Faculty of Computer Science
University of Indonesia

Topic 01: Introduction to Software Engineering

References

- [Pressman, 2010] Pressman, Roger S. Software Engineering: A Practitioner’s Approach. New York: McGraw-Hill Higher Education, 2010.
- [Sommerville] Sommerville, Ian, Software Engineering, 9th Edition, Pearson-Addison Wesley, England, 2011.

Outline

- What is Software?
- What is Software Engineering?
- The Essence of Software Engineering Practice

What is 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 and

(3) **Documentation** that describes the operation and use of the programs.

Characteristics of Software

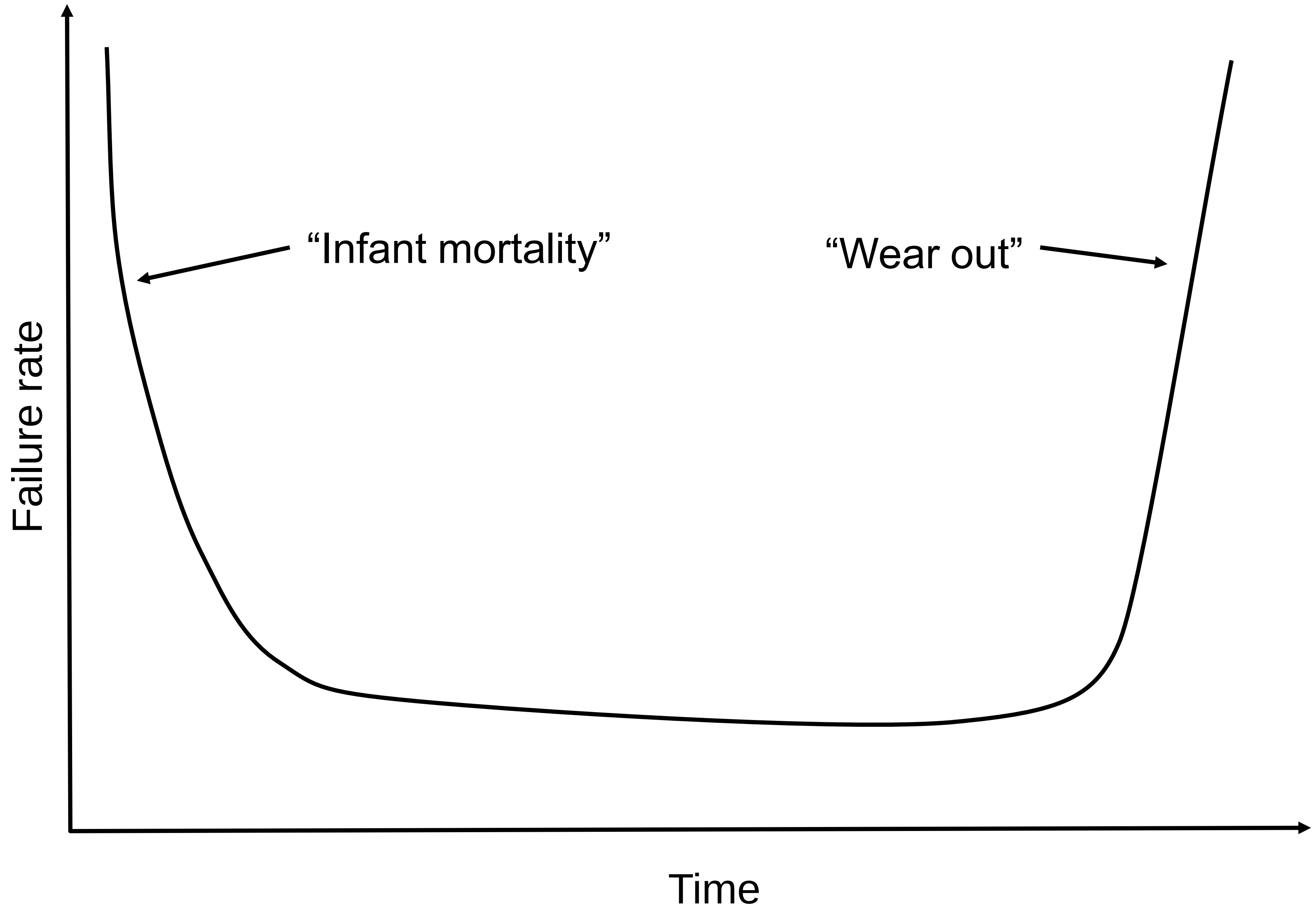
1. Software is **developed or engineered**, it is not manufactured in the classical sense.

