TOPICS

The Nature and History of Software Development

Problems with Software Development

Software Engineering Paradigms and Technology

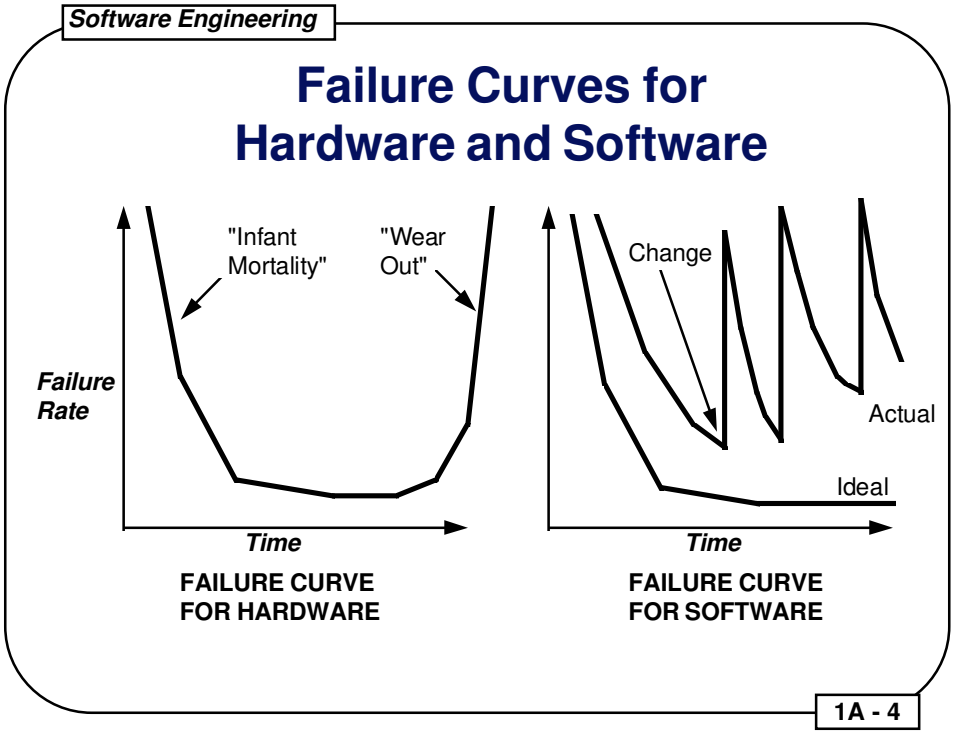
THE NATURE OF SOFTWARE

- Characteristics of Software
- Failure Curves for Hardware and Software
- Software Components
- Software Configuration
- Software Application Areas

Characteristics of Software

- **Software is *programs*, *documents*, and *data*.**
- **Software is developed or engineered; it is not manufactured like hardware.**
- **Software does not wear out, but it does *deteriorate*.**
- **Most software is custom-built, rather than being assembled from existing components.**
- **Software is a *business opportunity*.**

1. Many people have the non-engineering view of software:
   - as computer programs (*i.e.*, source code and/or executables),
   - as data structures (*e.g.*, data base schemas), and
   - as operation and user documentation (usually created as an afterthought when the "real" work is done)
2. A major factor in the speed at which quality software is developed is the failure to reuse software:
   - there are few reusable component libraries,
   - there is a bias against using "old" routines or routines "not invented here", and
   - software as a creative art is a perception that is held by many people.
3. Software has become a *business opportunity*, where the success or failure of a business (and the jobs of the people associated with it) depends upon the timely development of quality software. The customer is demanding both high quality in the software (and once a business has a reputation of putting out "junk", word gets around quickly) and timeliness of delivery (the customer wants the software *now*).



Hardware tends to have a wear-in time during which it has a higher probability of failure. This is generally referred to as *infant mortality*. Once the initial period is passed, hardware tends to operate without failure until components age enough to cause breakdown.

