Chapter 1
Introduction to Software Engineering

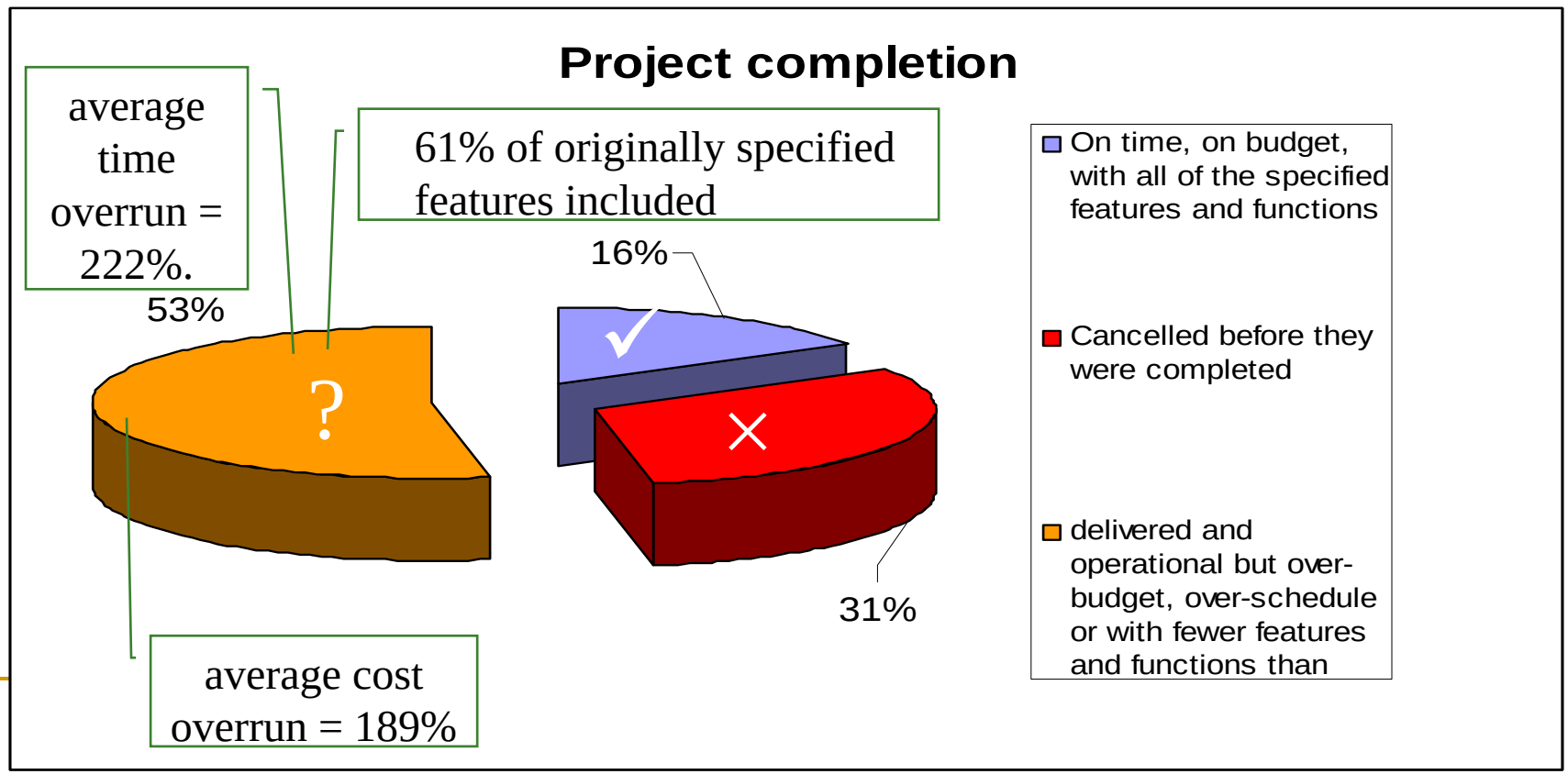
Contents

The statistics – Chaos Report

- **Standish Group – 1995**
- **365 IT executives in US companies in diverse industry segments.**
- **8,380 projects**