2. Software doesn't ***"wear out"*** - it’s ***deteriorate***.

   ✤ Hardware failure rate depicted in a form of the “bathtub curve”

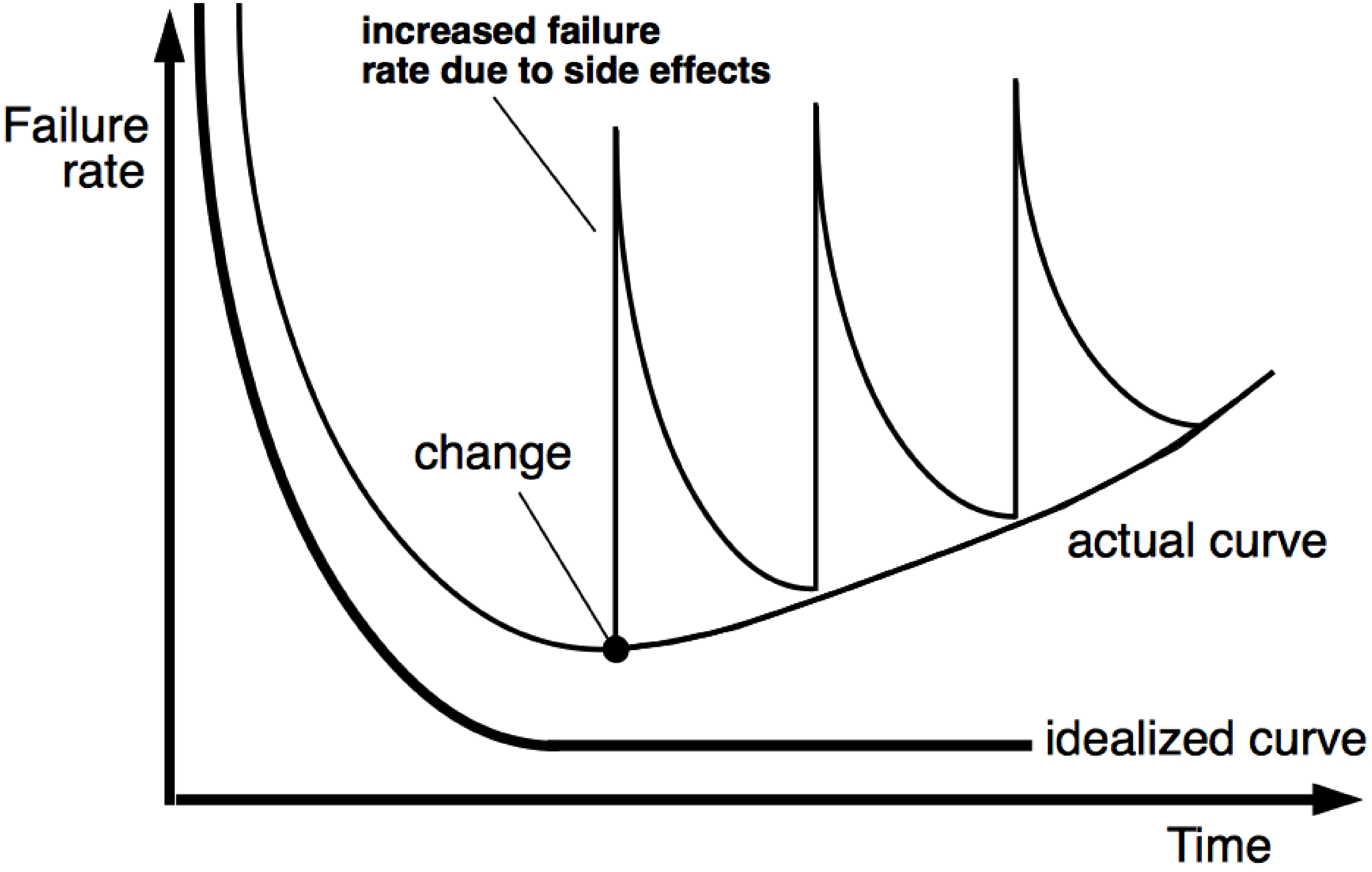
   ✤ Software failure rate depicted in a form of the “idealized curve”.

3. Although the industry is moving toward component-based construction, most software continues to be custom-built.



Failure Curve for Hardware: Bathtub Curve

(Source: Pressman)



Failure Curve for Software

(Source: Pressman)

Software Application

- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- WebApps (Web applications)
- AI software

Information System?