Moral

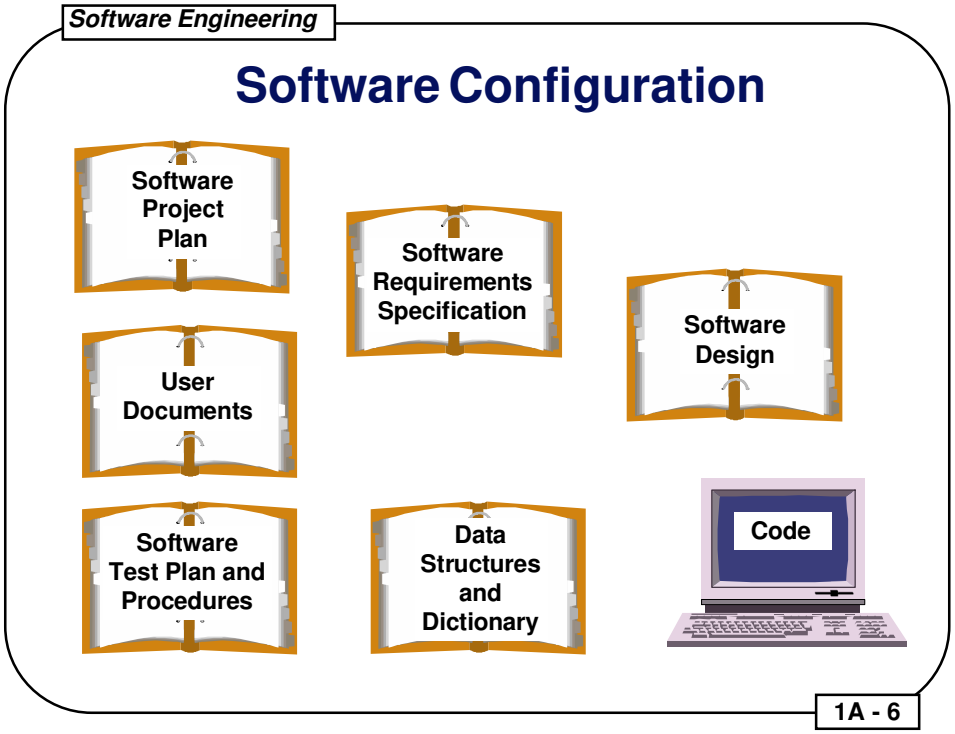
Don't buy extended warranty contracts.

Standard warranty is usually long enough to pass through the infant mortality period.

Software also shows an early error rate, but updates should remove the most obvious problems which render the software unreliable. Updates for added functionality often add more errors, as is shown by the spikes on the failure curve for software. As updates are made, more latent errors appear in the software to make it inherently less reliable until the software is finally considered unreliable enough to stop using the software product or to perform a major redesign and rewrite of the software.

Software Components

- Software programs, or software systems, consist of *components*.
- A set of components which comprise a logical unit of software is called a *software configuration item*.
- Reuse and development of reliable, trusted software components improves software *quality* and *productivity*.
- Computer language forms:
  - Machine level (microcode, digital signal generators)
  - Assembly language (PC assembler, controllers)
  - High-order languages (FORTRAN, Pascal, C, Ada, ...)
  - Specialized languages (LISP, OPS5, Prolog, ...)
  - Fourth generation languages (databases, windows apps)



Composition of Software

The software we develop is composed of these parts, also known as *software configuration items*:

- **Software Project Plan** - A document which details the tasks, schedules, needed resources, and approach to carry out development. This is the first document produced and it includes cost details.
- **Software Requirements Specification** - A document which identifies *what* is required of the software (as opposed to the design document, which describes *how* to implement the software). This document includes information on how implementation of the requirements will be verified (*i.e.*, some initial test considerations). This very important document is often quite time consuming to produce.
- **Software Test Plan and Procedures** - A document which describes the test methods, approaches, procedures, and the support required for testing the software code components and the integrated software system. This document includes test data and expected results and is developed during both the requirements definition and design phases of the project.
- **Data Structures and Dictionary** - The Data Dictionary documents all data structures and the definitions of terms, variables, and other items of interest regarding the details of the data in the system. It supports software design, coding, and maintenance and is developed during the requirements and design phases.
- **Software Design Document** - A document which clearly details the behavior and structure of the system as a whole and each software code component.
- **User Documents** - These are user guides, reference guides, application notes, and other items deemed necessary for the users.
- **Code** - The compilable source code of the system.

Software Development Activities

- Planning Activity
  - Software Project Plan
- Requirements Definition Activity
  - Software Requirements Specification
  - Software Test Plan and Procedures
  - Data Structures and Dictionary
  - User Documents
- Design Activity
  - Software Design Documents
  - Software Test Plan and Procedures
  - Data Structures and Dictionary
- Coding and Testing Activity
  - Code
  - Software Test Plan and Procedures
- Delivery and Maintenance Activity
  - User Documents
  - Others as needed

When are the Software Configuration Items Produced?

