Software Engineering (CSE3004)

Software Engineering Introduction

Puneet Kumar Jain

CSE Department

National Institute of Technology Rourkela

References

- Reference for the slides

© Ian Sommerville, Software Engineering, Eighth edition, Pearson education, 2007

- Pao-Ann Hsiung Dept of CSIE, National Chung Cheng Univ. Software Engineering http://www.cs.ccu.edu.tw/~pahsiung/courses/se/

Content

- Why Software Engineering?
  - Software crisis
  - Observed problems
  - Software myths
- Software Programming vs Software Engineering
- Software and software engineering (SE)
- Software process model
- Attributes of good software
- Key challenges facing software engineering
- Ethical and professional issues

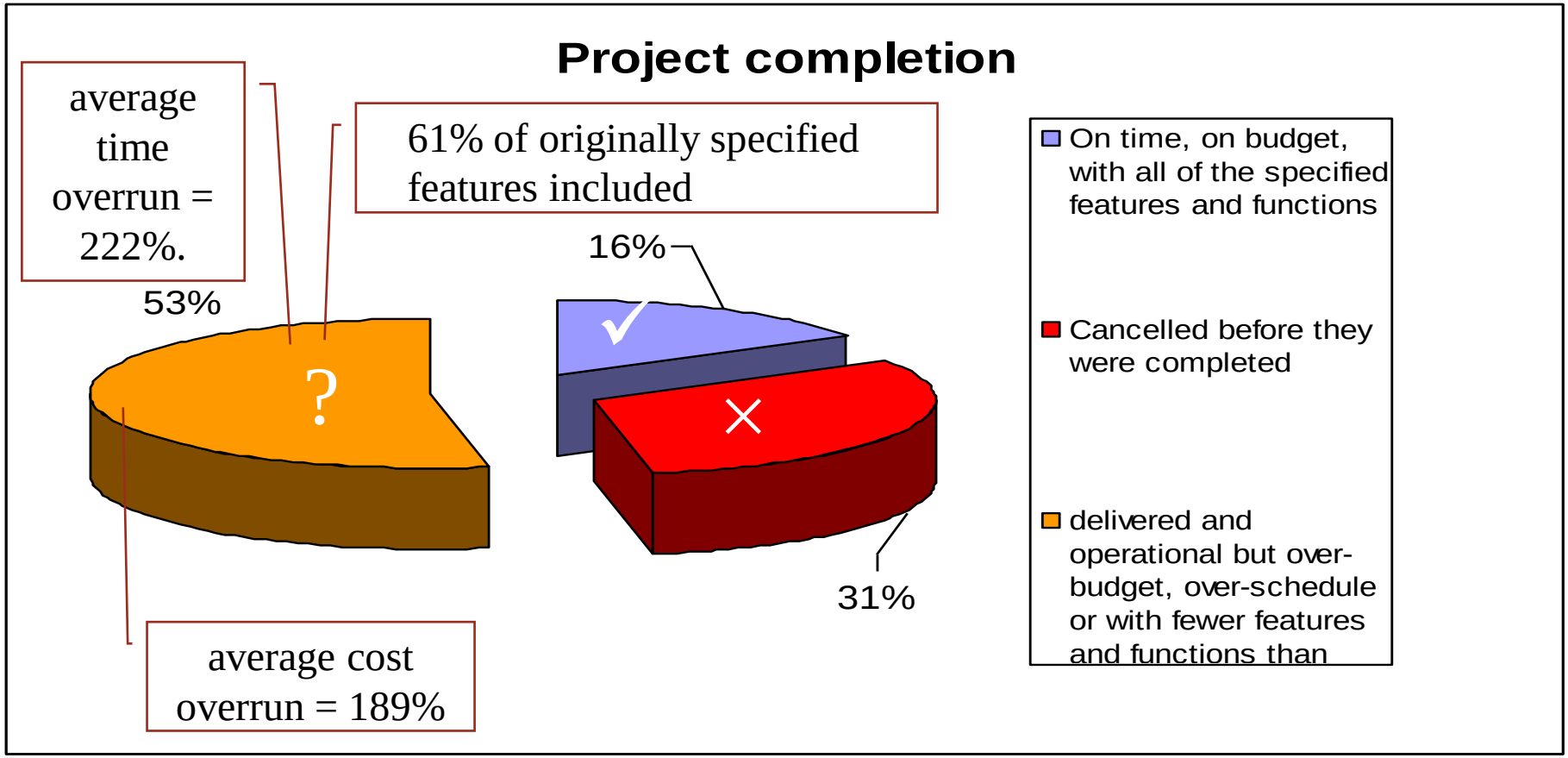
The statistics – Chaos Report

Standish Group – 1995

- 365 IT executives in US companies in diverse industry segments.
- 8,380 projects

In Averages (~53%)

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



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

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

Observed Problems

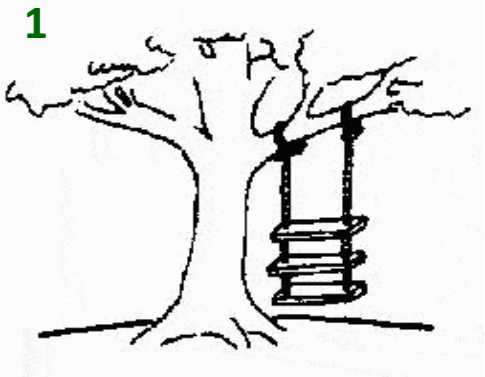