- Information system is a software
  - ✤ Consists of instructions, data structures, and documentations that provide values to its user
- But not all software are information systems!
  - ✤ Anti-virus? BIOS?
  - ✤ Medical records system? Tax filing system?
- An IS for different organization has different features

What is Software Engineering?

- Software engineers often encounter problems related to a computer or existing computer system but sometimes the underlying problem have nothing to do with computers
- Therefore, we must understand the nature of the problem first:
  - ✤ Do not impose computing machinery or techniques on every problems.
  - ✤ Solve the problem first and if needed we can use technology as a tool to implement our solution.

What is Software Engineering?

- Solving problems:
  - Begin investigating a problem by **analyzing** it.
    - Break the problem in pieces that we understand and try to deal with.
    - Understand the relationship between pieces that hold the problem.
  - Once analyzed, construct the solution from the components that address the problem’s various aspects.
    - This reverse process is called **synthesis**:
      - putting together of a large structure from small building blocks.

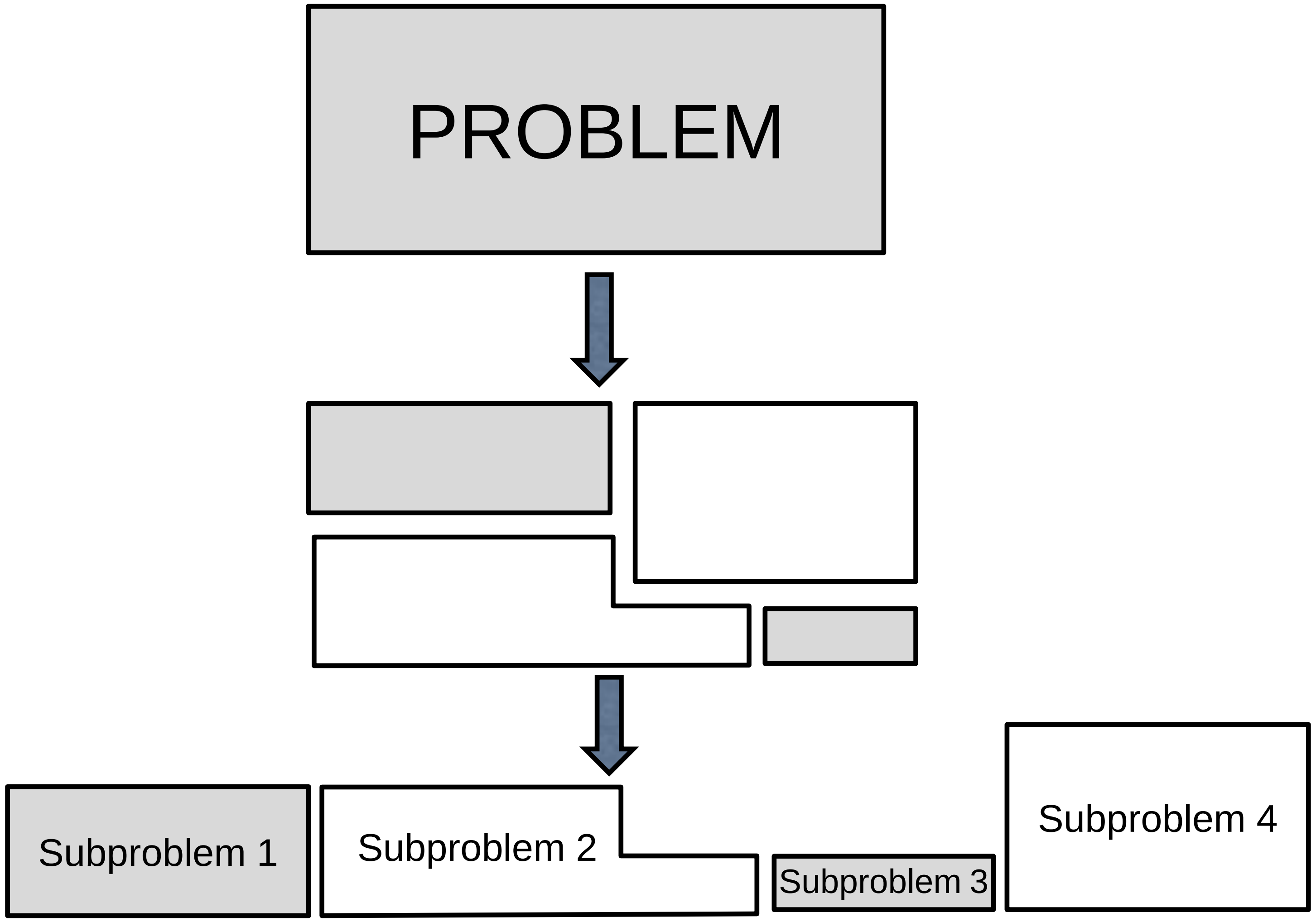


FIGURE 1.1 The process of analysis

Analyzing Problem

(Source: Pfleeger & Atlee)