In Averages

- **189% of original budget**
- **221% of original schedule**
- **61% of original functionality**



Symptom of Software Crisis

- About US$250 billions spent per year in the US on application development
- Out of this, about US$140 billions wasted due to the projects getting abandoned or reworked; this in turn because of not following best practices and standards

Ref: Standish Group, 1996

Symptom of Software Crisis

- 10% of client/server apps are abandoned or restarted from scratch
- 20% of apps are significantly altered to avoid disaster
- 40% of apps are delivered significantly late

Source: 3 year study of 70 large c/s apps 30 European firms. Compuware (12/95)

Observed Problems

- Software products:
  - fail to meet user requirements
  - crash frequently
  - expensive
  - difficult to alter, debug, enhance
  - often delivered late
  - use resources non-optimally

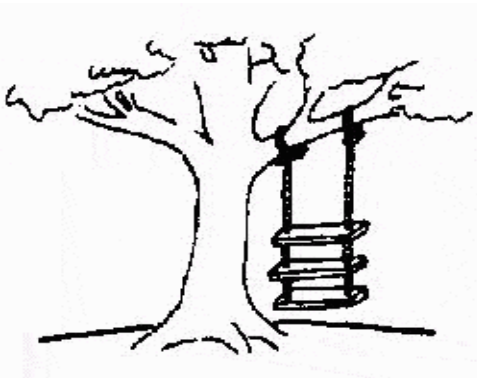
Why is the Statistics so Bad?

- Misconception on software development
  - Software myths, e.g., the man-month myth
  - False assumptions
  - Not distinguishing the coding of a computer program from the development of a software product
- Software programs have exponential growth in complexity and difficulty level with respect to size.
  - The ad hoc approach breaks down when size of software increases.

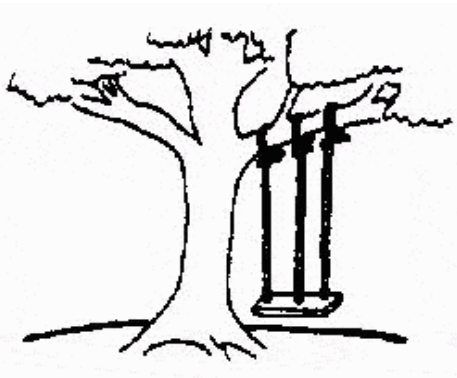
Why is the Statistics so Bad?

- Software professionals lack engineering training
  - Programmers have skills for programming but without the engineering mindset about a process discipline
- Internal complexities
  - Essences and accidents made by Fred. Brooks

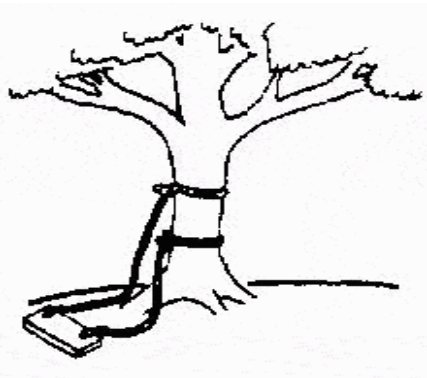
How is Software usually Constructed …



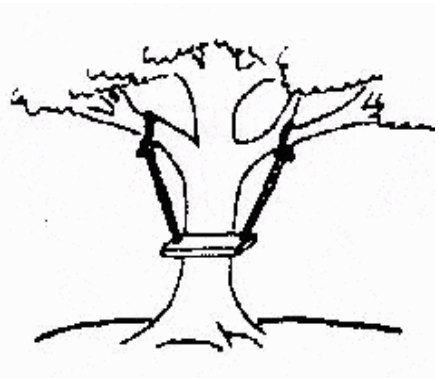
The requirements specification was defined like this



