “every restaurant has a fixed menu,” it will fail in real-world conditions.

Software Engineering vs. Programming - summary

👉 **In short:**

- **Programming**: “I built a program to order food from a menu.”
- **Software engineering**: “I designed a scalable food ordering platform that adapts to growth, integrates with payments & deliveries, and remains maintainable for years.”

Program vs Software

Program

- Usually small in size
- Author himself is sole user
- Single developer
- Lacks proper user interface
- Lacks proper documentation
- Ad hoc development

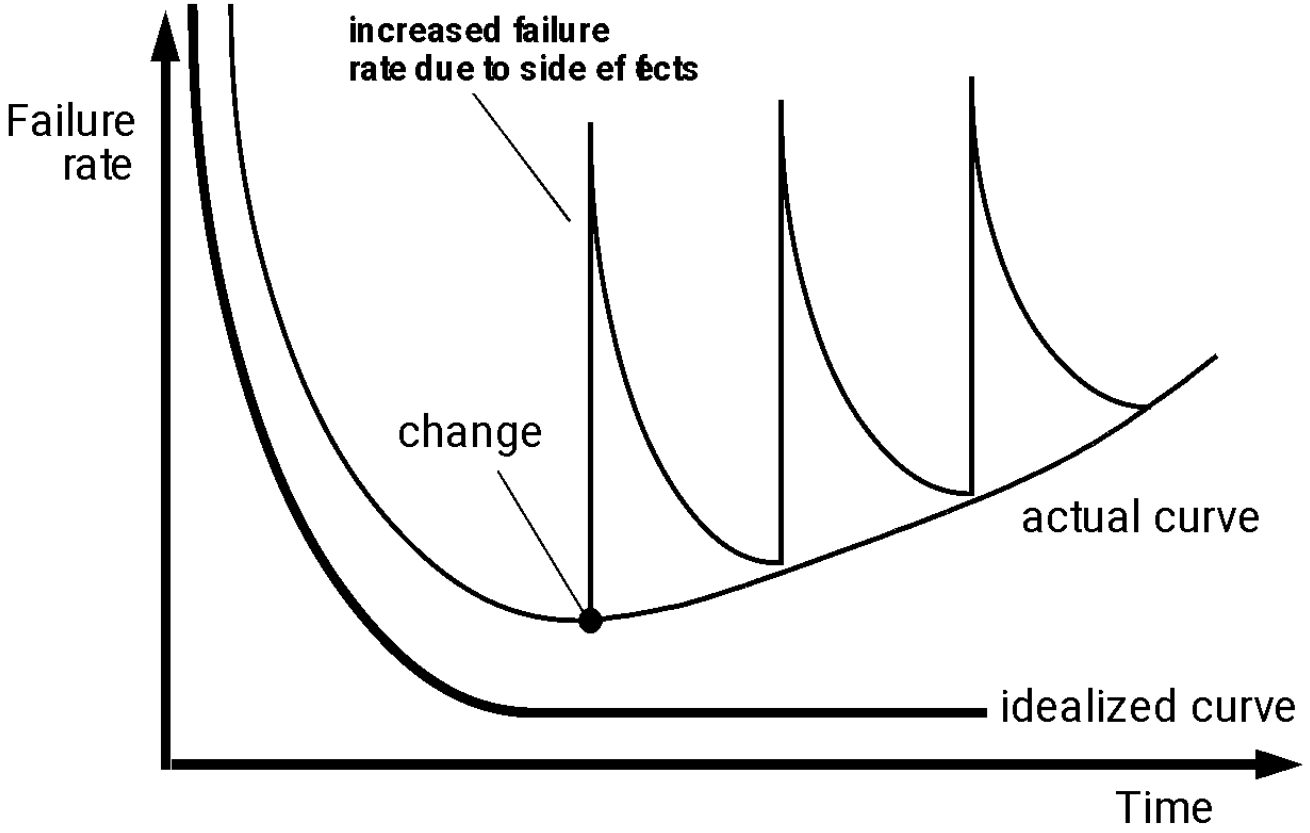
Software

- Large
- Large number of users
- Team of developers
- UI is an important aspect
- Well documented & user-manual prepared
- Systematic development

