

INTRODUCTION TO SOFTWARE ENGINEERING

1. INTRODUCTION TO SE

Nguyen Thanh Hung


hungnt@soict.hust.edu.vn



Content

1. What is Software?
2. Software Categories
3. What is SE?
4. Problems in SE

Content

- 
1. What is Software?
 2. Software Categories
 3. What is SE?
 4. Problems in SE

What is Software?

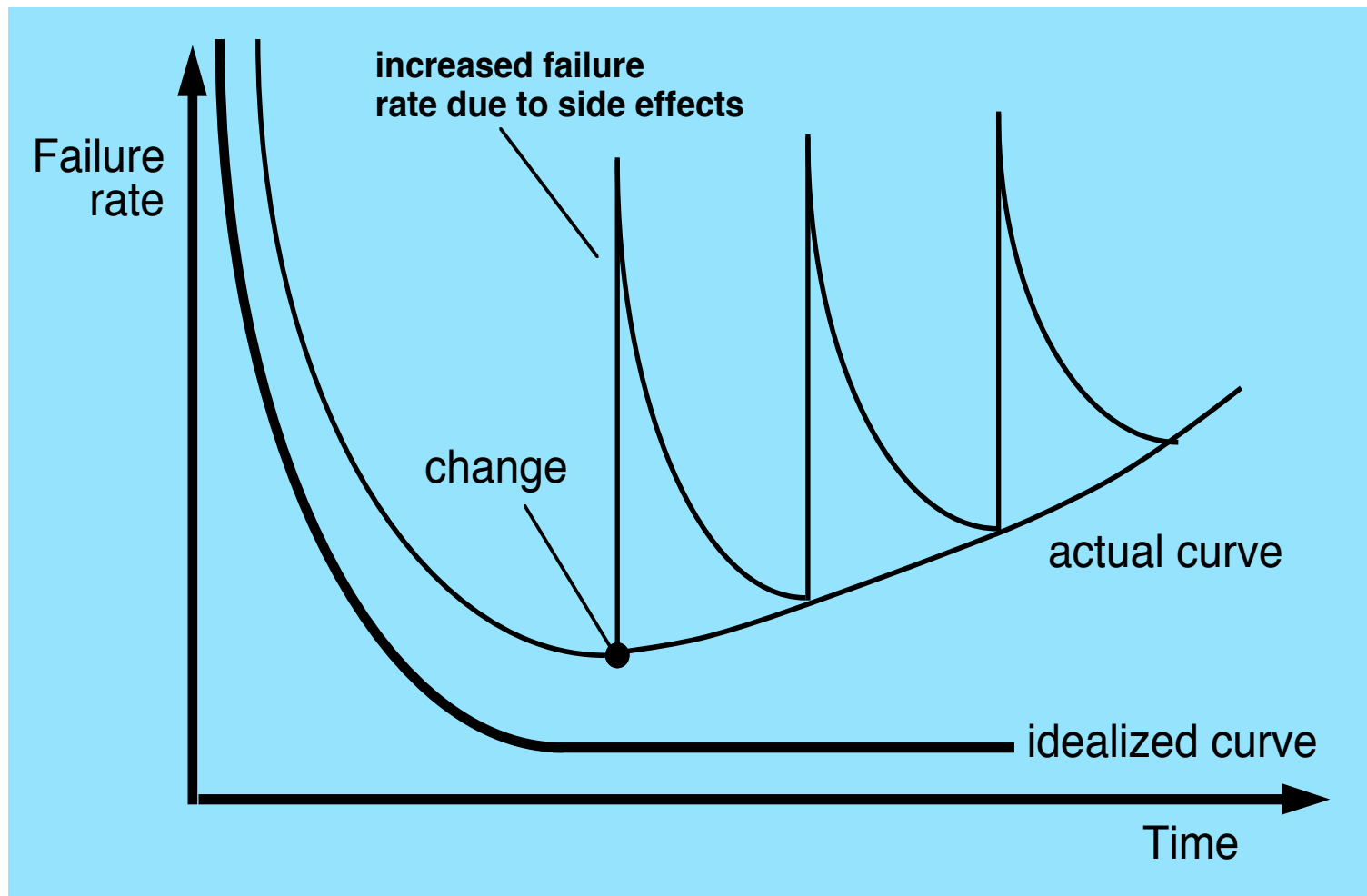
Software is:

- (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.

What is Software?

- Software is developed or engineered, it is not manufactured in the classical sense.
- Software doesn't “wear out”
- Although the industry is moving toward component-based construction, most software continues to be custom-built.

