Chapter 1

Introduction to Software Engineering

- **Software & Software Engineering**

Software Engineering: A Practitioner's Approach, 7/e
by Roger S. Pressman

What is Software?

The product that software professionals build and then support over the long term.

Software encompasses: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately store and manipulate information and (3) documentation that describes the operation and use of the programs.

Software products

- Generic products
  - Stand-alone systems that are marketed and sold to **any customer** who wishes to buy them.
  - Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- 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.

Why Software is Important?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment,)
- 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.

Software costs

- 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.
- Software engineering is concerned with cost-effective software development.

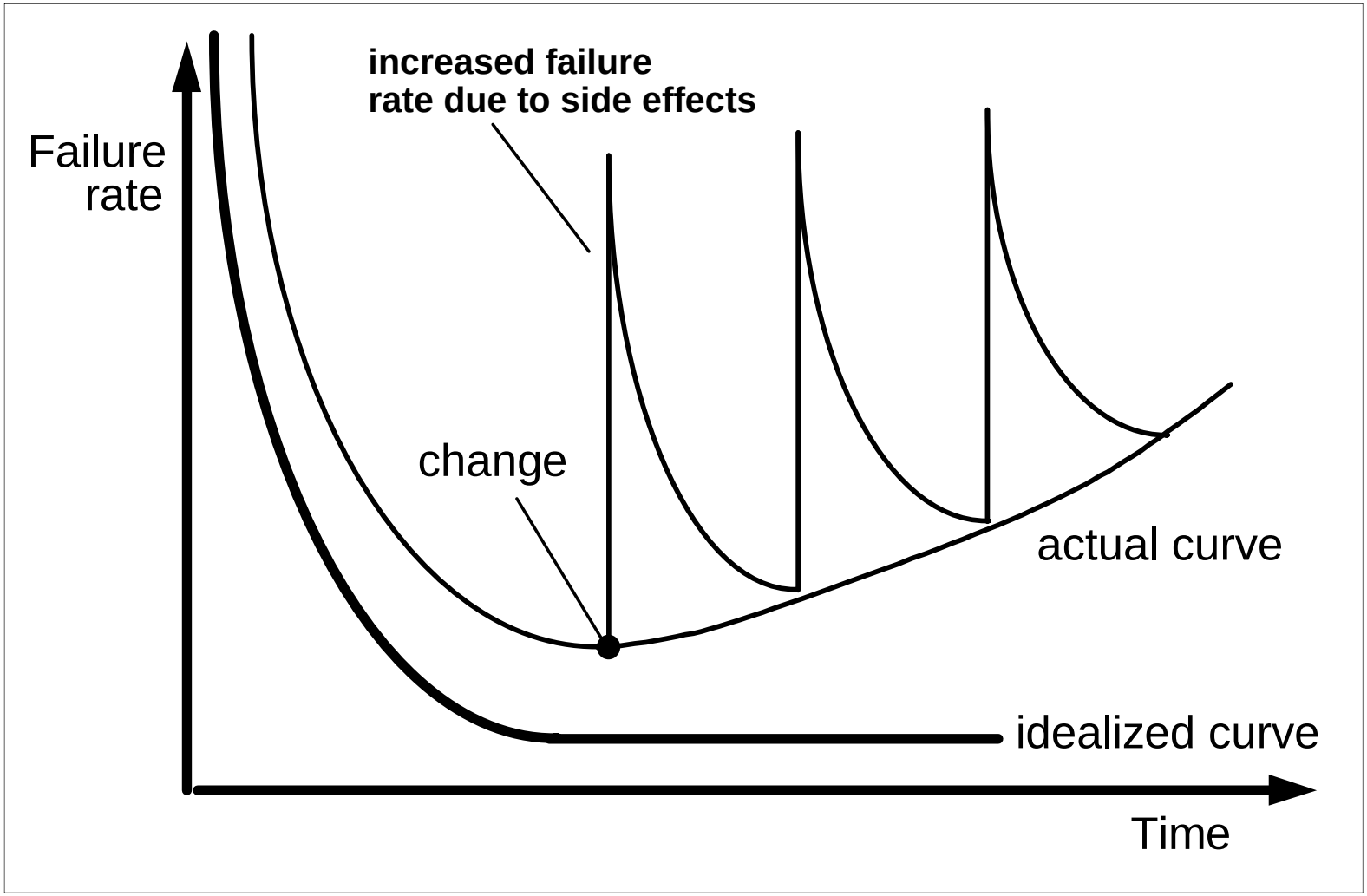
Features of Software?

- Its characteristics that make it different from other things human being build.