- The *Software Configuration Items* are drafted, reviewed, revised, etc., at many points throughout the activities performed during the development of the software. Seldom is a Software Configuration Item felt to be completely finished.
- All *Software Configuration Items* are placed under *configuration control*, allowing for them to be changed and all changes to them to be tracked. Any particular version of any of the configuration items may be recreated when desired.
- The control of the Software Configuration Items extends from the planning stages of the project through the maintenance activities -- the entire life of the software.

Software Application Domains

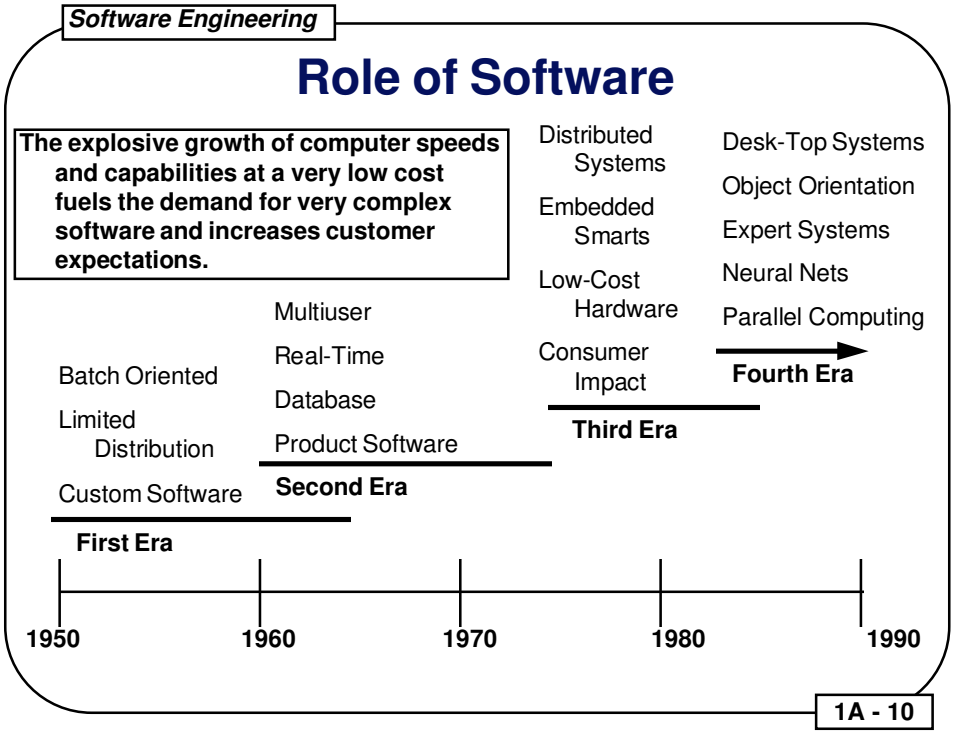
- System
  - compilers
  - editors
  - Operating Systems
- Real Time
  - machine control
  - auto controls
- Business
  - databases
  - stock management
- Personal Computer
  - all non-realtime above
- Embedded
  - appliance control
  - FPGA programs
  - auto controls
- Engineering and Scientific
  - simulation
  - computer-aided design
  - "number crunching"
- Artificial Intelligence
  - expert systems
  - neural networks

There are many, many diverse application domains in which software is being developed, and, for each domain and each organization within each domain, there are many, many different ways to develop this software:

- ad hoc, which is by far the most common
- using different accepted engineering methodologies, such as
  - the classic "waterfall" approach
  - rapid prototyping
  - fourth generation techniques
  - the spiral model
  - a combination of the above
- using different sets of procedures, which include
  - documentation standards
  - coding standards
  - test standards
  - procedures for estimating cost and schedule