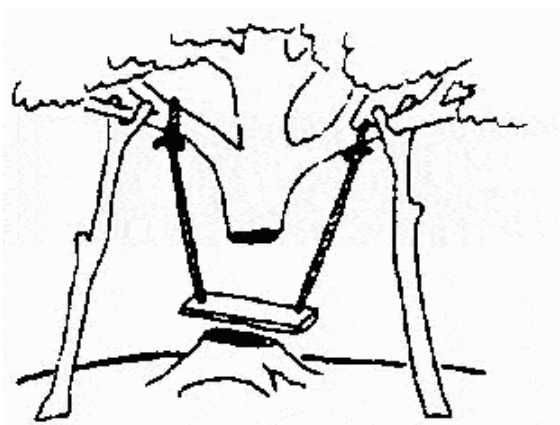
The developers understood it in that way



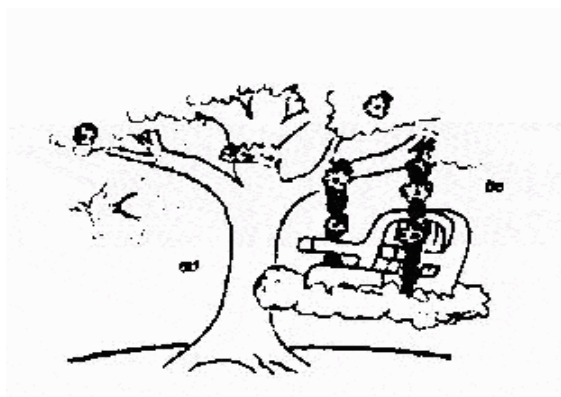
This is how the problem was solved before.



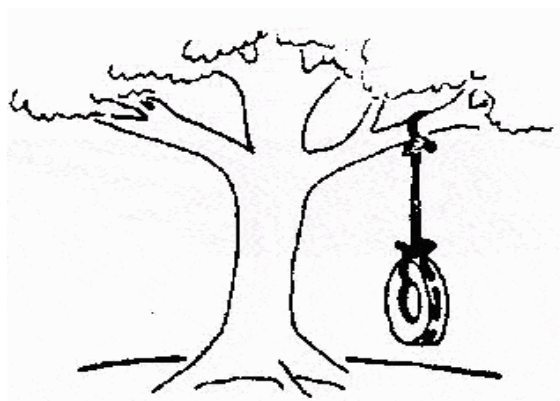
This is how the problem is solved now



That is the program after debugging



This is how the program is described by marketing dept.



This, in fact, is what the customer wanted … ;-)

Software Myths (Customer Perspectives)

- A general statement of objectives is sufficient to get started with the development of software. Missing/vague requirements can easily be incorporated/detailed out as they get concretized.
- Application requirements can never be stable; software can be and has to be made flexible enough to allow changes to be incorporated as they happen.

Software Myths (Developer Perspectives)

Once the software is demonstrated, the job is done.

Usually, the problems just begin!

Software Myths (Developer Perspectives)

Until the software is coded and is available for testing, there is no way for assessing its quality.

Usually, there are too many tiny bugs inserted at every stage that grow in size and complexity as they progress thru further stages!

Software Myths (Developer Perspectives)

The only deliverable for a software development project is the tested code.

The code is only the externally visible component of the entire software complement!

Software Myths (Management Perspectives)

As long as there are good standards and clear procedures in my company, I shouldn't be too concerned.

But the proof of the pudding is in the eating; not in the Recipe !

Software Myths (Management Perspectives)

As long as my software engineers(!) have access to the fastest and the most sophisticated computer environments and state-of-the-art software tools, I shouldn't be too concerned.

The environment is only one of the several factors that determine the quality of the end software product!

Software Myths (Management Perspectives)

When my schedule slips, what I have to do is to start a fire-fighting operation: add more software specialists, those with higher skills and longer experience - they will bring the schedule back on the rails!

Unfortunately, software business does not entertain schedule compaction beyond a limit!