- Software products:
  - fail to meet user requirements
  - crash frequently
  - expensive
  - difficult to alter, debug, enhance
  - often delivered late
  - use resources non-optimally

Why is the Statistics so Bad?

- Misconception on software development
  - Software myths, e.g., the man-month myth ("adding manpower to a late software project makes it later". This idea is known as Brooks' law,)
  - 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.
- Software professionals lack engineering training
  - Programmers have skills for programming but without the engineering mindset about a process discipline
- Internal complexities
  - **No Silver Bullet – Essence and Accident in Software Engineering**
    (research paper by Fred. Brooks, Turing award winner)

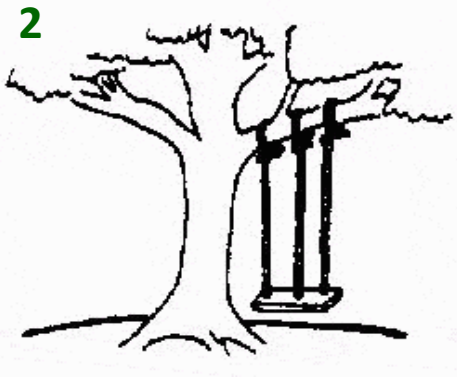
How is Software usually Constructed …

1



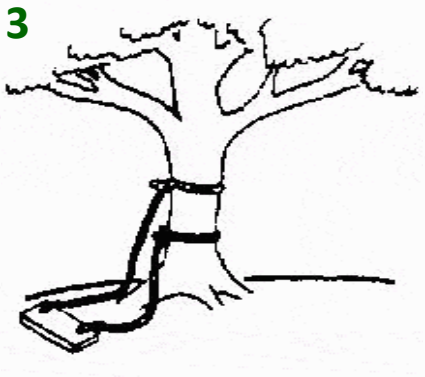
The requirements specification was defined like this

2



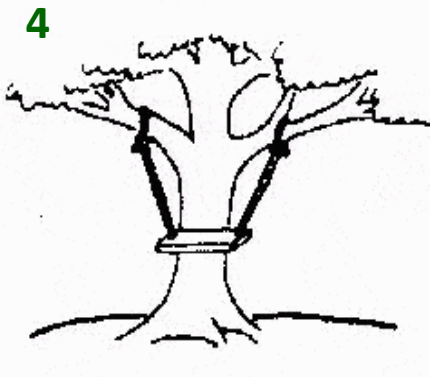
The developers understood it in that way

3



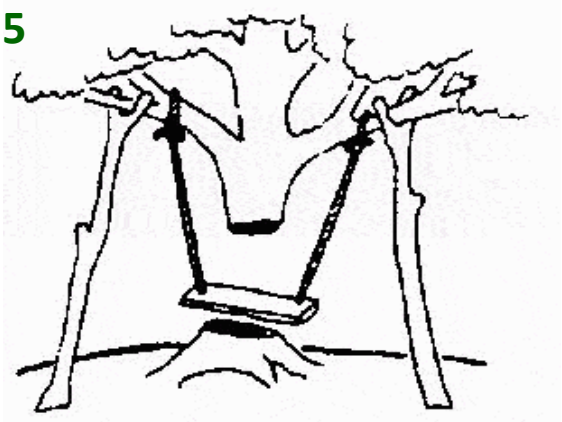
This is how the problem was solved before.

