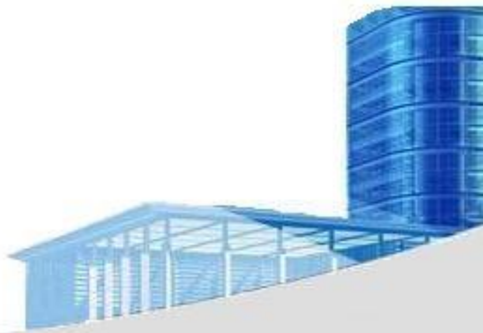
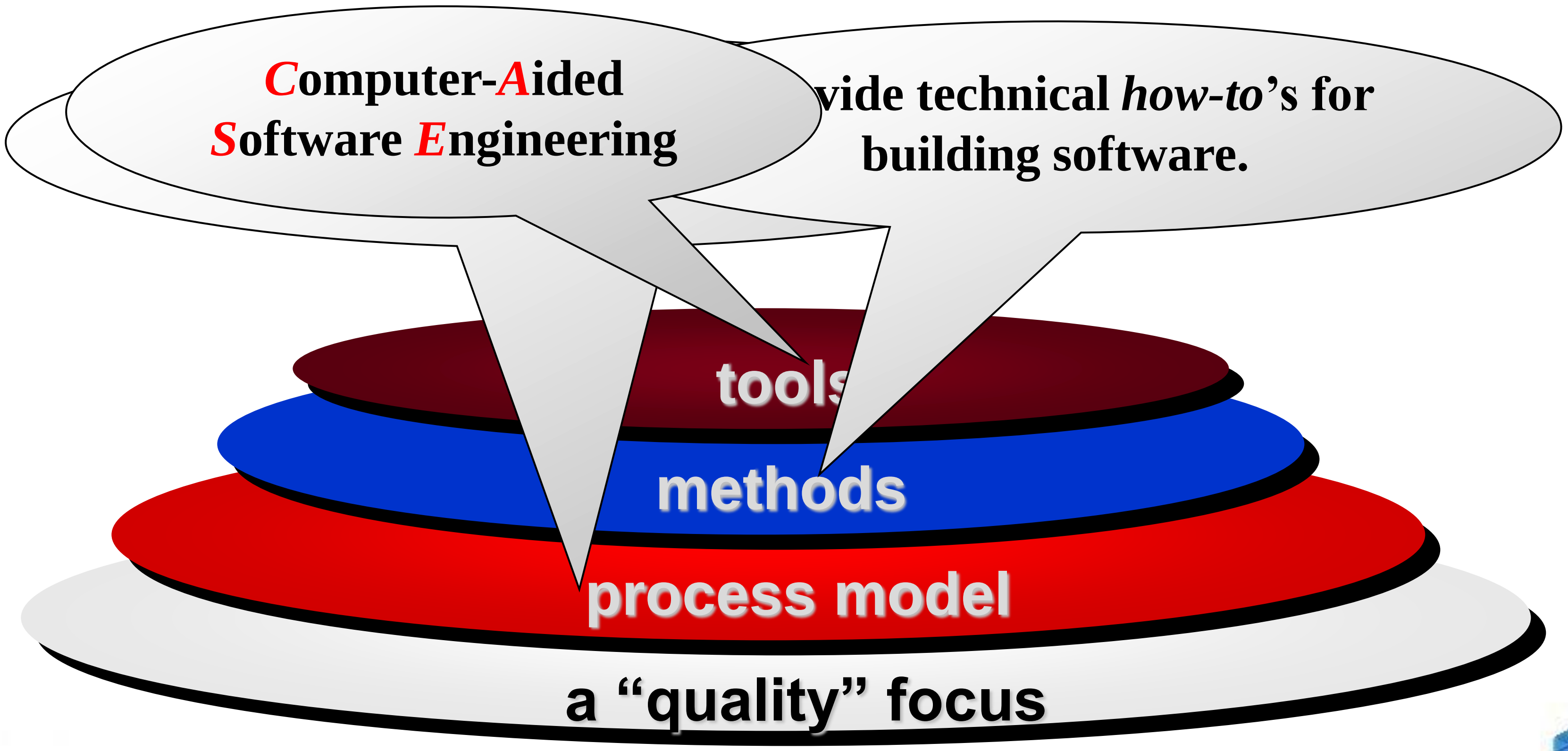
Ch.2 Software Engineering

2.1 Defining the Discipline

- **The IEEE Definition – Software Engineering**
    1. **The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.**
    2. **The study of approaches as in (1).**