Software Characteristics

- Software is intangible
  - Hard to understand development effort
- Software is easy to reproduce
  - Cost is in its *development*
  - In other engineering products, manufacturing is the costly stage
- Software is easy to modify
  - People make changes without fully understanding it
- Software does not ‘wear out’
  - It deteriorates by having its design changed:
    - erroneously, or
    - in ways that were not anticipated, thus making it complex

Software Failure Rate



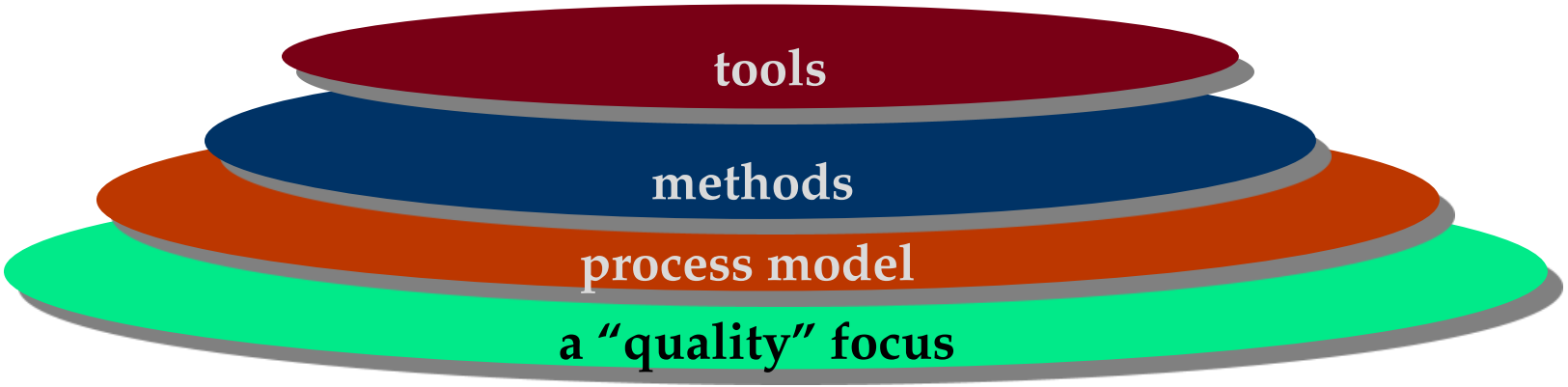
Failure Rate vs. Time in Software

- **Idealized Curve**
    i. Initially high failure rate due to defects.
    ii. Defects corrected → failure rate reduces and stabilizes.
    iii. But this perfect scenario **never occurs in reality**.
- **Actual Software Curve**
    i. Initially high failure rate from undetected defects.
    ii. Defects corrected → temporary stability.
    iii. Software changes/updates → **new defects introduced**.
    iv. Failure rate rises again, then comes down after fixes.
    v. This cycle **repeats continuously** → no wear-out phase like hardware.

Software Engineering - Defining

- Engineering approach to develop software.
  - It discusses systematic and cost effective techniques for software development.
- The process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints
- The application of a systematic, disciplined, quantifiable approach to the development, operation, maintenance of software; that is, the application of engineering to software. (IEEE)