Misplaced Assumptions

- All requirements can be pre-specified
- Users are experts at specification of their needs
- Users and developers are both good at visualization
- The project team is capable of unambiguous communication

Ref: Larry Vaughn

Confused with Programs and Products

Programs	Software Products
Usually small in size	Large
Author himself is sole user	Large number of users
Single developer	Team of developers
Lacks proper user interface	Well-designed interface
Lacks proper documentation	Well documented & user-manual prepared
Ad hoc development.	Systematic development

Software Programming ≠ Software Engineering

- Software programming: the process of translating a problem from its physical environment into a language that a computer can understand and obey. (*Webster's New World Dictionary of Computer Terms)*
  - Single developer
  - "Toy" applications
  - Short lifespan
  - Single or few stakeholders
    - Architect = Developer = Manager = Tester = Customer = User
  - One-of-a-kind systems
  - Built from scratch
  - Minimal maintenance

Software Programming ≠ Software Engineering

- Software engineering
  - Teams of developers with multiple roles
  - Complex systems
  - Indefinite lifespan
  - Numerous stakeholders
    - Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
  - System families
  - Reuse to amortize costs
  - Maintenance accounts for over 60% of overall development costs

What is Software?

Software is a set of items or objects that form a “configuration” that includes

- programs
- documents
- data ...

(“Software Engineering- a practitioner’s approach,” Pressman, 5ed. McGraw-Hill)

What is Software (ctd.)?

Or you may want to say:

- Software consists of
  - (1) instructions (computer programs) that when executed provided desired function and performance,
  - (2) data structures that enable the programs to adequately manipulate information, and
  - (3) documents that describe the operation and use of the programs.

What is Software (ctd.)?

But these are only the concrete part of software that may be seen, there exists also invisible part which is more important:

- Software is the dynamic behavior of programs on real computers and auxiliary equipment.
- "… a software product is a model of the real world, and the real world is constantly changing."
- Software is a digital form of knowledge. ("Software Engineering," 6ed. Sommerville, Addison-Wesley, 2000)

Unique Characteristics of Software

- Software is malleable
- Software construction is human-intensive
- Software is intangible and hard to measure
- Software problems are usually complex
- Software directly depends upon the hardware
  - It is at the top of the system engineering “food chain”
- Software doesn’t wear out but will deteriorate
- Software solutions require unusual rigor
- Software has discontinuous operational nature

Casting the Term

- The field of software engineering was born in NATO Conferences, 1968 in response to chronic failures of large software projects to meet schedule and budget constraints
- Since then, term became popular because software is getting more and more important to industry and business but the “software crisis” still persists.

What is Software Engineering?

- Different focuses for this term exist in various textbooks. Some are listed below.
- The application of a systematic, disciplined, quantifiable approach to development, operation, and maintenance of software; that is, the application of engineering to software. (*IEEE Standard Computer Dictionary*, 610.12, ISBN 1-55937-079-3, 1990)

What is Software Engineering? (ctd)

- Software engineering is concerned with the theories, methods and tools for developing, managing and evolving software products. (I. Sommerville, 6ed.)
- A discipline whose aim is the production of quality software, delivered on time, within budget, and satisfying users' needs. (Stephen R. Schach, Software Engineering, 2ed.)
- Multi-person construction of multi-version software (Parnas, 1987)

What is Software Engineering? (ctd.)

- The practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate and maintain them (B.W. Boehm)
- The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines (F.L. Bauer)

What is Software Engineering? (ctd.)