Wear vs. Deterioration



Content

1. What is Software?

→ 2. Software Categories

3. What is SE?

4. Problems in SE

Software Categories

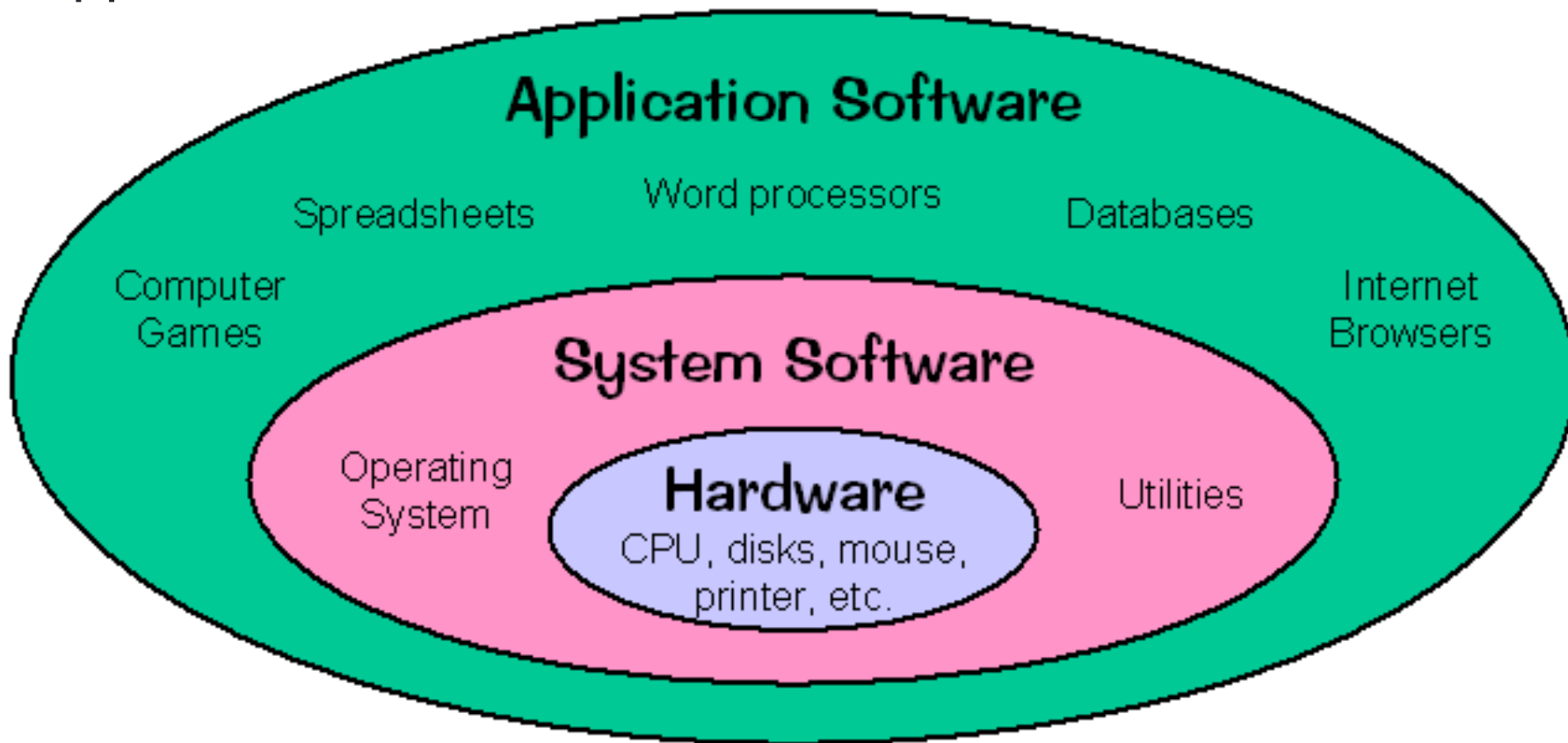
- System software
- Application software
- Engineering/scientific software
- Embedded software
- Product-line software
- Web applications
- AI software

Software Category – Video



System vs. Application Software

- Operate the computer hardware and to provide a platform for running application software
- Help the user to perform specific tasks for the benefit of user



Software Categories

- Open world computing – pervasive, distributed computing
- Ubiquitous computing – wireless networks
- Netsourcing – the Web as a computing engine
- Open source software
 - “free” source code open to the computing community (a blessing, but also a potential curse!)
- Commercial software

Legacy Software

Why must it change?