4



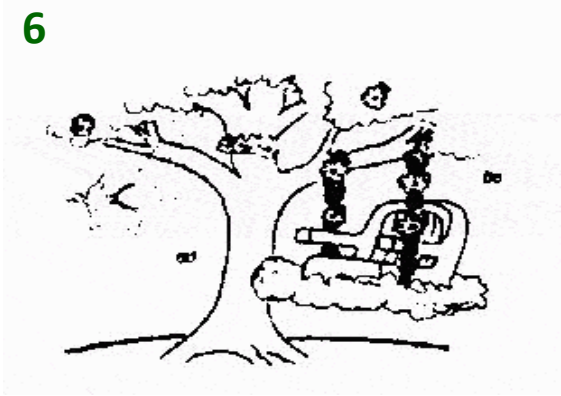
This is how the problem is solved now

5



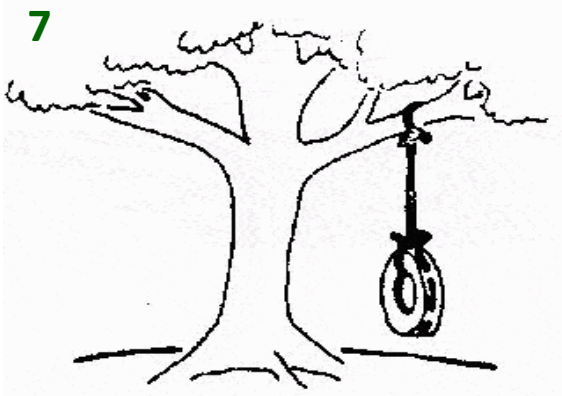
That is the program after debugging

6



This is how the program is described by marketing dept.

7



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

Software Myths (Developer 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.

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)

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 (Developer 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 (Developer 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!

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

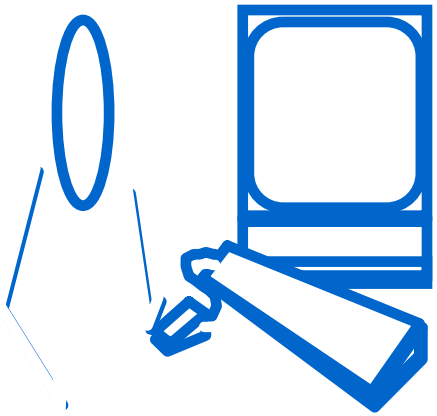
Software Programming ≠ Software Engineering

- 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)*

Software Programming	Software Engineering
Single developer	Teams of developers with multiple roles
"Toy" applications	Complex systems
Short lifespan	Indefinite lifespan
Single or few stakeholders Architect = Developer = Manager = Tester = Customer = User	Numerous stakeholders Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
One-of-a-kind systems	System families
Built from scratch	Reuse to amortize costs
Minimal maintenance	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**: instructions (computer programs) that when executed provided desired function and performance
  - **Documents**: that describe the operation and use of the programs. (such as requirements, design models and user manuals)
  - **Data**: Data and data structures that enable the programs to adequately manipulate information



Software is a digital form of knowledge. ("Software Engineering," 6ed. Sommerville, Addison-Wesley, 2000)

Ref: ("Software Engineering- a practitioner's approach," Pressman, 5ed. McGraw-Hill)

Casting the Term "Software Engineering"

- The field of software engineering was born in **NATO Conference, 1968**

- 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)

- 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.)

What is Software Engineering?

- 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)

- 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)

Other Definitions of Software Engineering

- "A systematic approach to the analysis, design, implementation and maintenance of software."