HISTORY OF SOFTWARE DEVELOPMENT

- Role of Software
- Industrial View

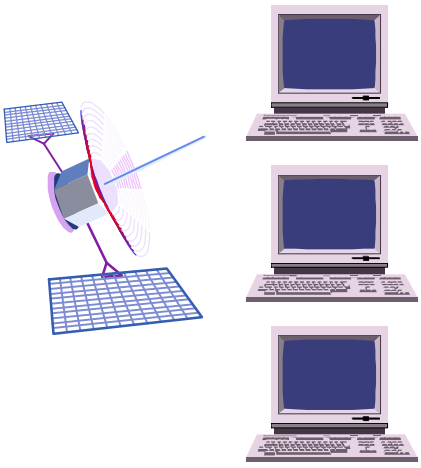


1. Early years (to about 1970):
    - large, expensive, few, protected computers
    - small programs inefficiently written
    - major constraints (memory, speed, I/O)
    - non-realtime batch-oriented software; single user
    - single programmer per program
2. Middle years (1970 to 1990):
    - realtime software development
    - multiple programmer teams
    - software development industry emerges
    - emerging interest in engineering the development of software
    - department-level computers make them more accessible; multiuser
3. Later years (1980 to 1990):
    - personal computer makes computing highly accessible
    - very large software industry develops
    - large programs and software systems emerge
    - hardware is distributed using networks
    - communications using computers evolves
    - software becomes highly departmentalized

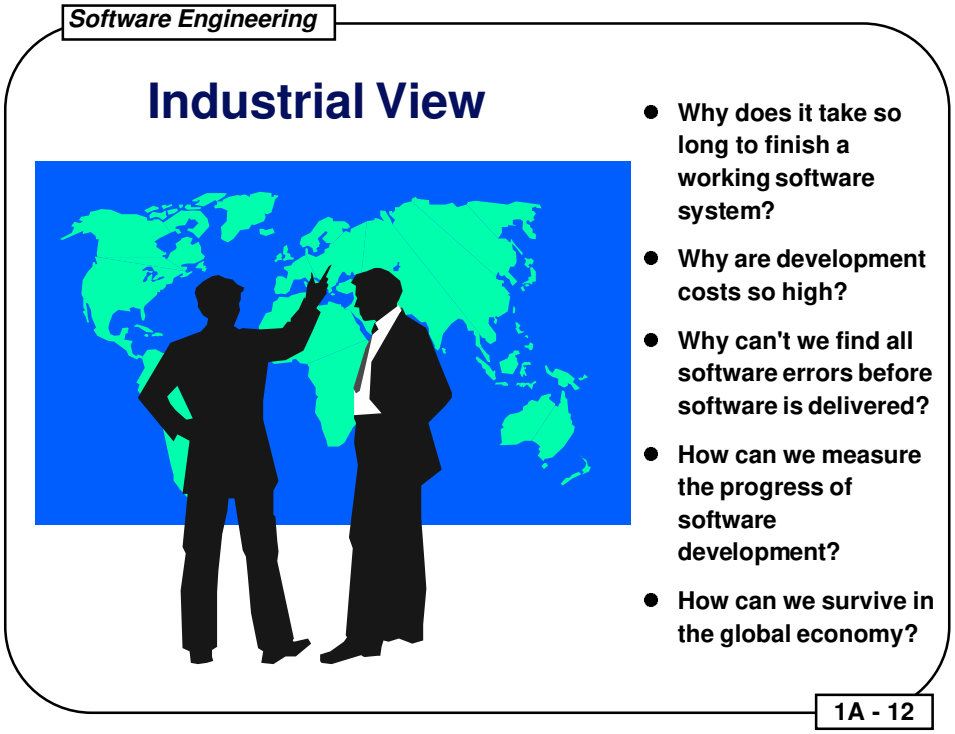
Role of Software, Continued

Where Do We Go From Here?

- **Parallel computing to extend speed of computation**
- **Object-oriented methods of software design**
- **Software frameworks evolve to handle larger and multiprogram systems**
- **Heavy dependence on graphics interfaces**
- **Artificial intelligence and neural computing become useful**
- **National computing motivates huge software systems**
- **Advanced programming languages**



- One concept which dominates all of these ideas is that high quality software is required in all cases.
- The software engineering community has learned that two things are needed to develop high quality software:
  - a good software development process
  - technological innovations which support the selected process
- Technology alone, without the process, is not enough and often adds to the risk and the problems rather than reducing the risk and the problems.



1. Early software development was considered to be an "art form"
2. Formal methods did not exist or were not followed
3. Programming education mainly by trial and error
4. Example of problems: Operating System for the IBM 360 (data extracted from **The Mythical Man-Month** by Fredrick Brooks, Addison-Wesley, 1975)
    - large software product (almost 1 million lines of code)
    - as errors were fixed, more errors were produced
    - adding people to the project made things worse
    - few formal methods of design were known or used
    - project was abandoned and the operating system was completely rewritten
    - project had a major impact on producing formal methods in software engineering