Software Engineering: A Layered Technology



Software Engineering

Describing the layers

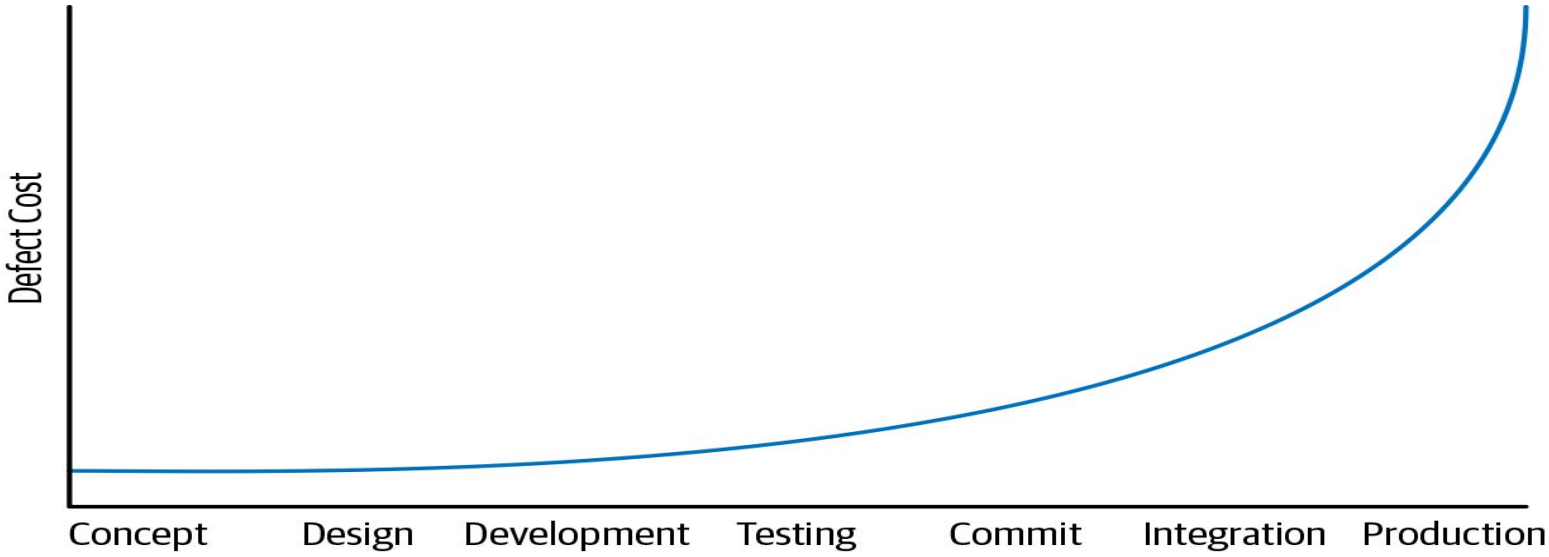
1. Quality focus :
    - Any engineering approach (including software engineering) must rest on an organizational commitment to quality.
    - Total quality management, Six Sigma, and similar philosophies foster a continuous process improvement culture, and it is this culture that ultimately leads to the development of increasingly more effective approaches to software engineering.
    - The bedrock that supports software engineering is a quality focus.

2. Process:
    - The foundation for software engineering is the process layer.
    - The software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software. Process defines a framework that must be established for effective delivery of software engineering technology.
    - The software process forms the basis for management control of software projects and establishes the context in which technical methods are applied, work products (models, documents, data, reports, forms, etc.) are produced, milestones are established, quality is ensured, and change is properly managed.

Describing the layers

3. Methods:
    - Software engineering methods provide the technical how-to's for building software.
    - Methods encompass a broad array of tasks that include communication, requirements analysis, design modeling, program construction, testing, and support.
    - Software engineering methods rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.
4. Tools:
    - Software engineering tools provide automated or semi automated support for the process and the methods.
    - When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called computer-aided software engineering, is established

Time and Change



Lifespan and the importance of upgrades

Time and Change

- You're performing a task that hasn't yet been done for this project;
  more hidden assumptions have been baked-in

- The engineers trying to do the upgrade are less likely to have experience in this sort of task.

- The size of the upgrade is often larger than usual, doing several years' worth of upgrades at once instead of a more incremental upgrade.