- The technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost constraints (R. Fairley)
- A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers (Ghezzi, Jazayeri, Mandrioli)

Other Definitions of Software Engineering

- “A systematic approach to the analysis, design, implementation and maintenance of software.” (*The Free On-Line Dictionary of Computing)*
- “The systematic application of tools and techniques in the development of computer-based applications.” (Sue Conger in *The New Software Engineering)*
- “Software Engineering is about designing and developing high-quality software.” (Shari Lawrence Pfleeger in *Software Engineering -- The Production of Quality Software)*

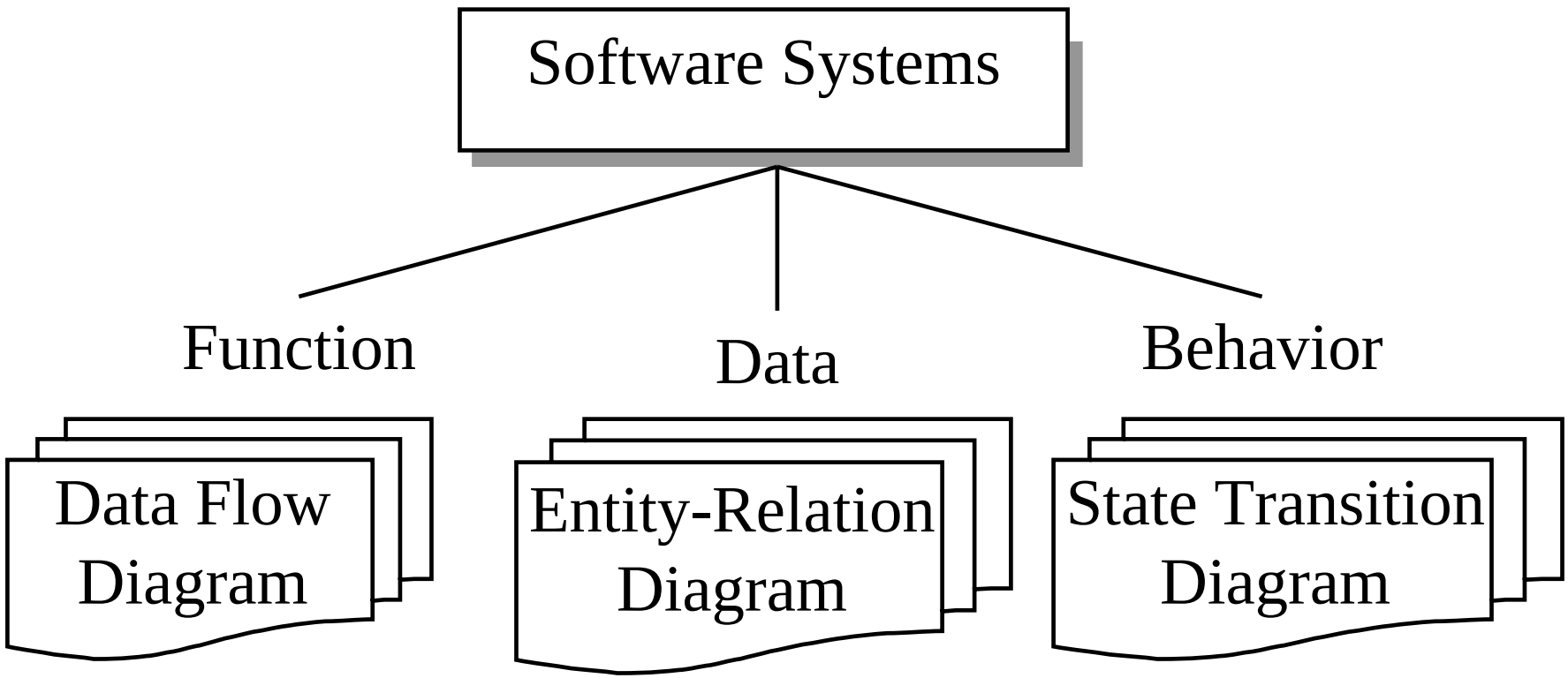
So, Software Engineering is ...

- **Scope**
  - study of software process, development principles, techniques, and notations
- **Goals**
  - production of quality software,
  - delivered on time,
  - within budget,
  - satisfying customers' requirements and users' needs

Software Process

- Waterfall life cycle
- Prototyping
- Spiral model
- Automatic synthesis model
- Object-oriented model
- 4 GL model

Traditional Software Engineering



Object-Oriented Software Engineering