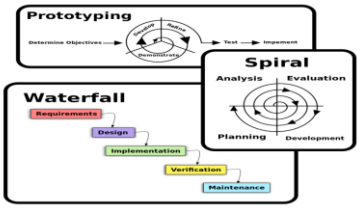
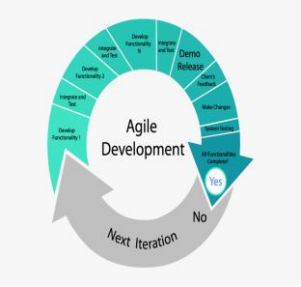
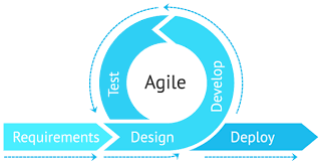


Software Engineering Introduction





Outline

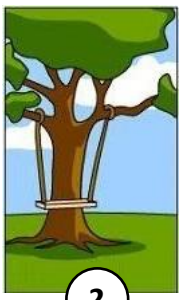
- Software and Role of Software
- Types (nature) of Software
- Software Engineering-A Layered Technology
- Software Process
- Software Myths
- Software Engineering Practices.

Why to Study Software Engineering?

- Software Development Life Cycle without Software Engineering

1

How the Customer Explains Requirement

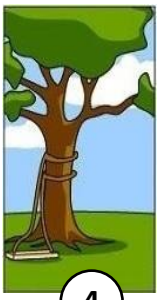


2

How the Project Leader understand it

3

How the System Analyst design it



4

How the Programmer Works on it

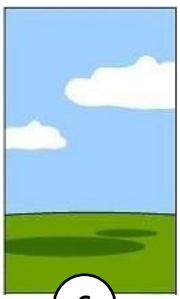
3

Why to Study Software Engineering?

- Software Development Life Cycle without Software Engineering

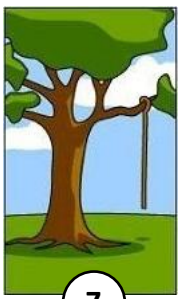
5

How the Business Consultant describe it



6

How the Project documented



7

What Operations Installed

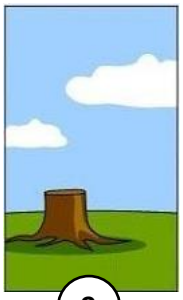
4

Why to Study Software Engineering?

- Software Development Life Cycle without Software Engineering

8

How the Customer billed



9

How it was supported

10

What the customer really needed

5

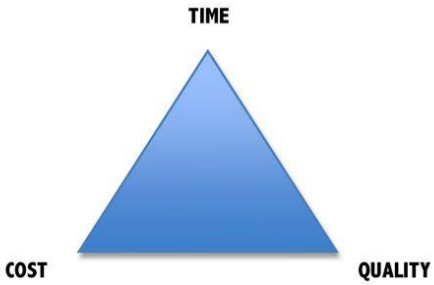
Software Engineer

- Software engineers build and support software, and virtually everyone in the industrialized world uses it either directly or indirectly.
- Software engineering encompasses a process, a collection of methods (practice) and an array of tools that allow professionals to build high-quality computer software.

6

Why is Software Engineering important?

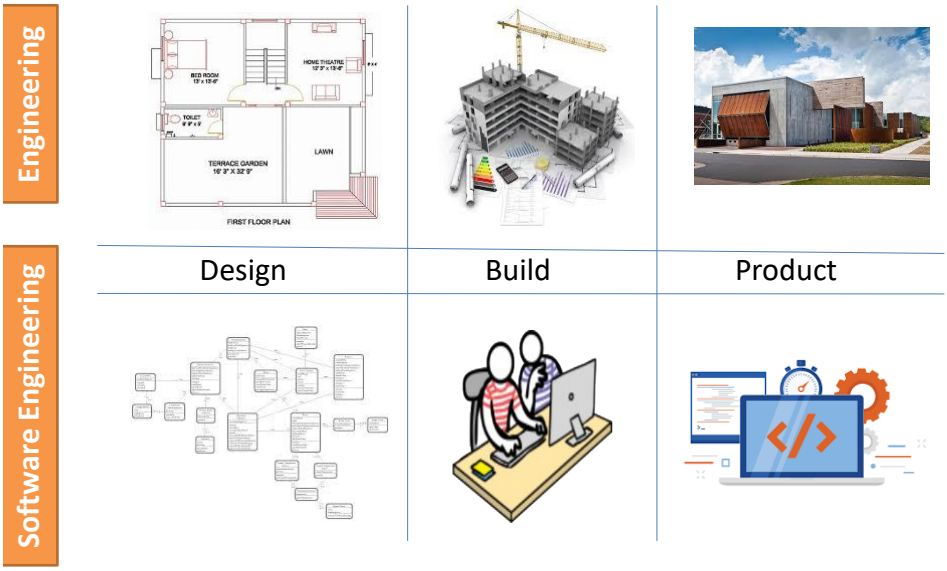
- Complex systems need a disciplined approach for designing, developing and managing them.