Hyrum's Law (Law of Leaky Abstractions)

- If you are maintaining a project that is used by other engineers, the most important lesson about "it works" versus "it is maintainable" is what we've come to call **Hyrum's Law**:
  "**With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviours of your system will be depended on by somebody.**"
- **Meaning**: Even if you document only certain behaviors of your API, users will start relying on unintended, undocumented, or accidental behavior.
  - Changing these "hidden" behaviors later can break users' code.
- Google engineers shared this principle based on their own experience at scale.

Hyrum's Law (Cont…)

For example, Google's **"Cloud Storage API"** had a method that returned a list of files in a bucket.

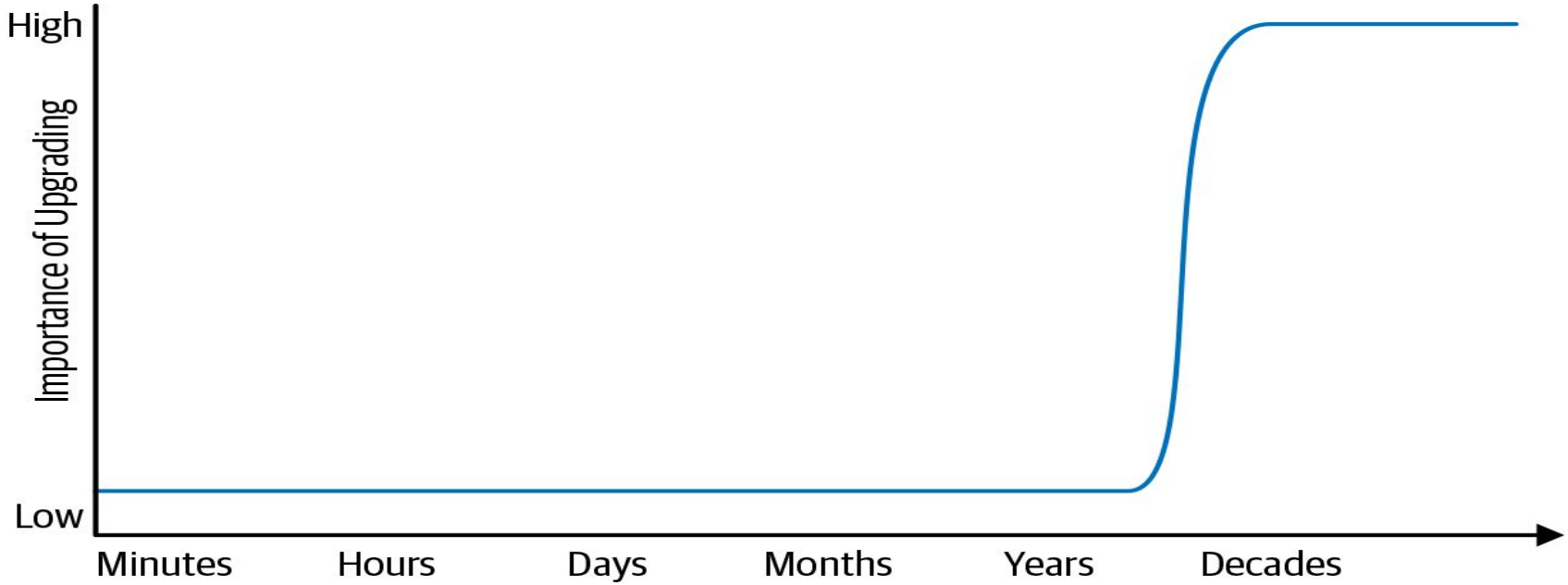
- **The Situation -** The API contract only promised: "Returns a list of files." But in practice, the implementation happened to return the files sorted by filename (even though sorting was not documented).
- **What Happened -**
  - Many developers wrote applications that depended on the **accidental ordering.**
  - When Google later optimized the backend and stopped sorting (to improve performance), thousands of applications broke.
- **Lesson** - Even though the API did not guarantee order, users had treated the observable side-effect (sorting) as a feature.
- **Google learned : Every observable detail becomes a dependency. Once released, it's nearly impossible to remove such behavior without breaking customers.**

