Chapter 1
History of Software Engineering

The Pioneering Era (1955-1965)

- New computers were coming out almost every year or two, rendering existing ones obsolete.

- Software people had to rewrite all their programs to run on these new machines.

- Programmers did not have computers on their desks and had to go to the "machine room".

The Pioneering Era - 2

- Jobs were run by
  - signing up for machine time or by operational staff.
  - putting punched cards for input into the machine's card reader and waiting for results to come back on the printer.
- The field was so new that the idea of management by schedule was non-existent.
- Making predictions of a project's completion date was almost impossible.

The Pioneering Era - 3

- Computer hardware was *application-specific*. Scientific and business tasks needed different machines.
- Due to the *need to frequently translate old software to meet the needs of new machines*, high-order languages like FORTRAN, COBOL, and ALGOL were developed.

The Pioneering Era - 4

- Hardware vendors gave away *systems software for free* as hardware could not be sold without software. A few companies sold the service of building custom software but no software companies were selling packaged software.

The Pioneering Era - 5

- The notion of *reuse* flourished. As software was free, user organizations commonly gave it away. Groups like IBM's scientific user group SHARE *offered catalogs of reusable components*.
- Academia did not yet teach the *principles of computer science*.
- *Modular programming and data abstraction* were already being used in programming.

The Stabilizing Era (1965-1980)

- The whole *job-queue system* had been *institutionalized* and so programmers no longer ran their jobs except for peculiar applications like on-board computers. To handle the jobs, an *enormous bureaucracy* had grown up around the central computer center.
- The major problem as a result of this bureaucracy was *turnaround time*, the time between job submission and completion. At worst it was *measured in days*.

The Stabilizing Era - 2

- *IBM 360* signaled the beginning of the *stabilizing era*.
- Largest software project to date ending the era of a faster and cheaper computer emerging every year or two.
- Software people could finally spend time writing new software instead of rewriting the old.
- The 360 also combined scientific and business applications onto one machine. It offered both binary and decimal arithmetic. With the advent of the 360, the organizational separation between scientific and business application people came to diminish and this had a massive impact on the sociology of the field. Scientific programmers who usually had bachelor degrees felt superior to business programmers who usually held only associate degrees.
- One scientific programmer remarked: "I don't mind working with business programmers, but I wouldn't want my daughter to marry one!"

The Stabilizing Era - 3

- The massive O/S, still coming largely free with the computer, controlled most of the services that a running program needed.
- The *job control language JCL* raised a whole new class of problems. The programmer had to write the program in a whole new language to tell the computer and OS what to do. JCL was the least popular feature of the 360.
- PL/I, introduced by IBM to merge all programming languages into one, failed.
- The demand for programmers exceeded the supply.

The Stabilizing Era - 4

- The notion of *timesharing*, using terminals at which jobs could be directly submitted to queues of various kinds was beginning to emerge, meeting with some resistance from traditionalists.
- As the software field stabilized, software became a corporate asset and its value became huge.
- *Stability* lead to the emergence of *academic computing disciplines* in the late 60's. However the *software engineering discipline did not yet exist*.

The Stabilizing Era - 5

- Many *"high-hype"* disciplines like *Artificial Intelligence* came into existence. As these new concepts could not be converted into predicted benefits, the credibility of the computing field began to diminish.
- "*Structured Programming*" burst on the scene *in the middle of this era*.
- *Standards organizations became control battle grounds*. The vendor who defined the standards could gain significant competitive advantage by making the standards match their own technology.

The Stabilizing Era - 6

- Although hardware vendors tried to put a brake on the software industry by keeping their prices low, software vendors emerged a few at a time.
- Most *customized applications* continued to be *done in-house*.
- Programmers still had to go to the "*machine room*" and did not have computers on their desks.

The Micro Era (1980-Present)

- The price of computing has dropped dramatically making ubiquitous computing (i.e., computing everywhere) possible. Now every programmer can have *a computer on his desk*.
- The old *JCL* has been *replaced by* the user-friendly *GUI*.