- "you can have it fast, you can have it right, or you can have it cheap. "
- Pick two!"

7

Software Engineer



8

Software Development Crises

- Projects were:
  - Late.
  - Over budget.
  - Unreliable.
  - Difficult to maintain.
  - Performed poorly.
- Software errors….the cost
  - Errors in computer software can have devastating effects.

9

Software Crises

- Example 1: 2009,Computer glitch delays flights
- Saturday 3rd October 2009-London, England (CNN)
- Dozens of flights from the UK were delayed Saturday after a glitch in an air traffic control system in Scotland, but the problem was fixed a few hours later.
- The agency said it reverted to backup equipment as engineering worked on the system.
- The problem did not create a safety issue but could cause delays in flights.

10

Software Crises

- Example 2: Ariane 5 Explosion
- European Space Agency spent 10 years and $7 billion to produce Ariane 5.
- Crash after 36.7 seconds.
- Caused by an overflow error. Trying to store a 64-bit number into a 16-bit space.
- Watch the video: http://www.youtube.com/watch?v=z-r9cYp3tTE

11

Software Crises

- Example 3: 1992, London Ambulance Service
- Considered the largest ambulance service in the world.
- Overloaded problem.
- It was unable to keep track of the ambulances and their statuses. Sending multiple units to some locations and no units to other locations.
- Generates many exceptions messages.
- 46 deaths.

12

Software Engineering

- Therefore…
  - A well-disciplined approach to software development and management is necessary. This is called engineering.
- The term software engineering first appeared in the 1968 NATO Software Engineering Conference and was meant to provoke thought regarding what was then called the “software crisis”
- “An engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use.”

13

Software

- Today, software takes on a dual role. It is a product, and at the same time, the vehicle for delivering a product.
  - As a product, it delivers the computing potential embodied by computer hardware or more broadly, by a network of computers that are accessible by local hardware (can vary from bit to tons of bits)
  - As the vehicle used to deliver the product, software acts as the basis for the control of the computer (operating systems), the communication of information (networks), and the creation and control of other programs (software tools and environments).
- Software delivers the most important product of our time—information.

14

Software

- Vast increases in memory and storage capacity, and a wide variety of exotic input and output options, have all precipitated more sophisticated and complex computer-based systems.
- Sophistication and complexity can produce dazzling results when a system succeeds, but they can also pose huge problems for those who must build complex systems.

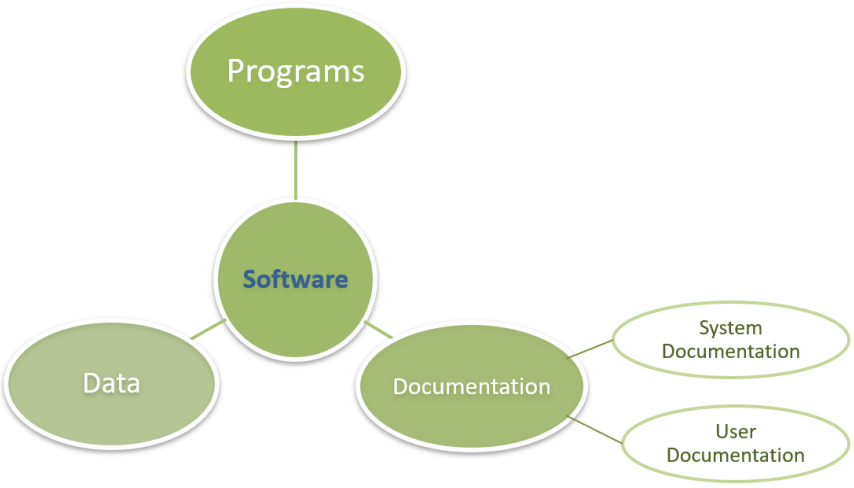
15

Questions Asked

- 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?