Hyrum's Law (Cont…)

- **Mitigation Strategies:** Organizations use several strategies to reduce the risks of Hyrum's Law:
  - Strict API Contracts – Be explicit about guarantees.
  - Black-box Testing – Check what clients might depend on unintentionally.
  - Deprecation Policies – Provide long transition periods before removing accidental behaviors.
  - Feature Flags / Versioning – Allow old behavior to coexist for legacy users.
- **Hyrum's Law reminds us that**

"Users will exploit every behavior they observe" — whether or not it was intended. Once shipped, your accidental design decisions become permanent API commitments.

Timeline of the developer workflow



Timeline of the developer workflow

Efforts and Costs

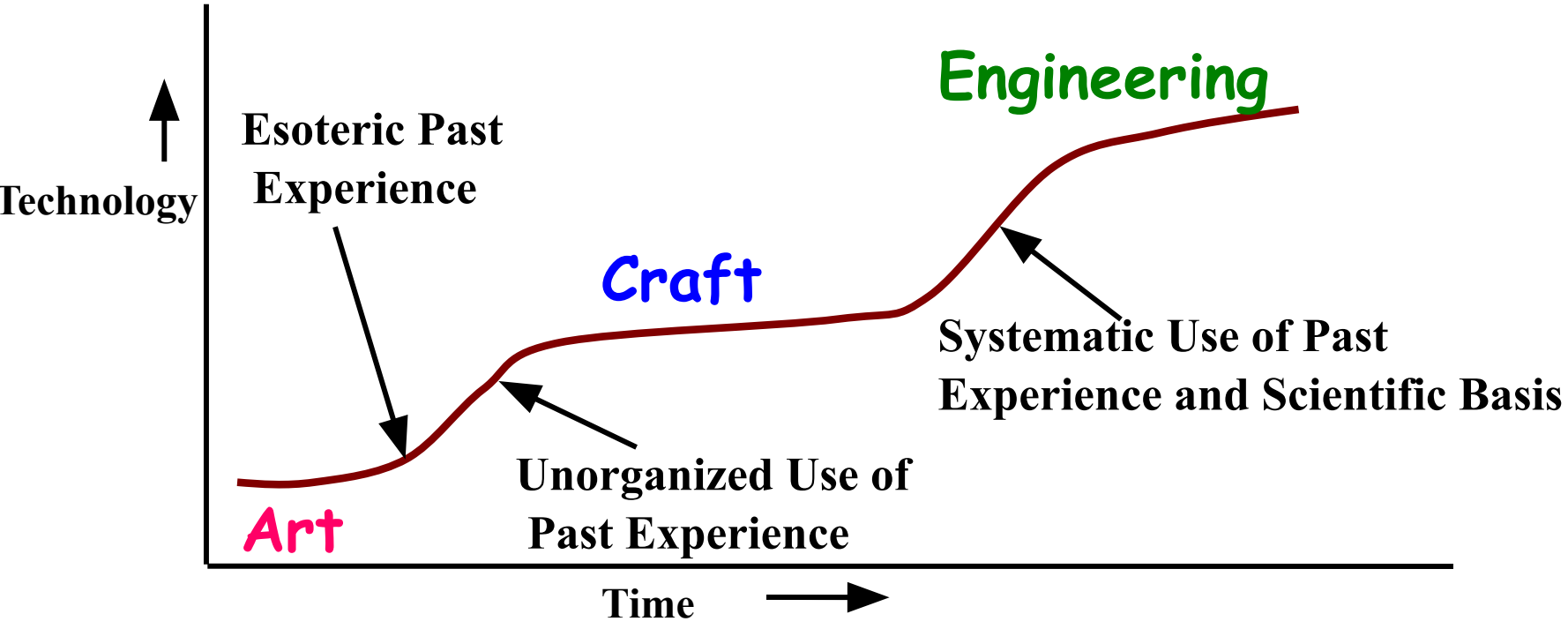
"Cost" roughly translates to effort, and can involve any or all of these factors:

- Financial costs (e.g., money)
- Resource costs (e.g., CPU time)
- Personnel costs (e.g., engineering effort)
- Transaction costs (e.g., what does it cost to take action?)
- Opportunity costs (e.g., what does it cost to "not" take action?)
- Societal costs (e.g., what impact will this choice have on society at large?)

Evolution Pattern

- Every technology in the initial years starts from an art.
- Over time, it graduates to a craft and finally emerges as an engineering discipline.
  - Example: Iron making, paper making, software development, or building construction.
- i.e. say iron making:
  - The esoteric knowledge got transferred from generation to generation as a family secret
  - Slowly a tradesman shared their knowledge with their apprentices and the knowledge pool continues to grow.
  - Much later, through a systematic organization and documentation of knowledge, and incorporation of scientific basic, modern steel making technology emerged.

Evolution Pattern of Technology



Software Engineering Evolution: From An Art To Engineering Discipline

Software Engineering as an Art

- Early phase → limited to a few skilled individuals
- Relied on **creativity, intuition, and ad-hoc coding**
- No rules, standards, or formal methodologies
- Code quality = personal skill & style → often inefficient
- Still important today → creativity, aesthetics, user-friendliness

Example:

- Like **Gold ornament making** in ancient times – only a few knew the craft, kept as a secret, and accuracy was low.
- Similarly, only a few people could design/write software efficiently; others found it mysterious.

Transition from Art to Craft

- Knowledge spread → more people trained in coding & design
- Introduction of **rules, degrees, and coding standards**
- Focus on structured approaches & methodologies (e.g., Waterfall, Agile)
- Program writing became **more efficient & reusable**
- Accuracy improved but still not fully systematic

Example:

- Gold ornament making expanded beyond families → more artisans trained → accuracy and shape improved.
- Likewise, software engineers developed **standards and best practices** for more reliable coding.

Transition from Craft to Engineering Discipline

- Matured into a **formal engineering field**
- Recognized as an **engineering discipline** (e.g., Pressman's 1975 book)
- Use of scientific methods:
  - Formal methods
  - Model-driven development
  - Software architecture
- **Everyone can now learn coding** (with or without degrees)
- Improved **accuracy, scalability, maintainability, cost-effectiveness**

Example:

- Today gold ornament making uses **machines + professional training**, open for anyone to learn.
- Similarly, software engineering is open for all, backed by **science, standards, and automation tools**.
- Benefits: Improved quality, increased productivity, better communication, greater maintainability, better cost management, better scalability.

Types of Software - I

- Customized
  - For a specific customer according to requirement.
- Generic
  - Used by a diverse range of customers
  - Sold on open market (Ms. Windows, Oracle, …)
  - Often called
    - COTS (Commercial Off The Shelf)
    - Shrink-wrapped
- Embedded
  - Built into hardware
  - Hard to change

Types of Software - II

- Real time software
  - E.g. control and monitoring systems
  - Must react in time
  - Safety often a concern
- Data processing software
  - Used to run businesses
  - Accuracy and security of data are key
- Some software has both aspects

Generic versus Custom S/W

Generic

- The specification is owned by the product developer.
- The developer can quickly decide to change the specification in response to some external change.
- Users of generic products have no control over the software specification so cannot control the evolution of the product.
- The developer may include/exclude features and change the user interface which may affect user's business processes and add extra training costs when new versions of the system are installed. Again it may limit the customer's flexibility to change their own business processes.

Custom

- The specification is owned and controlled by the customer as the domain, requirement and environment being unique to the customer.
- changes have to be negotiated between the customer and the developer and may have contractual implications.