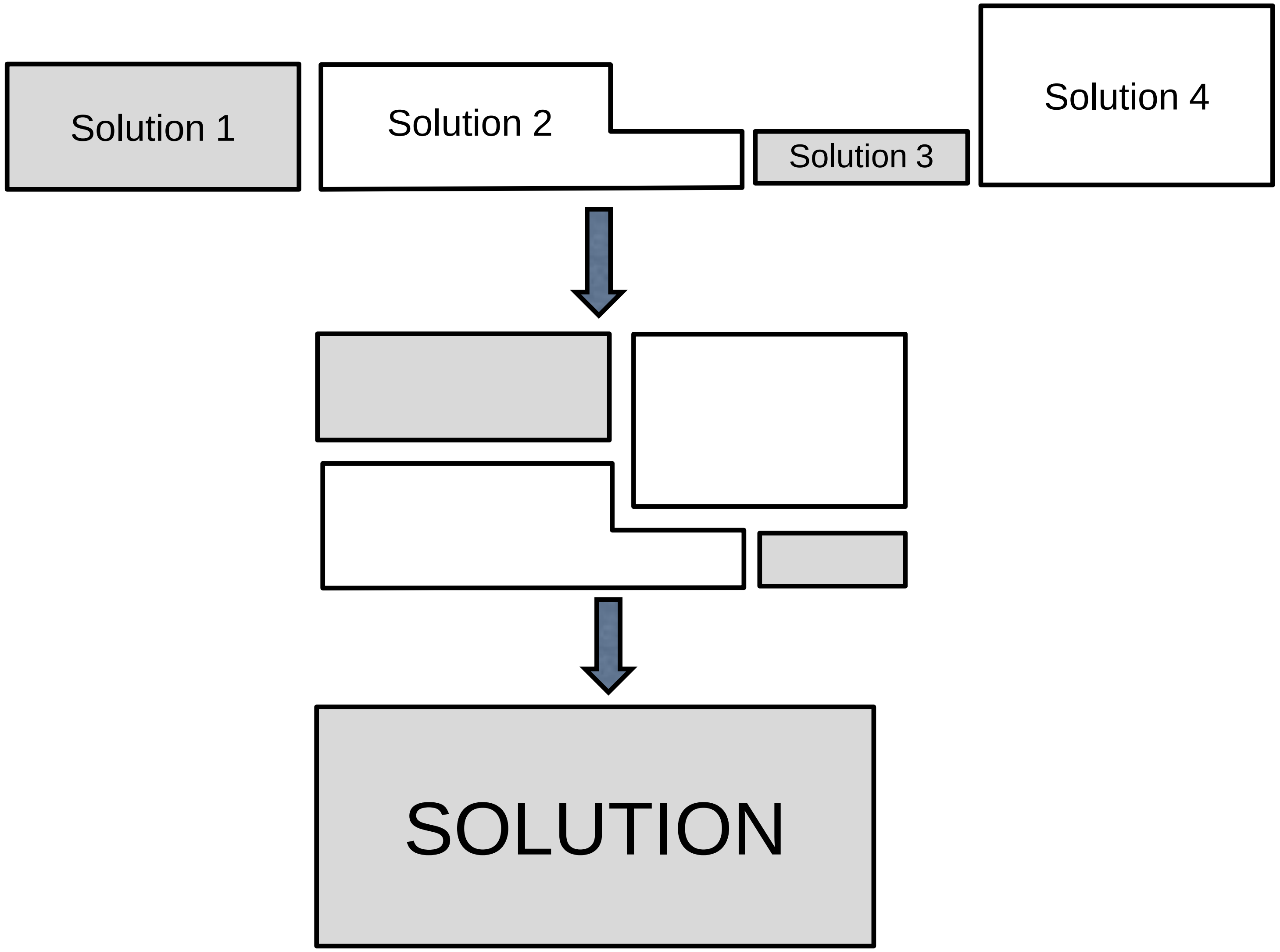


FIGURE 1.2 The process of synthesis

Synthesizing Solution

(Source: Pfleeger & Atlee)

What is Software Engineering?