2.1 Defining the Discipline


Computer-Aided Software Engineering
vide technical *how-to*'s for building software.
tools
methods
process model
a "quality" focus

A layered technology

2.2 The Software Process

Common Process Framework

Framework Activities

work tasks

work products

milestones & deliverables

QA checkpoints

Umbrella Activities

- Project management
- Quality assurance
- Work product production
- Measurement
- Formal technical reviews
- Configuration management
- Reusability management
- Risk management

2.2 The Software Process

- **Generic Process Framework**

1. **Communication** (customer collaboration and requirement gathering)
2. **Planning** (establishes engineering work plan, describes technical risks, lists resource requirements, work products produced, and defines work schedule)
3. **Modeling** (creation of models to help developers and customers understand the requires and software design)
4. **Construction** (code generation and testing)
5. **Deployment** (software delivered for customer evaluation and feedback)

2.2 The Software Process

- **Process Adaptation**
  - overall **flow** of activities, actions, and tasks and the interdependencies among them
  - degree to which **actions and tasks** are defined within each framework activity
  - degree to which **work products** are identified and required
  - manner which **quality assurance** activities are applied
  - manner in which **project tracking and control** activities are applied
  - overall degree of **detail and rigor** with which the process is described
  - degree to which the **customer and other stakeholders** are involved with the project
  - level of autonomy given to the **software team**
  - degree to which **team organization and roles** are prescribed

2.3 Software Engineering Practice

- **The Essence of Practice**

  1. **Understand the problem** (communication and analysis).
  2. **Plan a solution** (modeling and software design).
  3. **Carry out the plan** (code generation).
  4. **Examine the result for accuracy** (testing and quality assurance).

2.3 Software Engineering Practice

- **General Principles**

1. The reason it all exists — Provide **Value** to users
2. **KISS** — Keep It Simple, Stupid!
3. Maintain the **Vision**
4. What you produce, others will consume
5. Be open to the future
6. Plan ahead for reuse
7. Think!

2.4 Software Development Myths

- **Management myths**

Myth: We already have a book that's full of standards and procedures for building software. Won't that provide my people with everything they need to know?

Reality: Does everybody care?

Myth: If we get behind schedule, we can add more programmers and catch up.

1 + 1 << 2

Reality: Software development is not a mechanistic process like manufacturing. In the words of Brooks, "adding people to a late software project makes it later."

Myth: If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality: If you cannot manage your own people well, you will invariably struggle when you outsource.

2.4 Software Development Myths

- **Customer myths**

Myth: A general statement of objectives is sufficient to begin writing programs – we can fill in the details later.

Case 2. In the late 1960s, a bright-eyed young engineer* was chosen to "write" a computer program for an automated manufacturing application. The reason for his selection was simple. He was the only person in his technical group who had attended a computer programming seminar. He knew the in's and out's of assembler language and Fortran, but nothing about software engineering and even less about project scheduling and tracking.

His boss gave him the appropriate manuals and a verbal description of what had to be done. He was informed that the project must be completed in two months.

He read the manuals, considered his approach, and began writing code. After two weeks, the boss called him into his office and asked how things were going.

"Really great," said the young engineer with youthful enthusiasm, "This was much simpler than I thought. I'm probably close to 75 percent finished."

The boss smiled. "That's really terrific," he said. He then told the young engineer to keep up the good work and plan to meet again in a week's time.

2.4 Software Development Myths

Case 2 (cont.)

A week later the boss called the engineer into his office and asked, “Where are we?”

“Everything’s going well,” said the youngster, “but I’ve run into a few small snags. I’ll get them ironed out and be back on track soon.”

“How does the deadline look?” the boss asked.

“No problem,” said the engineer. “I’m close to 90 percent complete.”

If you’ve been working in the software world for more than a few years, you can finish the story. It’ll come as no surprise that the young engineer stayed 90 percent complete for the entire project duration and only finished (with the help of others) one month late.

2.4 Software Development Myths

Case 3. In the early 1980s, the United States' Internal Revenue Service (IRS) hired Sperry Corporation to build an automated federal income tax form processing system. According to the *Washington Post*, the "system has proved **inadequate to the workload, cost nearly twice what was expected and must be replaced soon**" (Sawyer 1985). In 1985, an extra **$90 million** was needed to enhance the original **$103 million** worth of Sperry equipment. In addition, because the problem prevented the IRS from returning refunds to taxpayers by the deadline, the IRS was forced to pay **$40.2 million** in interest and **$22.3 million** in overtime wages for its employees who were trying to catch up.