- software must be **adapted** to meet the needs of new computing environments or technology.
- software must be **enhanced** to implement new business requirements.
- software must be **extended to make it interoperable** with other more modern systems or databases.
- software must be **re-architected** to make it viable within a network environment.

Characteristics of WebApps - I

- **Network intensiveness.** A WebApp resides on a network and must serve the needs of a diverse community of clients.
- **Concurrency.** A large number of users may access the WebApp at one time.
- **Unpredictable load.** The number of users of the WebApp may vary by orders of magnitude from day to day.
- **Performance.** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.
- **Availability.** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a “24/7/365” basis.

Characteristics of WebApps - II

- **Data driven.** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end-user.
- **Content sensitive.** The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.
- **Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously.
- **Immediacy.** Although *immediacy*—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks.
- **Security.** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application.
- **Aesthetics.** An undeniable part of the appeal of a WebApp is its look and feel.

Content

1. What is Software?
2. Software Categories
- 3. What is SE?
4. Problems in SE

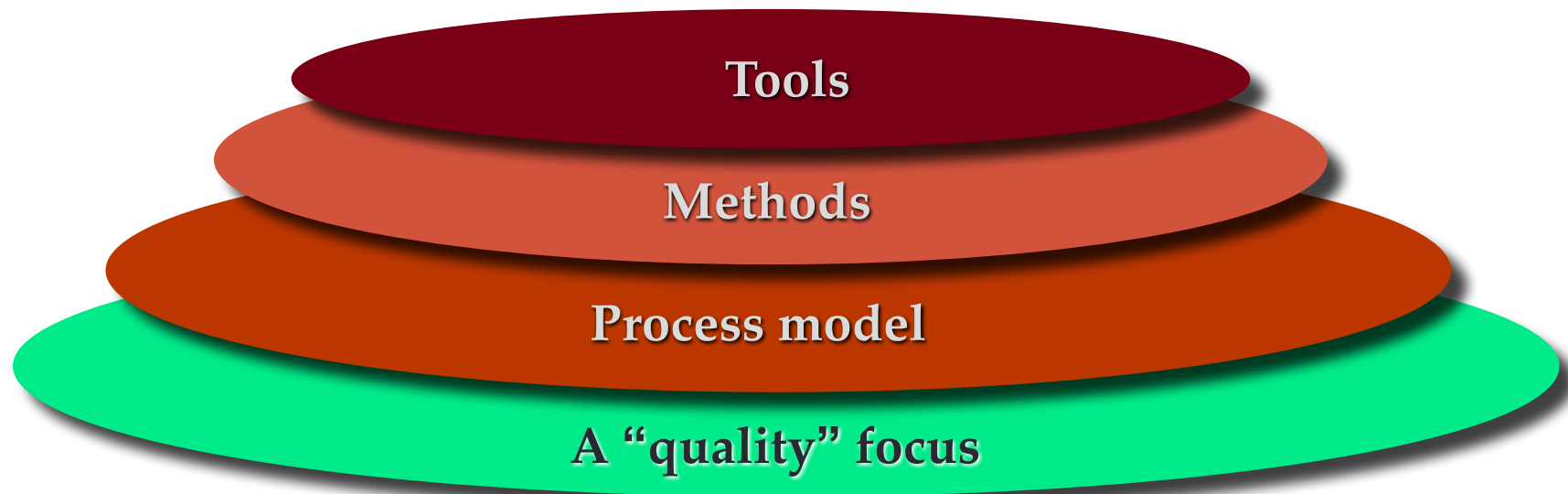
Software Engineering

- 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*
- 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**.*

Software Engineering

- 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).*

A Layered Technology



Software Engineering

A Process Framework

Process framework

Framework activities

Work tasks

Work products

Milestones & deliverables

QA checkpoints

Umbrella Activities

Framework Activities

- Communication
- Planning
- Modeling
 - Analysis of requirements
 - Design
- Construction
 - Code generation
 - Testing
- Deployment

Umbrella Activities

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

Adapting a Process Model

- the overall flow of activities, actions, and tasks and the interdependencies among them
- the degree to which actions and tasks are defined within each framework activity
- the degree to which work products are identified and required
- the manner which quality assurance activities are applied
- the manner in which project tracking and control activities are applied
- the overall degree of detail and rigor with which the process is described
- the degree to which the customer and other stakeholders are involved with the project
- the level of autonomy given to the software team
- the degree to which team organization and roles are prescribed

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).