  (*The Free On-Line Dictionary of Computing)*

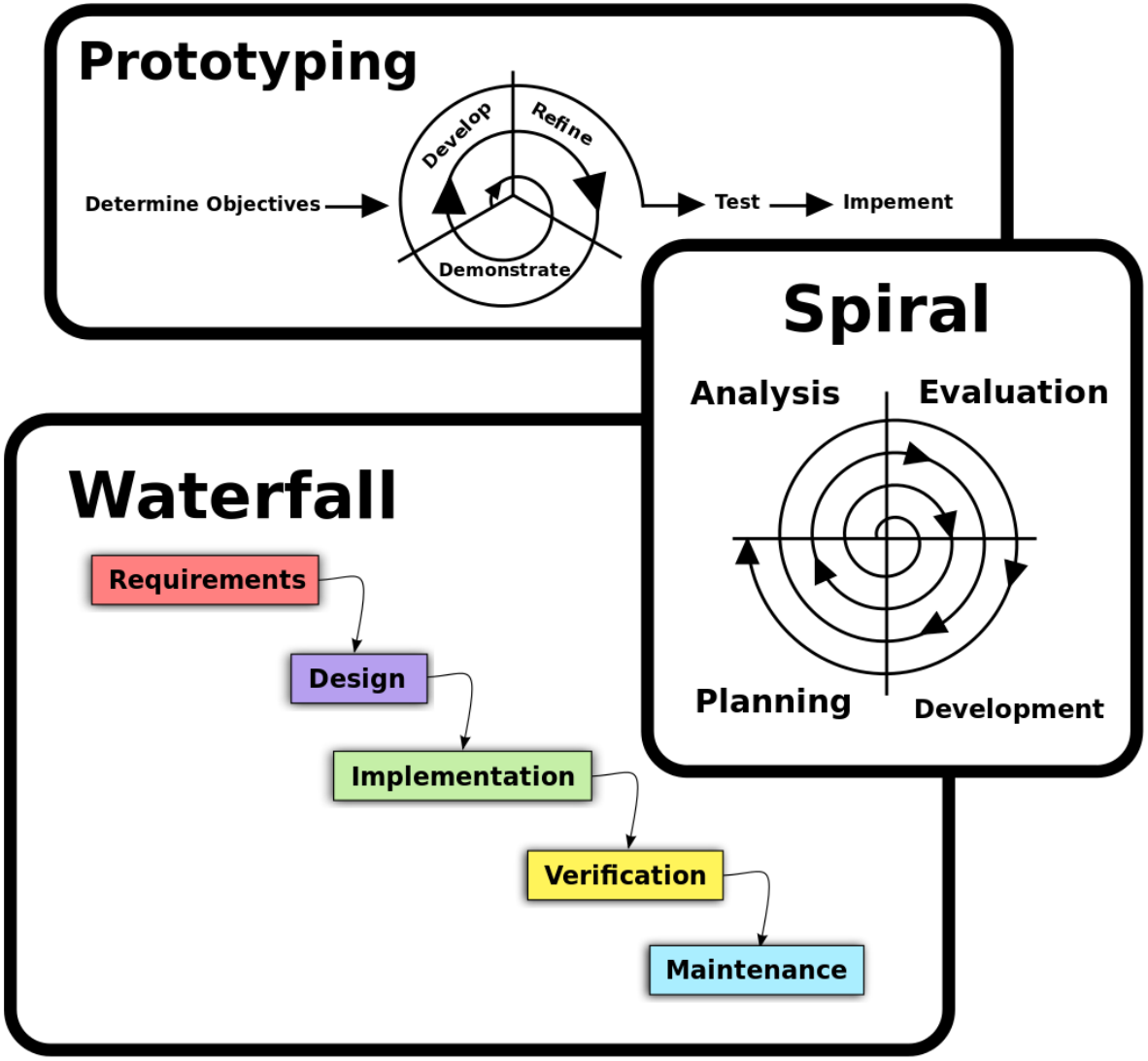
So, Software Engineering is …

- **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 Models

- A set of activities whose goal is the development or evolution of software.