Quiz - I

- Classify the following software according to whether it is likely to be custom, generic or embedded (or some combination); and whether it is data processing or real-time.
  - A system to control the reaction rate in a nuclear reactor.
  - A program that runs inside badges worn by nuclear plant workers that monitors radiation exposure.
  - A program used by administrative assistants at the nuclear plant to write letters.
  - A system that logs all activities of the reactor and its employees so that investigators can later uncover the cause of any accident.
  - A program used to generate annual summaries of the radiation exposure experienced by workers.
  - An educational web site containing a Flash animation describing how the nuclear plant works.

Quiz - I

- Classify the following software according to whether it is likely to be custom, generic or embedded (or some combination); and whether it is data processing or real-time.
  - A system to control the reaction rate in a nuclear reactor.
  - A program that runs inside badges worn by nuclear plant workers that monitors radiation exposure.
  - A program used by administrative assistants at the nuclear plant to write letters.
  - A system that logs all activities of the reactor and its employees so that investigators can later uncover the cause of any accident.
  - A program used to generate annual summaries of the radiation exposure experienced by workers.
  - An educational web site containing a Flash animation describing how the nuclear plant works.

Cust & RT, Embd & RT, Gen and DP, Gen and DP, Cust and DP, Cust/Gen and RT

Stakeholders in Software Engineering

- Users
  - Those who use the software
- Customers
  - Those who pay for the software
- Software developers
- Development Managers
- All four roles can be fulfilled by the same person

Case Study: ATM stake holders

- Bank customers
- Representatives of other banks
- Bank managers
- Counter staff
- Database administrators
- Security managers
- Marketing department
- Hardware and software maintenance engineers
- Banking regulators

Software Quality

- Usability
  - Users can learn it fast and get their job done easily
- Efficiency
  - It doesn't waste resources such as CPU time and memory
- Reliability
  - It does what it is required to do without failing
- Maintainability
  - It can be easily changed
- Reusability
  - Its parts can be used in other projects, so reprogramming is not needed

Quiz - II

- Find most and the least important attributes for each of the following systems.
  - A web-based banking system, enabling the user to do all aspects of banking on-line.
  - An air traffic control system.
  - A program that will enable users to view digital images or movies stored in all known formats.
  - A system to manage the work schedule of nurses that respects all the constraints and regulations in force at a particular hospital.
  - An application that allows you to purchase any item seen while watching TV.

Quiz - II

- Find most and the least important attributes for each of the following systems.
  - A web-based banking system, enabling the user to do all aspects of banking on-line.
  - An air traffic control system.
  - A program that will enable users to view digital images or movies stored in all known formats.
  - A system to manage the work schedule of nurses that respects all the constraints and regulations in force at a particular hospital.
  - An application that allows you to purchase any item seen while watching TV.

a - rel, reusa, b- rel, reusa, c- usa, eff, d- maint, reusa ,e- rel/usa/maint, eff

Software Quality and the Stakeholders

Customer:
solves problems at an acceptable cost in terms of money paid and resources used

User:
easy to learn;
efficient to use;
helps get work done

Developer:
easy to maintain;
easy to design;
easy to reuse its parts

Development manager:
pleases customers
sells more and costing less to develop and maintain

Software Quality: Conflicts and Objectives

- The different qualities can conflict
  - Increasing efficiency can reduce maintainability or reusability
  - Increasing usability can reduce efficiency
- Setting objectives for quality is a key engineering activity
  - You then design to meet the objectives
  - Avoids 'over-engineering' which wastes money
- Optimizing is also sometimes necessary
  - e.g. obtain the highest possible reliability using a fixed budget

