Chapter 1
Software and Software Engineering

Software Engineering: A Practitioner's Approach
by Roger S. Pressman and Bruce R. MAXIM

What is Software?

- When we write a program for computer we named it as software. But software is not just a program; many things other than the program are also included in software.

Some of the constituted items of software are described below.

***Program*:** The program or code itself is definitely included in the software.

***Data*:** The data on which the program operates is also considered as part of the software.

***Documentation*:** Another very important thing that most of us forget is documentation. All the documents related to the software are also considered as part of the software.

Software Engineering :we can define ***software engineering*** as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

IEEE defines software engineering as: *The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.*

What is the work product? From the point of view of a software engineer, the work product is the set of programs, content (data), and other work products that are computer software. But from the user's viewpoint, the work product is the resultant information that somehow makes the user's world better.

Software's Dual Role

- Software is a product
  - *Transforms* information - produces, manages, acquires, modifies, displays, or transmits information
  - Delivers computing potential of hardware and networks
- Software is a vehicle for delivering a product
  - Controls other programs (operating system)
  - Effects communications (networking software)
  - Helps build other software (software tools & environments)

Software Applications

- system software
- application software
- engineering/scientific software
- embedded software
- product-line software
- web applications
- AI software

System software—a collection of programs written to service other programs. Some system software (e.g., compilers, editors, and file management utilities) processes complex, but determinate information structures. Other systems applications (e.g., operating system components, drivers, networking software, telecommunications processors)

Application software—stand-alone programs that solve a specific business need. Applications in this area process business or technical data in a way that facilitates business operations or management/technical decision making.

Engineering/scientific software—has been characterized by "number crunching" algorithms. Applications range from astronomy to volcanology,

rom automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.

Embedded software—resides within a product or system and is used to implement and control features and functions for the end user and for the system itself. Embedded software can perform limited and esoteric functions (e.g., key pad control for a microwave oven)

Product-line software—designed to provide a specific capability for use by many different customers. Product-line software can focus on a limited and esoteric marketplace (e.g., inventory control products) or address mass consumer markets (e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications).

Web applications—called "WebApps," this network-centric software category spans a wide array of applications. In their simplest form, WebApps can be little more than a set of linked hypertext files that present information using text and limited graphics.

Artificial intelligence software—makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straightforward analysis. Applications within this area include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem proving, and game playing.

Hardware vs. Software

Hardware	Software
Manufactured	Developed/engineered
Wears out	Deteriorates
Built using components	Custom built
Relatively simple	Complex

Manufacturing vs. Development

- Once a hardware product has been manufactured, it is difficult or impossible to modify. In contrast, software products are routinely modified and upgraded.
- In hardware, hiring more people allows you to accomplish more work, but the same does not necessarily hold true in software engineering.
- Unlike hardware, software costs are concentrated in design rather than production.