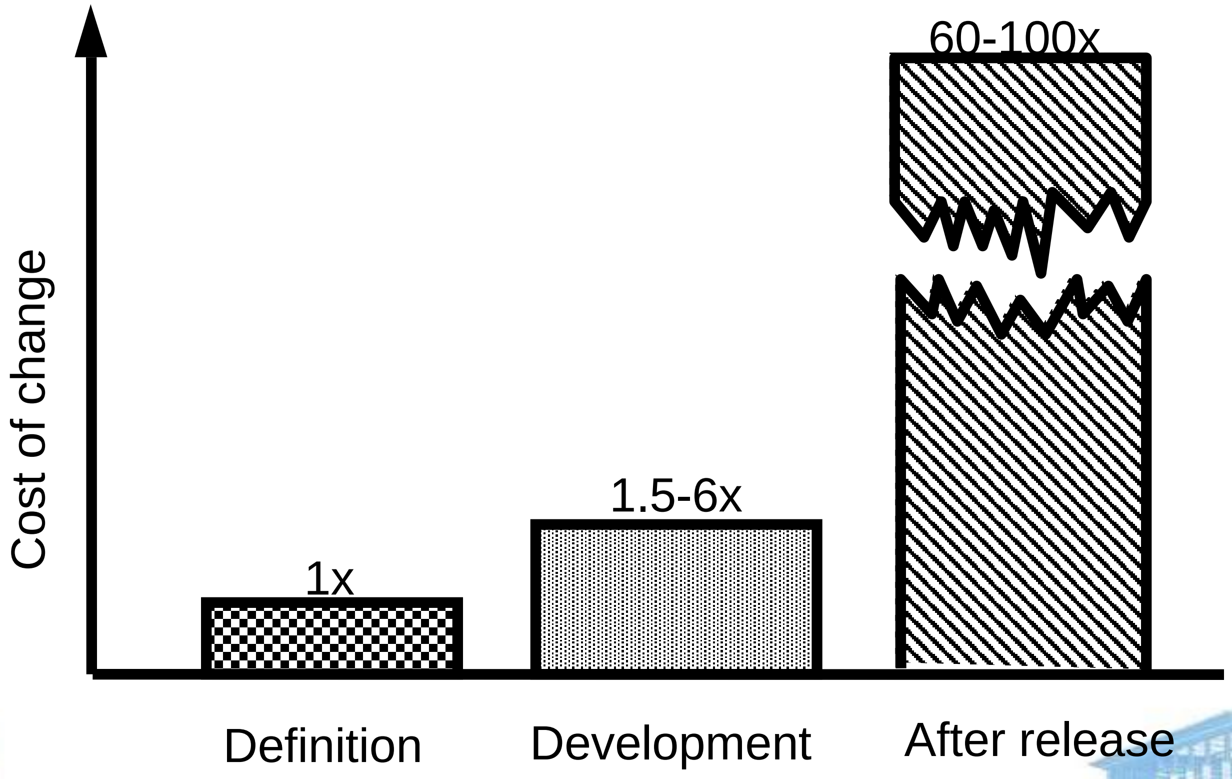
In 1996, the situation had not improved. The *Los Angeles Times* reported on March 29 that there was still no master plan for the modernization of IRS computers, only a six-thousand-page technical document. Congressman Jim Lightfoot called the project "a **$4-billion** fiasco that is floundering because of **inadequate planning**" (Vartabedian 1996).

2.4 Software Development Myths

- **Customer myths**

Myth: Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: The impact of change is shown by the figure.



2.4 Software Development Myths

- **Practitioner's myths**

Myth: Once we write the program and get it to work, our job is done.

Case 4.　某公园有一游船码头，负责人希望开发一游船管理系统，要求如下：当游客租船时，管理员输入S表示租船周期开始；当游客还船时，管理员输入E表示租船周期结束。一天结束时，要求系统打印出租船次数和平均租船时间。

Reality: Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done."

Industry data indicate that between 60 and 80 percent of all effort expended on a program will be expended after it is delivered to the customer for the first time.

新要求：输出一天中的最长租用时间。

新要求：将报告分上午和下午输出。

新要求：当通信线路出问题时，能从计算中删除一切不完整的租船信息。

2.4 Software Development Myths

- **Practitioner's myths**

Myth … ssing its qu…

Rea…

Managers : evaluate, track progress, ...

Programmers : communicate to each other

Maintainers : *VITAL!*

Myth: The only deliv… …ccessful project is the working program.

Reality: A working program is only one part of a so…ware configuration that includes programs, documents, and data. **Documentation** forms the foundation for successful development and, more important, provides guidance for software support.

Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

Reality: Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.