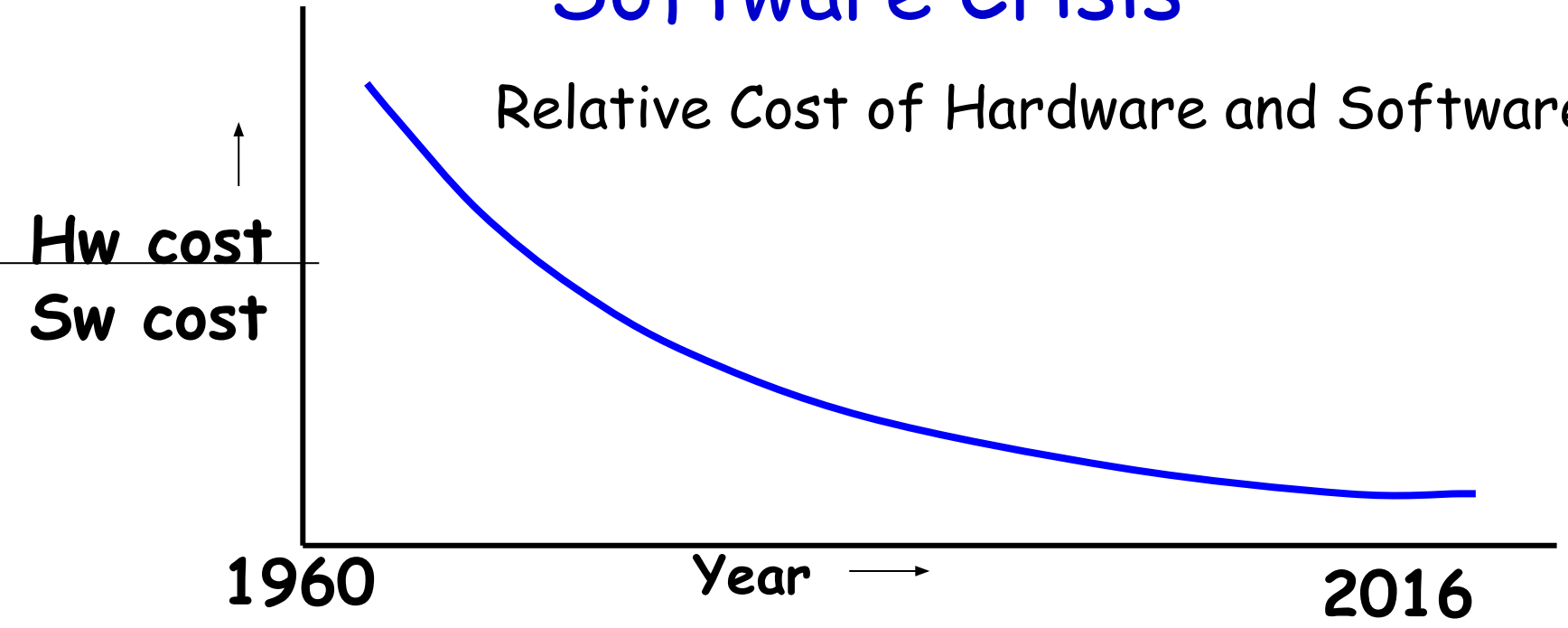
Quiz- III

- How do you think each of the four types of stakeholders would react in each of the following situations?
- Problem: A new system that will completely automate the work of one individual in the customer's company.
  - Case a:
  - You discover that the cost of developing the system would be far more than the cost of continuing to do the work manually, so you recommend against proceeding with the project.
  - Case b: You implement a system according to the precise specifications of a customer.
  - However, when the software is put into use, the users find it does not solve their problem.

Software Crisis

- Software products:
  - fail to meet user requirements.
  - frequently crash.
  - expensive.
  - difficult to alter, debug, and enhance.
  - often delivered late.
  - use resources non-optimally.
  - Relative Cost of Hardware and Software

Software Crisis



- Imagine: When you buy a software, the hardware on which the software runs would come free with the software!!!

Factors contributing to the software crisis

- Larger problems
- Lack of adequate training in software engineering
- Increasing skill shortage
- Low productivity improvements

Thank You