- In solving a problem, we employ a variety of **methods**, **tools**, **procedures** and **paradigms**.
    - ✤ Method or technique: a formal procedure for producing some results.
    - ✤ Tool: an instrument or automated system for accomplishing something in a better way.
    - ✤ Procedure: a combination of tools and techniques that, in concert, produce a particular product.
    - ✤ Paradigm: a particular approach or philosophy for building software.

What is Software Engineering?

- 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.
- 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).

Relationship Between Software Engineering and Computer Science

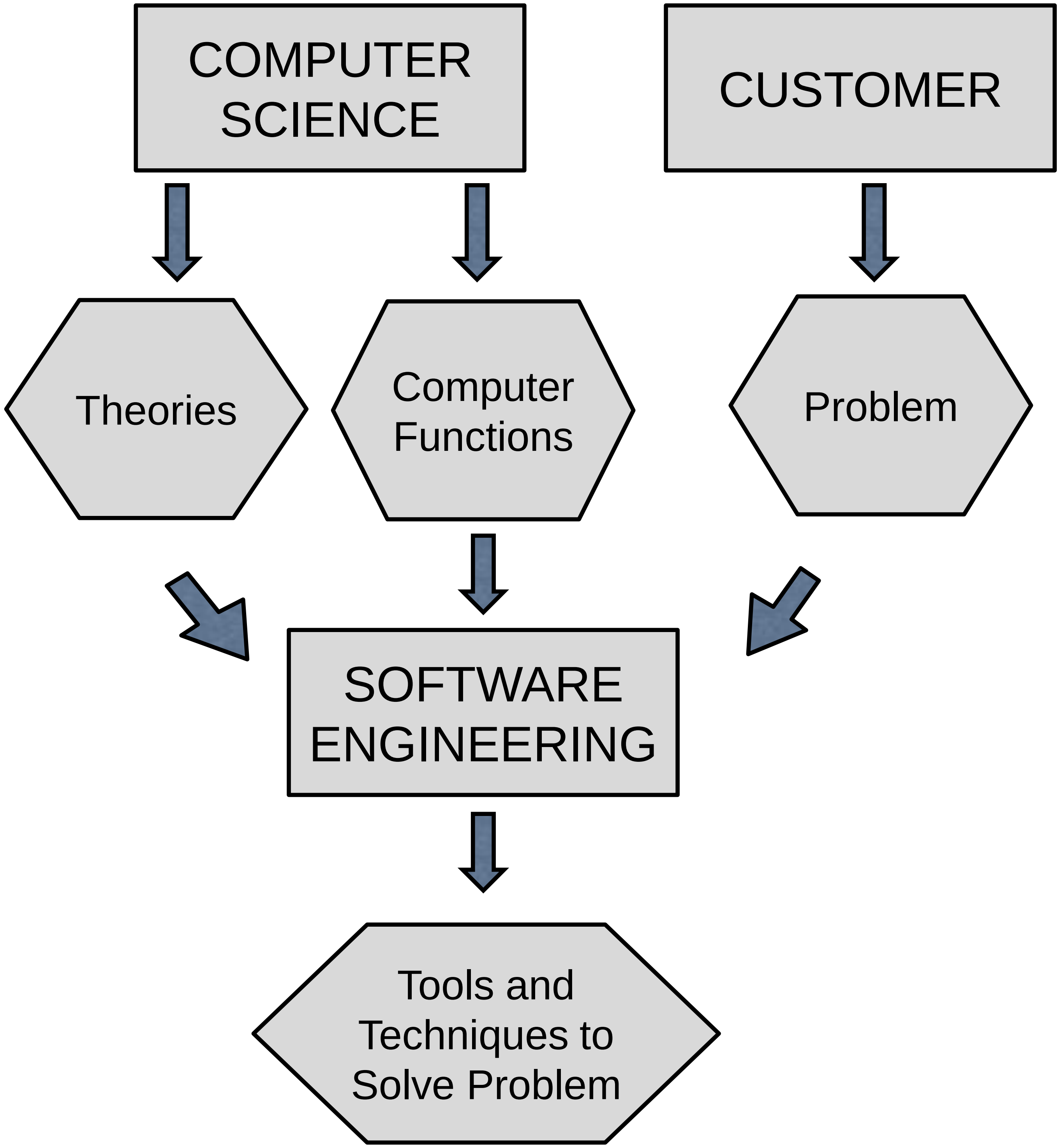
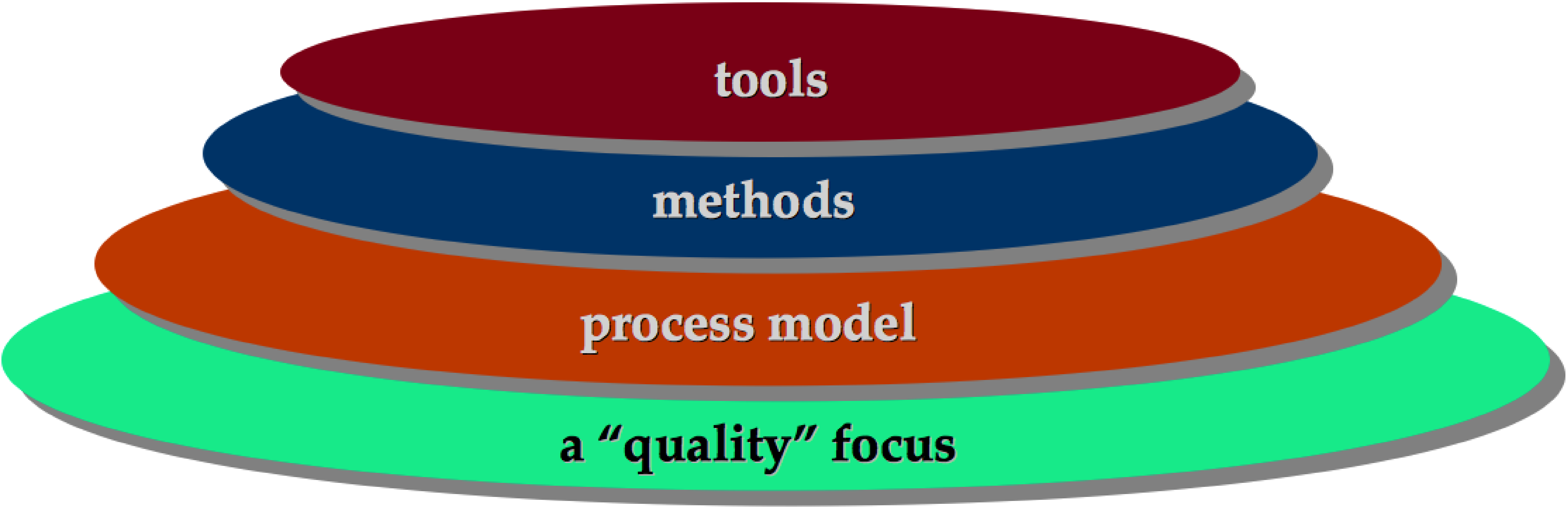