16

What is Software?



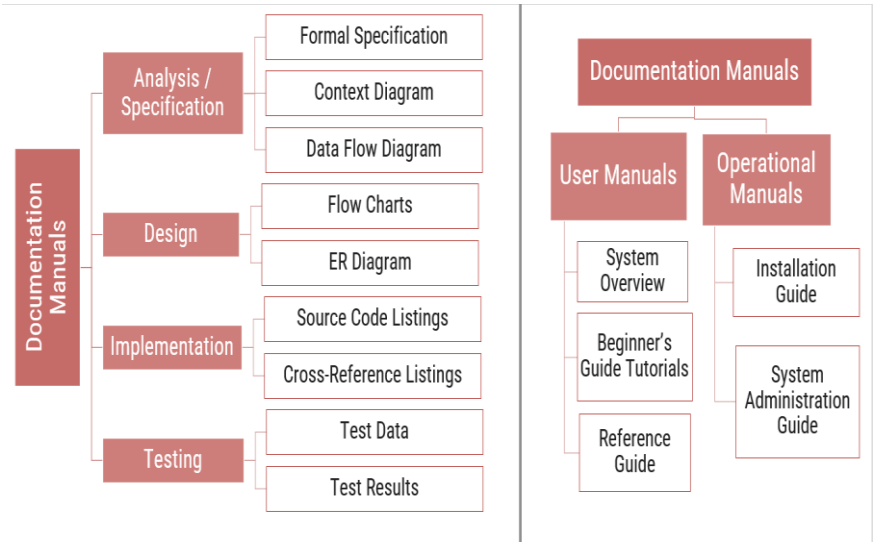
17

Defining Software

- Software is:
  - instructions (computer programs) that when executed provide desired features, function, and performance;
  - data structures that enable the programs to adequately manipulate information and
  - documentation that describes the operation and use of the programs.
- Software is a logical rather than a physical system element.

18

List of documentation & manuals



19

Types of Software

- Generic products.
  - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
  - Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
  - The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.

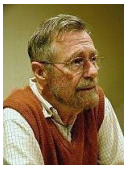
20

Types of Software

- Customized products.
  - Software that is commissioned by a specific customer to meet their own needs.
  - Examples – embedded control systems, air traffic control software, traffic monitoring systems.
  - The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.

21

Software Engineering vs. Computer Science



- "Computer science is no more about computers than astronomy is about telescopes." Edsger Dijkstra

Computer Science	Software Engineering
• Theory. • Fundamentals.	• Practicalities of software design, development and delivery.

22

Software Engineering vs. Systems Engineering

- Systems Engineering:
  - Interdisciplinary engineering field (computer, software, and process eng.).
  - Focuses on how complex engineering projects should be designed and managed.

Systems Engineering	Software Engineering
• All aspects of computer-based systems development: HW + SW + Process. • Older than SWE.	• Deals with the design, development and delivery of SW. • Is part of Systems Engineering.

23

Software Engineering Characteristics

1. **Software is developed or engineered, it is not manufactured in the classical sense.**

- Although some similarities exist between software development and hardware manufacturing, the two activities are fundamentally different.
- In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.
- Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different
- Both activities require the construction of a "product," but the approaches are different. Software costs are concentrated in engineering.
- This means that software projects cannot be managed as if they were manufacturing projects.