The Micro Era - 2

- The field still has its problems. The software part of the hardware architecture that the programmer must know about, such as the instruction set, has not changed much since the advent of the IBM mainframe and the first Intel chip. The most-used programming languages today are between 15 and 40 years old. The *Fourth Generation Languages* never achieved the *dream of "programming without programmers"* and the *idea is pretty much limited to report generation from databases*. There is an increasing clamor though for more and better software research.

A Simple Introduction to SW Engineering

What is software (SW)?

- SW is
  - not only *programs*
  - but also all *associated documentation*, and
  - *configuration data*

  that make these programs operate correctly.
- More specifically, a SW system consists of
  - *separate programs*
  - *configuration files* setting up these programs
  - *system documentation* describing the structure of the system in good detail
  - *user documentation* explaining how to use and operate the system.

Two Classes of SW Products

- *Generic* products
  - *stand-alone systems*
  - *sold* on open market *to any customer* such as
    - *word processors*,
    - *databases*,
    - *drawing packages*
- *Custom* or *bespoke* products
  - systems developed *specifically for a customer*

What is SW Engineering (SWE)?

- SW engineering is an *engineering discipline*
  - concerned with *all aspects of SW production* starting from the early stages of system specification through to the maintenance of the system after it has started to be used.
- *engineering discipline*:
  - implies solving a well-defined (or in SW engineering vaguely defined) problem optimally using resources (e.g., time, man power and machine power) and remaining within
    - organizational (i.e., the customer)
    - financial, and
    - other possible
  - constraints.
- *all aspects of SW production*:
  - encompasses
    - not only the technical processes
    - but also deals with project management, development of tools, methods and theories to support SW production.

Difference between SWE and Computer Science (CS)...?

- CS ...
  - ... is concerned with theories and methods which establish a basis for computers and SW systems while ...
- SWE ...
  - ... is concerned with the practical problems of producing SW.
- CS is as essential for SW engineers as ...
  - ... physics is for electrical or mechanical engineers.

What is a SW process?

- *Four fundamental activities* and associated results which produce a SW product.
  - *SW (requirements) specification*
  - *SW development ...*
    - *Design phase*
    - *Implementation phase*
  - *SW verification and validation (SW V&V)*
  - *SW evolution*

SW (requirements) specification ...

- ... answers the question: ***what?***
- ... starts with the definition of requirements by the requirements specification team
- ... proceeds with *requirements engineering process* including
  - *Requirements elicitation & analysis* composed of
    - Domain understanding / Requirements collection / Classification / Conflict resolution / Prioritization / Requirements checking
  - *Requirements validation*
    - ... Shows that the system defined by the above requirements is the actual expectation of the customer,
    - ... Complete & consistent, realizable by the existing technology, verifiable?
- ... ends up with *requirements management*
  - ... is the process of understanding and controlling changes to system requirements.

SW development ...

... answers the question: ***how?***

- SW that is expected to meet the specification must be produced.
- *Design phase*: alternative solutions to how the intended goals are accomplished
- *Implementation phase*: actual code development

SW verification and validation: (SW V&V)

- ... answers the following two questions:

  am I doing the right product?

  am I doing the product right?