Features of such logical system:

- Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out.” but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).
- Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built. Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

Wear vs. Deterioration



Software Applications

- 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 “number crunching”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. Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- 6. WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
- 7. AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

Software—New Categories

- Open world computing—pervasive, ubiquitous, distributed computing due to wireless networking. How to allow mobile devices, personal computer, enterprise system to **communicate across vast network**.
- Netsourcing—the Web as a computing engine. How to architect simple and sophisticated applications to target end-users worldwide.
- Open source—"free" source code open to the computing community (a blessing, but also a potential curse!)
- Also … (see Chapter 31)
  - Data mining
  - Grid computing
  - Cognitive machines
  - Software for nanotechnologies

Software Engineering Definition

The seminal 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.*

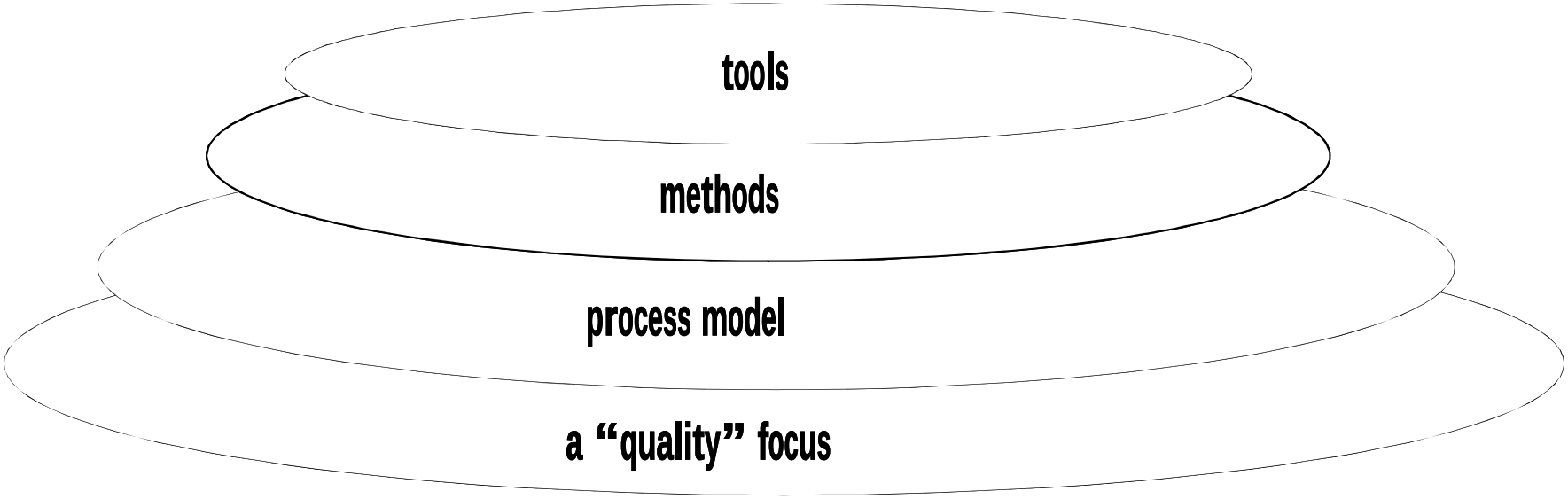
The IEEE definition:

> *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).*

Importance of Software Engineering

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

A Layered Technology



- Any engineering approach must rest on organizational commitment to **quality** which fosters a continuous process improvement culture.
- **Process** layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.
- **Method** provides technical how-to’s for building software. It encompasses many tasks including‼ communication, requirement analysis, design modeling, program construction, testing and support.
- **Tools** provide automated or semi-automated support for the process and methods.

Software Process

- A process is a collection of activities, actions and tasks that are performed when some work product is to be created. It is **not a rigid prescription** for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work to pick and choose the **appropriate set of work actions** and tasks.
- Purpose of process is to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

- Communication: communicate with customer to understand objectives and gather requirements
- Planning: creates a “map” defines the work by describing the tasks, risks and resources, work products and work schedule.
- Modeling: Create a “sketch”, what it looks like architecturally, how the constituent parts fit together and other characteristics.
- Construction: code generation and the testing.
- Deployment: Delivered to the customer who evaluates the products and provides feedback based on the evaluation.

- These five framework activities can be used to all software development regardless of the application domain, size of the project, complexity of the efforts etc, though the details will be different in each case.

- For many software projects, these framework activities are applied **iteratively** as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

Five Activities of a Generic Process framework

Umbrella Activities

Complement the five process framework activities and help team manage and control progress, quality, change, and risk.

- Software project tracking and control: assess progress against the plan and take actions to maintain the schedule.
- Risk management: assesses risks that may affect the outcome and quality.
- Software quality assurance: defines and conduct activities to ensure quality.
- Technical reviews: assesses work products to uncover and remove errors before going to the next activity.
- Measurement: define and collects process, project, and product measures to ensure stakeholder ' s needs are met.
- Software configuration management: manage the effects of change throughout the software process.
- Reusability management: defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- Work product preparation and production: create work products such as models, documents, logs, forms and lists.

Plan the Solution

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

Carry Out the Plan

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

Examine the Result

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