24

Software Engineering Characteristics

2. Software doesn't "wear out. But it does deteriorate!

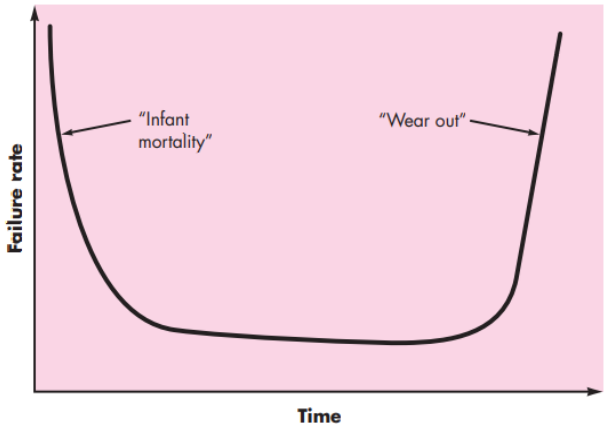


Figure depicts failure rate as a function of time for hardware

25

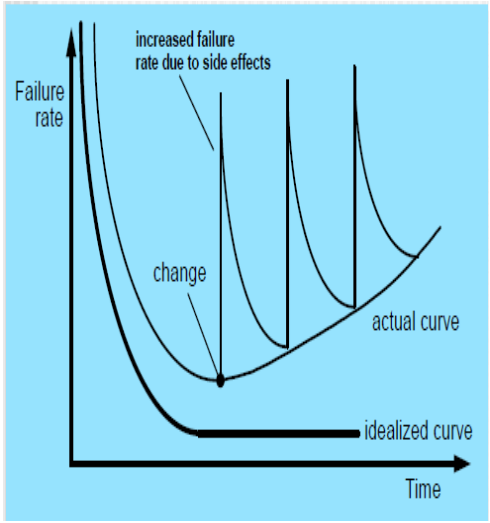
Software Engineering Characteristics

- The relationship, often called the "bathtub curve,"
- Indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects);
- defects are corrected and the failure rate drops to a steady-state level (hopefully, quite low) for some period of time.
- As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.

26

Software Engineering Characteristics

2. **Software doesn't "wear out. But it does deteriorate!**



- During its life, software will undergo change. As changes are made, it is likely that errors will be introduced, causing the failure rate curve to spike as shown in the "actual curve"
- There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code.

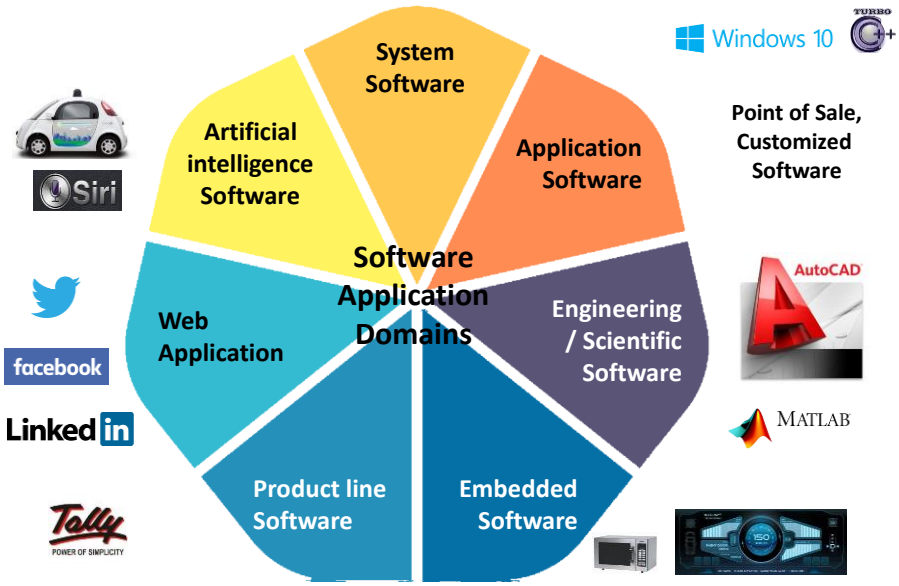
27

Software Engineering Characteristics

3. **Although the industry is moving toward component-based construction, most software continues to be custom built.**
   - As an engineering discipline evolves, a collection of standard design components is created.
   - The reusable components have been created so that the engineer can concentrate on the truly innovative elements of a design, that is, the parts of the design that represent something new.
   - A software component should be designed and implemented so that it can be reused in many different programs.
   - The data structures and processing detail required to build the interface are contained within a library of reusable components for interface construction.

28

Software Applications Domains



Software Engineering

- Some realities:
  - a concerted effort should be made to understand the problem before a software solution is developed
  - design becomes a pivotal activity
  - software should exhibit high quality
  - software should be maintainable
- These simple realities lead to one conclusion: software in all of its forms and across all of its application domains should be engineered.