  – SW must be verified & validated to ensure that it does what the customer wants

- *SW evolution:*

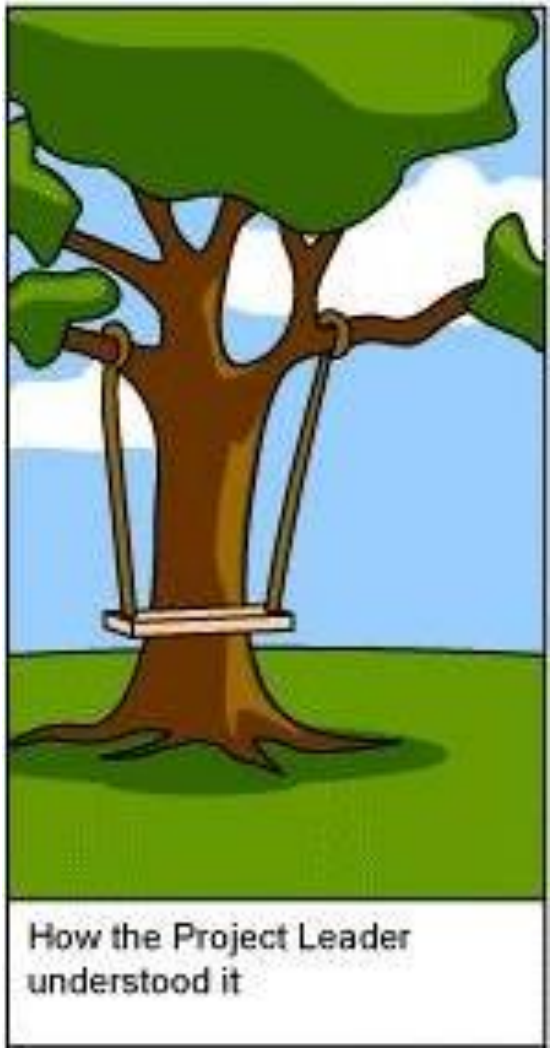
  – SW must evolve to meet the changing needs of the customer.

SW evolution ...

- SW must evolve to meet the changing needs of the customer.
- ... starts *after the delivery and on-site installation* of SW product
- *Maintenance* is an activity of this phase

Customer's Version of Requirements

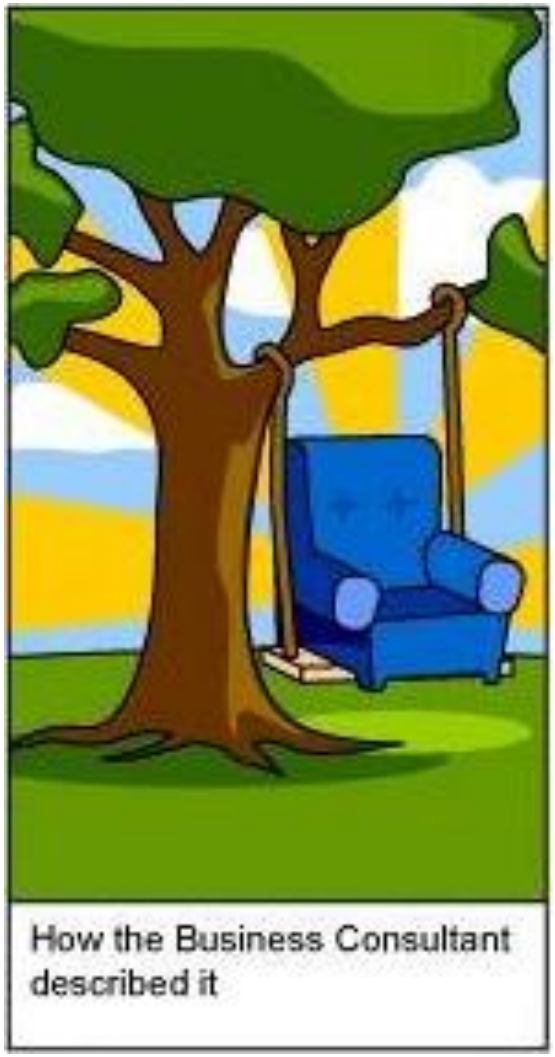
Project Leader's Understanding of Requirements