FIGURE 1.3 The relationship between computer science and software engineering

Source: Pfleeger & Atlee

A Layered Technology



Software Engineering

Process

- Process layer is the **foundation**.
  - the **glue** that holds the technology layers together
  - **enables rational and timely development**
  - **defines framework**
  - forms the **basis for management control** of software project
  - **establishes the context** in which **technical methods** are applied, **work products** are produced, **milestone** are established, **quality** is ensured, and **change** is properly managed.

Method

- Method layer
  - ✤ **provide the technical how-to's** for building software
  - ✤ **encompass a broad array of tasks** (include communication, requirement analysis, design modeling, program construction, testing, and support)
  - ✤ **rely on a set of basic principles** that govern each area of the technology and include modeling activities and other descriptive techniques.

Tools

- Tools layer
  - **provide automated or semiautomated support** for the process and the methods

Framework Activities

- Communication
- Planning
- Modeling
  - Requirement analysis & definition
  - Design
    - Architecture design / System design
    - Component design / Program design
- Construction
  - Code generation
  - Testing
    - Unit testing
    - Integration testing
    - System testing
- Deployment
    - System delivery
    - Maintenance

Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

Why do we need SE?

- Can’t we just go straight into coding?
- You can, but...
  - ✤ Many professional environments do not endorse ad-hoc activities such as ‘brute force, trial-and-error’ coding
  - ✤ Even if it does succeed, can it be repeated in the next project?
- It seems a lot of work initially, but the benefits will be worth it in the long run

The Essence of Practice

- Polya suggests:

  1. Understand the problem (communication and analysis).
  2. Plan a solution (modeling and software design).
  3. Carry out the plan (code generation).
  4. Examine the result for accuracy (testing and quality assurance).

1 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?

2

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?

3

Carry Out the Plan

- ***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?

4 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?

Q & A