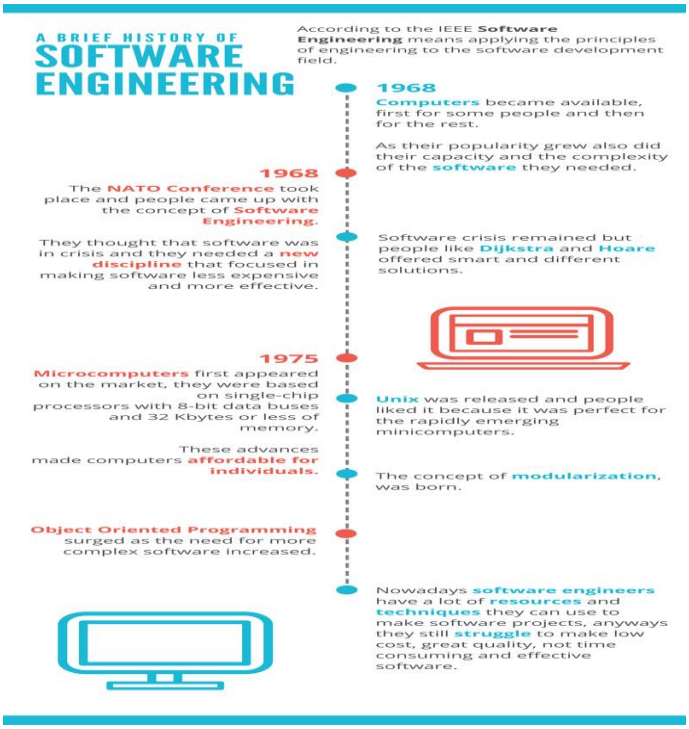
Definition

- Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.- Fritz Bauer
- it omits mention of the importance of measurement and metrics; it does not state the importance of an effective process. And yet, Bauer's definition provides us with a baseline.
- The IEEE definition: Software Engineering:

  (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

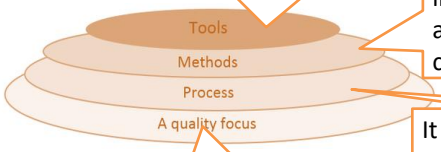
  (2) The study of approaches as in (1).

31



A Layered Technology

Software Engineering Tools **allows automation of activities** which helps to perform systematic activities. A system for the support of software development, called **computer-aided software engineering** (CASE). **Examples:** Testing Tools, Bug/Issue Tracking Tools etc…



It provides **technical how-to's** for building software, it encompasses **many tasks** including communication, requirement analysis, design modeling, program construction, testing and support

It is a foundation of Software Engineering, It is the **glue** the holds the **technology layers**, It **defines** a **framework activities**

Defines continuous **process improvement principles**

33

Software Process

- A **process** is a collection of **activities**, **actions** and **tasks** that are performed when some work product is to be created
- A process is not a **rigid prescription** for how to build the software, rather it is **adaptable approach** that enables the people doing the work to **pick** and **choose** the **appropriate set of work actions** and tasks
- An **activity** try to **achieve** a **broad objective** (e.g., communication with stakeholders)
- An **activity** is **applied** regardless of the **application domain**, **size of the project**, **complexity of the effort**, or **degree of accuracy** with which software engineering is to be applied.
- An **action** (e.g., architectural design) **encompasses** a **set of tasks** that **produce** a major **work product** (e.g., an architectural design model).

34

Software Process

- A **task** **focuses** on a **small, but well-defined objective** (e.g., conducting a unit test) that **produces** a **noticeable outcome**.
- **Each of these** activities, actions & tasks **reside within** a **framework** or model

35

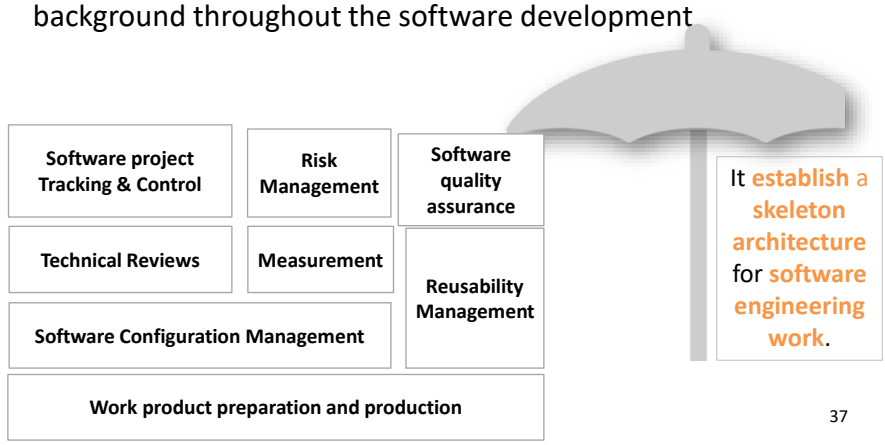
The Process Framework

- Software engineering process framework activities are complemented by a number of umbrella activities.
- In general, umbrella activities are applied throughout a software project and help a software team manage and control progress, quality, change, and risk. Typical umbrella activities include:
  - Software project management
  - Formal technical reviews
  - Software quality assurance
  - Software configuration management
  - Work product preparation and production
  - Reusability management
  - Measurement
  - Risk management