Design of SW by Specification

Implementation of SW

Business Consultant's Understanding of the SW

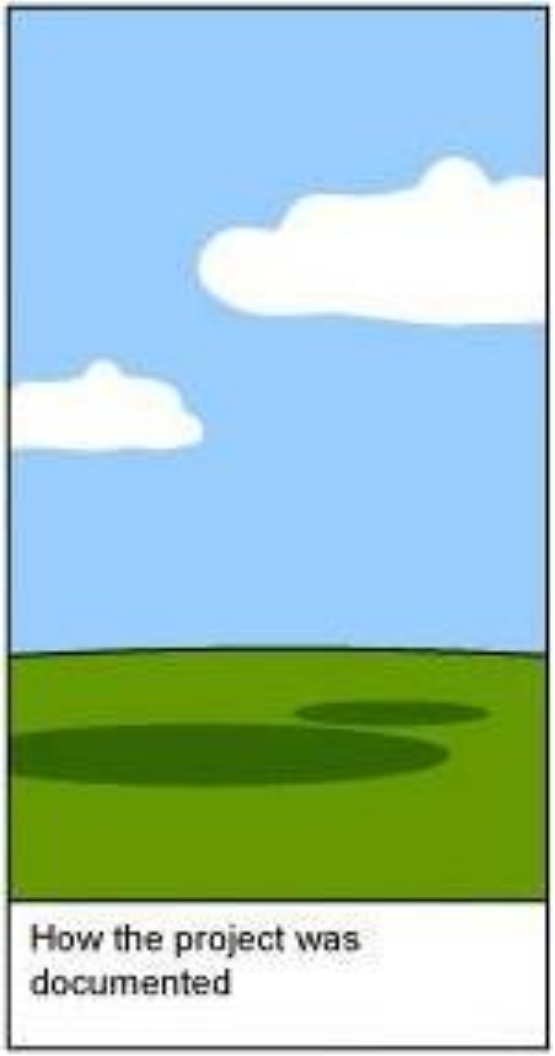


Documentation Quality of SW

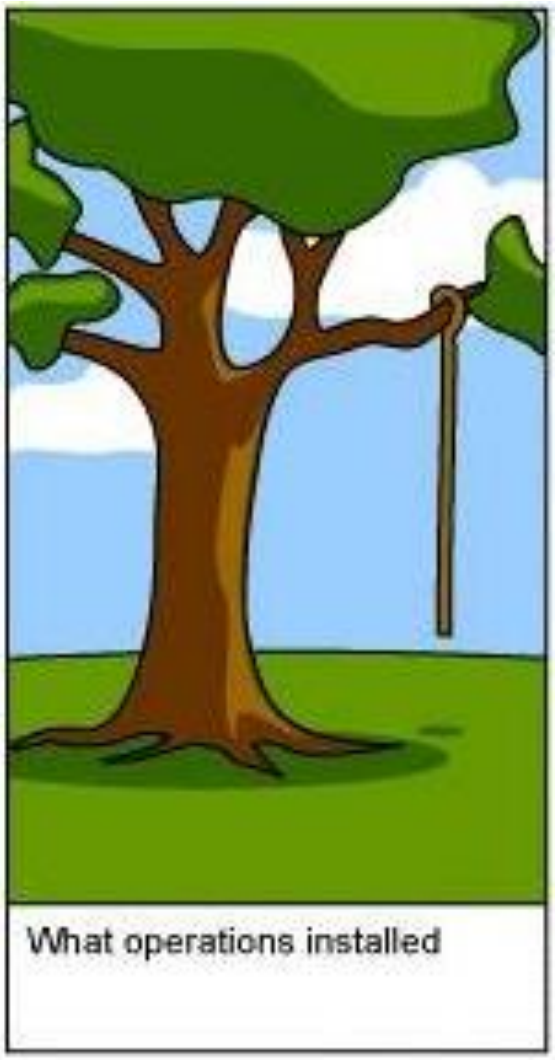
!!!

Here please remember how you documented your projects so far and please do not forget this figure when writing project reports!

!!!