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?

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 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?

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?

Hooker's General Principles

- 1: *The Reason It All Exists*
- 2: *KISS (Keep It Simple, Stupid!)*
- 3: *Maintain the Vision*
- 4: *What You Produce, Others Will Consume*
- 5: *Be Open to the Future*
- 6: *Plan Ahead for Reuse*
- 7: *Think!*

Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth, *but ...*
- Invariably lead to bad decisions, *therefore ...*
- Insist on reality as you navigate your way through software engineering

How It all Starts

- SafeHome:
 - Every software project is precipitated by some business need
 - the need to correct a defect in an existing application;
 - the need to the need to adapt a ‘legacy system’ to a changing business environment;
 - the need to extend the functions and features of an existing application, or
 - the need to create a new product, service, or system.

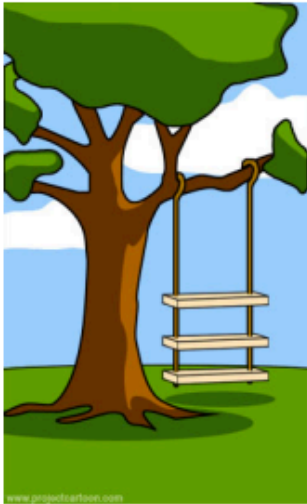
Content

1. What is Software?
2. Software Categories
3. What is SE?

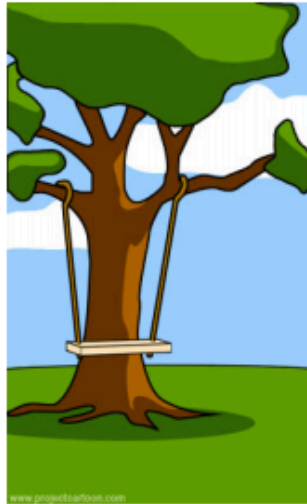


4. Problems in SE

What is the software that you want to build?



How the customer explained it



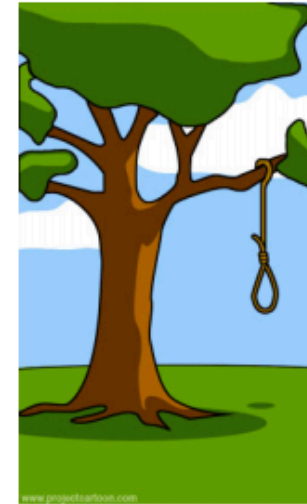
How the project leader understood it



How the analyst designed it



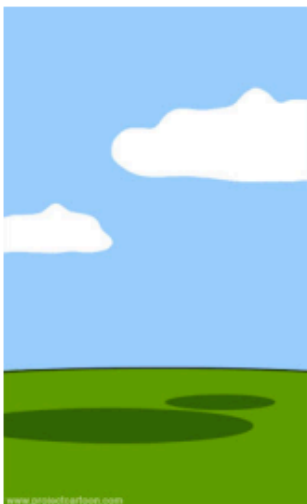
How the programmer wrote it



What the beta testers received



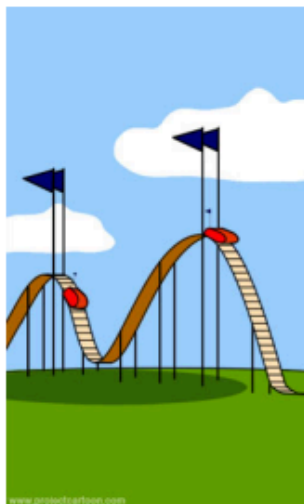
How the business consultant described it



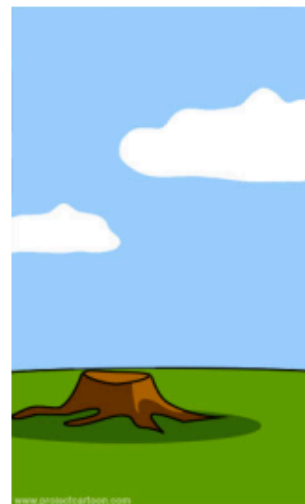
How the project was documented



What operations installed



How the customer was billed



How it was supported



What marketing advertised



What the customer really needed

Software Development Problems

- User or business needs not met
- Requirements not addressed
- Modules not integrating
- Difficulties with maintenance
- Late discovery of flaws
- Poor quality of end-user experience
- Poor performance under load
- No coordinated team effort
- Build-and-release issues