36

Umbrella activities

- **Umbrella activities** applied throughout the software project & **help a software team to manage and control progress, quality, change & risks**
- Umbrella activities are those which keep running in the background throughout the software development



37

Software Myths

- Beliefs about software and the process used to build it.

"**Misleading Attitudes that cause serious problem**" are myths.

Management Myths

Customer Myths

Practitioner's (Developer) Myths

38

Management Myths

We **have standards and procedures** to build a system, which is enough.

Reality

- Are software **practitioners aware** of standard's existence?
- Does it **reflect modern software engineering** practice?
- Is it **complete**?
- Is it streamlined to **improve time to delivery** while **still maintaining a focus on quality**?
- In many cases, the answer to all of these questions is "no."

39

Management Myths

We have **the newest computers and development tools**.

Reality

- It **takes much more than the latest model** computers to do high-quality software development.
- **Computer-aided software engineering (CASE)** tools are more important than hardware.

40

Management Myths

We **can add more programmers** and can catch up the schedule.

Reality

- Software **development is not a mechanistic process** like manufacturing.
- In the words of Fred Brooks : "**adding people to a late software project makes it later.**"
- **People** who were **working** must **spend time educating** the newcomers.
- People can be added but only **in a planned and well-coordinated** manner.

41

Management Myths

I **outsourced the development** activity, now I **can relax** and **can wait** for the **final** working **product**.

Reality

- If an **organization** does **not understand how to manage** and **control** software projects internally, it will invariably struggle when it outsources software projects.

42

Customer Myths

A **general statement of objectives** (requirements) is **sufficient** to start a development.

Reality

- Comprehensive (**detailed**) **statements** of requirements is not always possible, an **ambiguous** (unclear) "**statement of objectives**" can lead to disaster.
- Unambiguous (clear) requirements can be gathered only through effective and continuous communication between customer and developer.

43

Customer Myths

Requirement Changes can be **easily accommodated** because software is very flexible.

Reality

- It is true that software **requirements change**, but the **impact** of change **varies with the time** at which it is introduced.
- When requirements changes are requested early the cost impact is relatively small.

44

Practitioner's (Developer) Myths

Once we **write** the **program**, our **job is done**.

Reality

- Experts say "**the sooner you begin 'writing code', the longer it will take you to get done.**"
- Industry data indicates that 60 to 80 % effort expended on software will be after it is delivered to the customer for the first time.

45

Practitioner's (Developer) Myths

I **can't** access **quality until** it is **running**.

Reality

- One of the most effective software **quality assurance mechanisms** can be **applied from** the **beginning** of a project - **the technical review**.
- Software reviews are more effective "quality filter" than testing for finding software defects.

46

Practitioner's (Developer) Myths

Working **program** is the **only deliverable** work **product**.

Reality

- **A working program** is only one **part of** a **software configuration**.
- A variety of work products (e.g., **models, documents, plans**) provide a foundation for successful engineering and, more important, guidance for software support.

47

Practitioner's (Developer) Myths

Software engineering is about **unnecessary** documentation.

Reality

- Software engineering is not about creating documents. It is about **creating a quality product**.
- Better quality leads to reduced rework. And reduced rework results in faster delivery times.

48

The Software Engineering Practice

- George Polya suggests the essence of problem solving, and consequently, the essence of software engineering practice:
  - Understand the problem (communication and analysis).
  - Plan a solution (modeling and software design).
  - Carry out the plan (code generation).
  - Examine the result for accuracy (testing and quality assurance).

49

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?

50

Plan a 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?

51

Carry out the plan

- The design you've created serves as a road map for the system you want to build. There may be unexpected detours, and it's possible that you'll discover an even better route as you go, but the "plan" will allow you to proceed without getting lost.
- 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?

52

Examine the results

- 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?

53

FAQ about SE

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.

FAQ about SE

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.

55

56