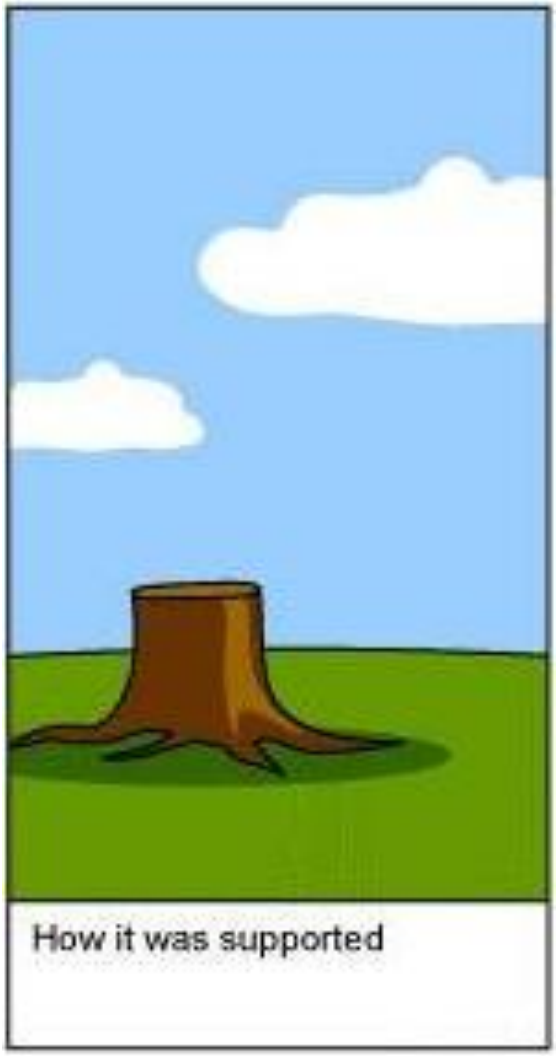


SW After On-site Installation



What Customer Pays For

Contribution of Computer Aids and SW Development Tools



The Real Need of Customer

What is a software process model?

- ... is a simplified description of a SW process.
- ... shows through what phases SW is developed.
- SW developers select among various process models regarding the character of the particular SW system they are to develop.
- Generic process models
  - *Waterfall*;
  - *Iterative development*;
  - *Component-based software engineering*.

What are the costs of SW engineering?

- Roughly 60% are development costs, 40% are testing costs. *For custom software, evolution costs often exceed development costs*.
- Costs vary depending on the type of system being developed and the requirements of system attributes such as performance and system reliability.
- Distribution of costs depends on the development model used.

What is CASE (Computer-Aided Software Engineering)?

- Software systems that are intended to provide automated support for software process activities.
- CASE systems are often used for method support.
- *Upper-CASE*
  - Tools to support the early process activities of requirements and design;
- *Lower-CASE*
  - Tools to support later activities such as programming, debugging and testing.

What are the attributes of good software?

- SW should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable.
- *Maintainability*
  - Software must evolve to meet changing needs;
- *Dependability*
  - Software must be trustworthy;
- *Efficiency*
  - Software should not make wasteful use of system resources;
- *Acceptability*
  - Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.

What are the key challenges facing software engineering?

- *Heterogeneity*
  - Developing techniques for building software that can cope with heterogeneous platforms and execution environments;
- *Delivery*
  - Developing techniques that lead to faster delivery of software;
- *Trust*
  - Developing techniques that demonstrate that software can be trusted by its users.

Professional and ethical responsibility

- Software engineering involves wider responsibilities than simply the application of technical skills.
- Software engineers must behave in an *honest and ethically responsible* way if they are to be respected as professionals.
- ***Ethical behaviour is more than simply upholding the law***.

Issues of professional responsibility

- *Confidentiality*
  - Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- *Competence*
  - Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.

Issues of professional responsibility

- *Intellectual property rights*
  - Engineers should be aware of local laws governing the use of intellectual property such as patents, copyright, etc. They should be careful to ensure that the intellectual property of employers and clients is protected.
- *Computer misuse*
  - Software engineers should not use their technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine, say) to extremely serious (dissemination of viruses).

Ethical Dilemmas

- Disagreement in principle with the policies of senior management.
- Your employer acts in an unethical way and releases a safety-critical system without finishing the testing of the system.
- Participation in the development of military weapons systems or nuclear systems.