- Waterfall
- V model,
- Evolutionary,
- Prototyping
- Spiral model,
- Agile models



What are the attributes of good software?

- Quality attributes expected of software
  - **Reliability**
  - **Extensibility/Maintainability**
  - **Performance**
  - **Usability**
  - **Security**
  - **Availability**
  - **Portability**

- Quality attributes expected of software engineering
  - Productivity
  - Cost

Attributes of good software

- **Reliability:**
  - Probability of failure free operation for a specified time in a specified environment for a given purpose
  - Informally, reliability is a measure of how well system users think it provides the services they require
  - One measure of Reliability is the **MTF (mean-time-between-failures)** OS/400 has a MTF of 1.3 years and MS Windows 5.0 had a MTF of 1.3 hours; incidentally, OS/400 is the ONLY software to have won the **Malcolm-Baldrige Quality Award** so far!!

Attributes of good software

- **Extensibility / Maintainability**
  - Process of modifying a software system or component after delivery
    - to correct faults,
    - to enhance functionality,
    - to improve performances or other quality attributes
    - to adapt to a changed environment, etc.

- One of the measures of Extensibility / Maintainability is the average person-days of effort to extend / maintain a software.
  - modularity quotient of a software has a direct impact on the extensibility / maintainability.

Attributes of good software

- **Usability**
  - The ease with which a user can utilize software package.
  - Usability is a culturally sensitive subject
  - A measure of usability is the average satisfaction rating of a suitably sampled user group. (Note: Attempts by industry to mine texts of blogs, news-groups to get a measure for usability of their software)

- **Security:**
  - Security of a software is the degree of its capability to protect its services and data from unauthorized users
  - Authentication (to log-in) and authorization (to use only certain services) are used to protect the services and encryption to protect the data

Attributes of good software

- **Availability**
  - Software packages contain many modules, which are integrated and work together; e.g Web servers, App servers, DBMS servers, application software modules, OS, N/W layers put together offer Infosys Finacle based banking services!
  - Availability refers to the percentage time that such as a complex software package is continuously available measured over a 100 units of time!
  - Many banks of reputation demand 99.9% availability for every 1000 days from the IT service provider!

- **Portability**
  - Portability of a software is the degree of ease with which one can install and use that software in **different environments**;

Attributes of good software engineering

- **Productivity**
  - Average number of LoC (Line of code) per person taken over the complete life-cycle of a software project
  - well engineered projects eliminate or minimize re-design or re-coding or re-testing and thereby achieve high productivity;
  - use of engineering models, methodologies, standards, tools and training of personnel etc. contribute towards high productivity;

- **Cost**
  - Appropriate balance between high productivity and high cost is very essential
  - High productivity and low cost is the dream of every software project manager!

Current State

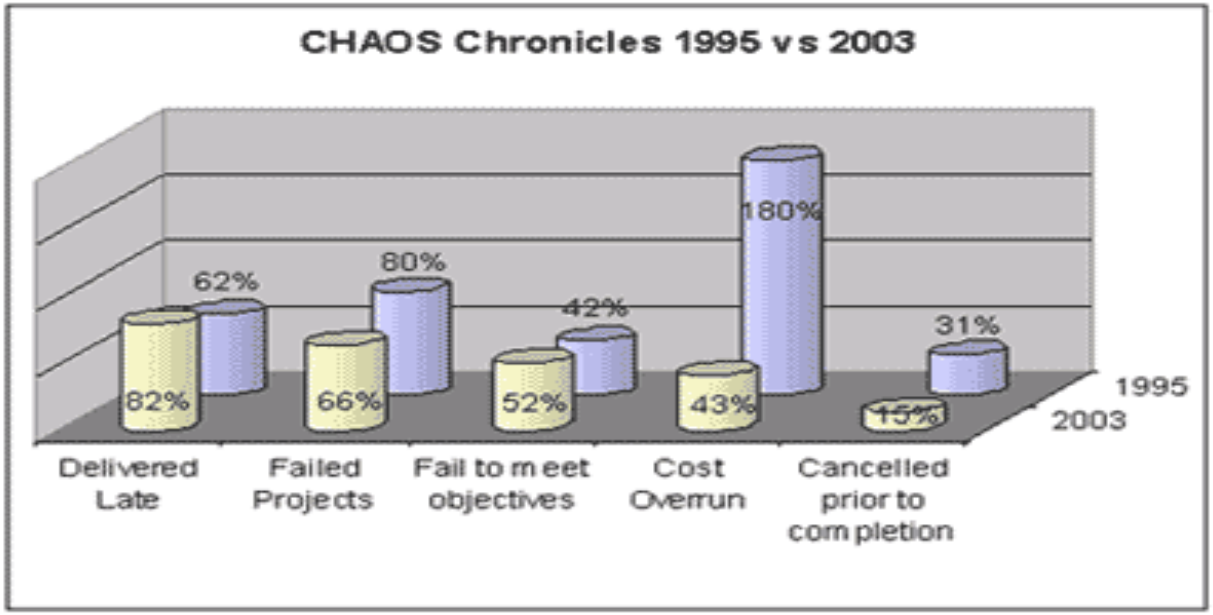
- **Software is everywhere:** There are at least 1 billion people in today's world who depend upon throughout 24 x 7 on cell phones, ATMs, Air and Train travel, Google, TV channels, Retail shops, electronic devices, etc, where a software package is continuously working!