Wear vs. Deterioration

Hardware wears out over time

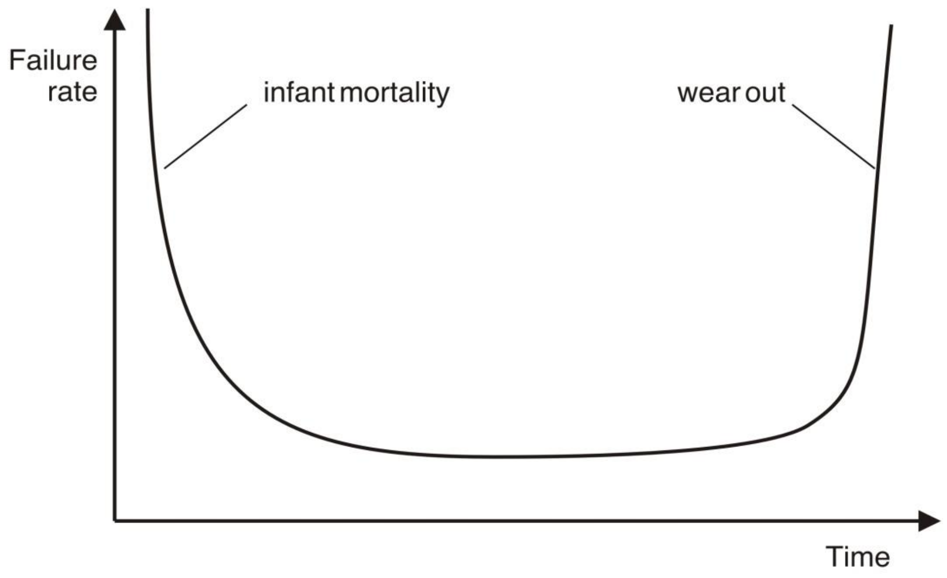
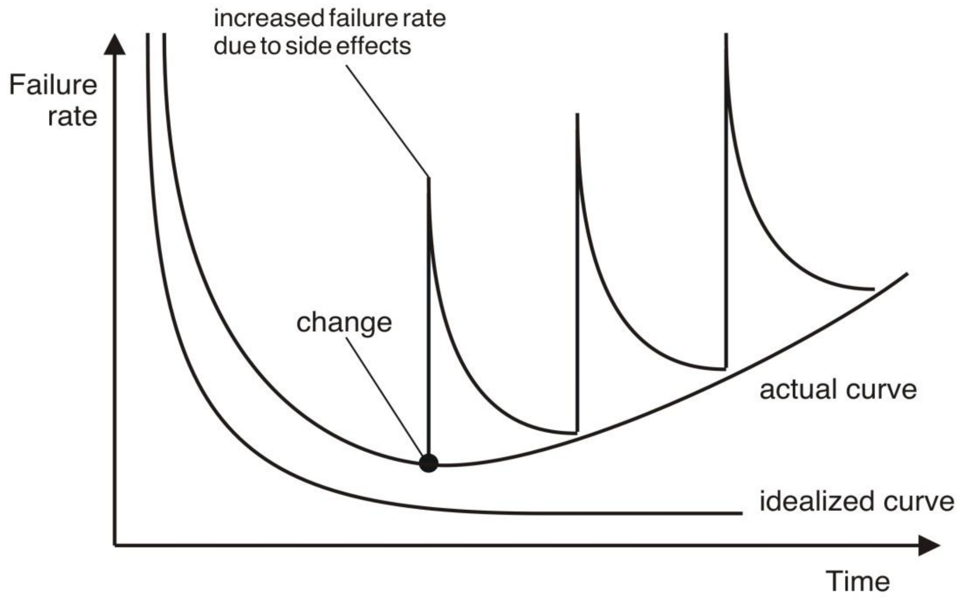


Figure depicts failure rate as a function of time for hardware. The relationship, often called the “bathtub curve,” indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the 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

Wear vs. Deterioration

Software deteriorates over time



Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, therefore, the failure rate curve for software should take the form of the “idealized curve” shown in Figure . Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected and the curve flattens as shown. The idealized curve is a gross over simplification of actual failure models for software. However, the implication is clear—software doesn’t wear out. But it does *deteriorate!*

Component Based vs. Custom Built

- Hardware products typically employ many standardized design components.
- Most software continues to be custom built.
- The software industry does seem to be moving (slowly) toward component-based construction.

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. Modern reusable components encapsulate both data and the processing that is applied to the data, enabling the software engineer to create new applications from reusable parts. For example, today's interactive user interfaces are built with reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms. The data structures and processing detail required to build the interface are contained within a library of reusable components for interface construction.

Legacy Software

The older programs—often referred to as *legacy software*

Why must it change?

- It must be fixed to eliminate errors.
- It must be enhanced to implement new functional and non-functional requirements
- 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.

laws of software evolution

Observing that most software is subject to change in the course of its existence, the authors set out to determine laws that these changes will typically obey, or must obey in order for the software to survive.

Lehman qualified the application of such laws by distinguishing between three categories of software:

- An *S*-program is written according to an exact specification of what that program can do
- A *P*-program is written to implement certain procedures that completely determine what the program can do (the example mentioned is a program to play chess)
- An *E*-program is written to perform some real-world activity; how it should behave is strongly linked to the environment in which it runs, and such a program needs to adapt to varying requirements and circumstances in that environment
- The laws are said to apply only to the last category of systems.