- Some of the software packages have 30M LoC, 30K Classes, 1K modules interacting with each other using 1K database tables, 100GB data and processing 50K transactions per day with more than 99% availability!

- Still, **only one software package in the last 40 years has won the Malcolm Baldrige Quality Award** so far!

What's Wrong?

- Does software engineering have no progress at all? Not quite true.
  - We have indeed seen a lot of improvements, e.g. high level programming, object-oriented technology, etc.



The comparison with 1995's report does show that there is some progress in the past eight years.

- But it does not achieve its promise, why?

So, What's the Problem?

- **Process and quality issue:**
  - "Out of date" practices become institutionalized
  - Organizations "go with what has worked in the past"
- **Education issue:** more emphasis on methods and tools but lack of sufficient education and training on people
- Driven by intense market forces, including persistent pressure to deliver software on unrealistic delivery schedules
- Continuing rapid evolution of software methodologies and systems
- Moore's law: number of transistors on a microchip doubles every two years, though the cost of computers is halved

Software Changes in the Past Years

- Although significant improvements have been made in specific areas, the rapidly evolving nature of the software industry has resulted in little overall improvement in the overall situation.

- Changes in software over time:
  - grew in size unprecedentedly
  - operating environment changed from simple “batch” operations to complex multiprogramming systems, to time-sharing and distributed computing to today’s Internet network computing environment.

The Software Industry Today

- Component-Based Engineering and Integration
- Technological Heterogeneity
- Enterprise Heterogeneity
- Greater potential for Dynamic Evolution
- Internet-Scale Deployment
- Many competing standards
- Much conflicting terminology

Three key Challenges

Software engineering in the 21st century faces three key challenges:

- **Legacy systems**
  - Old, valuable systems must be maintained and updated
- **Heterogeneity**
  - Systems are distributed and include a mix of hardware and software
- **Delivery**
  - There is increasing pressure for faster delivery of software

- Few guiding scientific principles
- Few universally applicable methods
- As much managerial / psychological / sociological as technological

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**
  - They should not knowingly accept work which is outside their competence.
- **Intellectual property rights**
  - Engineers should be aware of local laws governing the use of intellectual property such as **patents, copyright**, etc.
- **Computer misuse**
  - Software engineers **should not use their technical skills to misuse other people's computers**.

ACM / IEEE Code of Ethics - principles

These professional societies have cooperated to produce a code of ethical practice.

1. **PUBLIC interest:** Software engineers shall act consistently with the public interest.

2. **CLIENT AND EMPLOYER:** Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. **PRODUCT :** Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. **JUDGMENT:** Software engineers shall maintain integrity and independence in their professional judgment.

5. **MANAGEMENT :** Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.

6. **PROFESSION :** Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.

7. **COLLEAGUES :** Software engineers shall be fair to and supportive of their colleagues.

8. **SELF :** Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

- **End of Chapter**