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 …

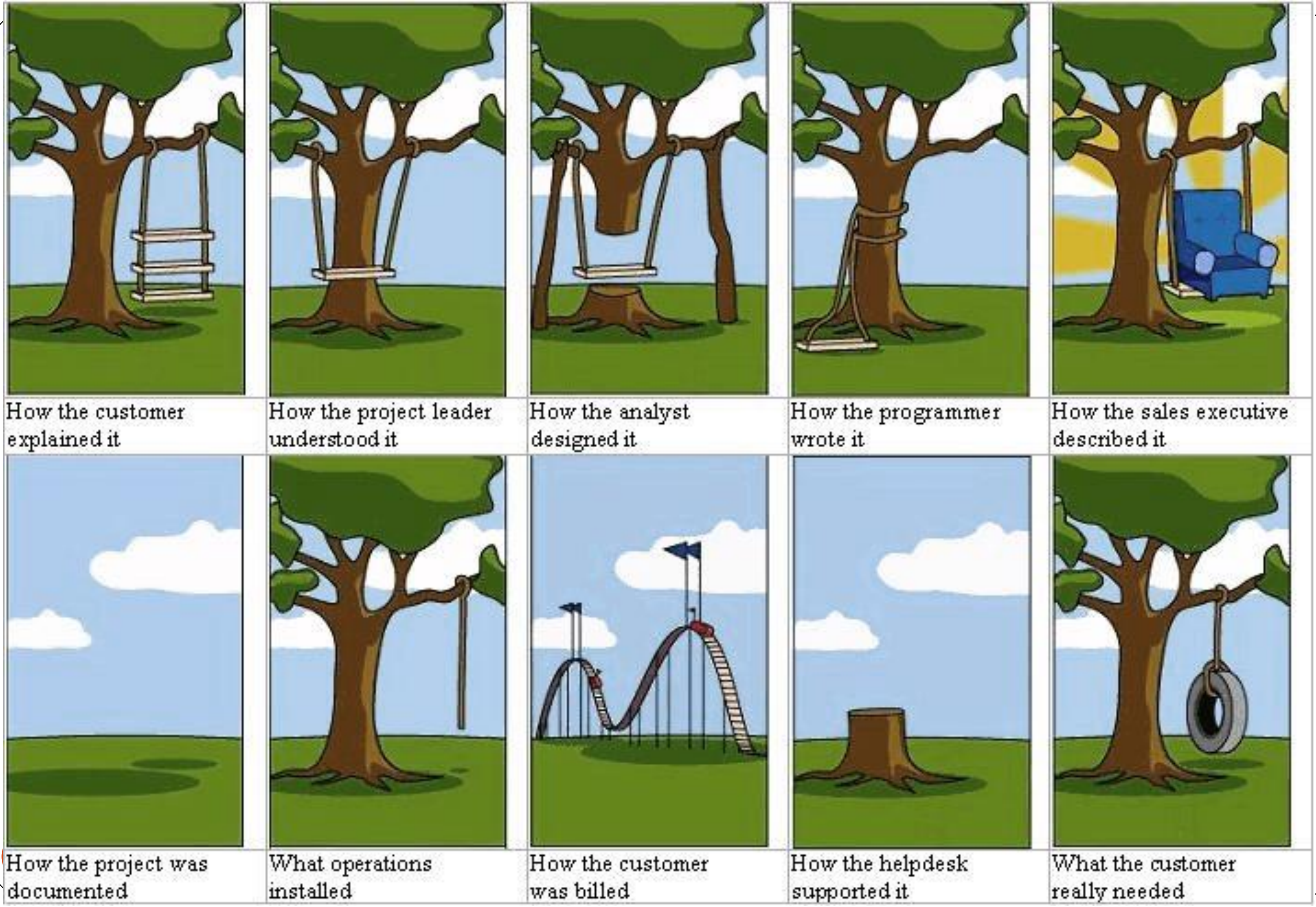
- Insist on reality as you navigate your way through software engineering

Software Myths

- If we get behind schedule, we can add more programmers and catch up.
- A general statement about objectives is sufficient to begin building programs.
- Change in project requirements can be easily accommodated because software is flexible.

Software Myths

- Once we write a working program, we're done.
- Until I get the program running, I have no way of assessing its quality.
- The only deliverable work product for a successful project is the working program.
- Software engineering will make us create too much documentation and will slow us down.


How the customer explained it
How the project leader understood it
How the analyst designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed