A、B战队分组名单（目前共37人）

组号	职责	姓名	学号	
A-1	组长	黄琲	3210104881	
	组员	展翼飞	3190102196	
	组员	何永瑞	3230400061	
	组员	赵元康	3210106046	
	组员			
组号	职责	姓名	学号	
A-2	组长	薛杰怀	3210100662	A大组长
	组员	王一哲	3210102169	
	组员	张匡令	3210104612	
	组员	胡家齐	3210104424	
	组员	陈艺真	3210300493	
组号	职责	姓名	学号	
A-3	组长	谢瑞航	3210106035	
	组员	李心羽	3210104749	
	组员	文博韬	3210102562	
	组员	项峥	3210102501	
	组员	胡炘炎	3210102517	
组号	职责	姓名	学号	
A-4	组长	林方芊	3210100527	
	组员	黄静彪	3200105271	
	组员	李杭奇	3210104821	
	组员	刘志化	3230400064	
	组员			

组号	职责	姓名	学号	
B-1	组长	董冬	3210104573	
	组员	张汉宸	3210106029	
	组员	栗威	3210106175	
	组员	陈书陶	3210105352	B大组长
	组员	郑维康	3210102381	
组号	职责	姓名	学号	
B-2	组长	唐朝	3210102187	
	组员	赵子炎	3210105581	
	组员	孟澍	3210101819	
	组员	陈苇远	3210105677	
	组员	钱闻博	3210100736	
组号	职责	姓名	学号	
B-3	组长	吴迪	3210105557	
	组员	陈科睿	3210104320	
	组员	卢峰杰	3210102198	
	组员	郑浩博	3210105321	
	组员			
组号	职责	姓名	学号	
B-4	组长	刘佳星	3210106007	
	组员	李力扬	3210105647	
	组员	王程业	3210101733	
	组员	俞心宇	3210104724	
	组员	潘臻琦	3210102495	

Ch.1 Introduction to Software Engineering(Cont.)

March 4, 2024

1.5 Software 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（讨论会）. He knew the in’s and out’s of assembler language（汇编语言） 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.

Case 2 (cont.)

"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.

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 （踢出去）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.

1.5 Software Myths

- Case 3. The Story of Kai-fu Lee (李开复)

In July, 1981, 20-year-old Kai-fu Lee studied at Columbia University and was very good at programming at that time. The Law School Dean wanted to rewrite Course Selection System software from expensive IBM hosts with Cobol to cheap DECVAX transplanted computer.

---Wages: 7 USD/Hour

--- Kai-fu Lee gladly accepted and promised that the task could be completed in early August.

---However……

1.5 Software Myths

Case 4. 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(退款) 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.

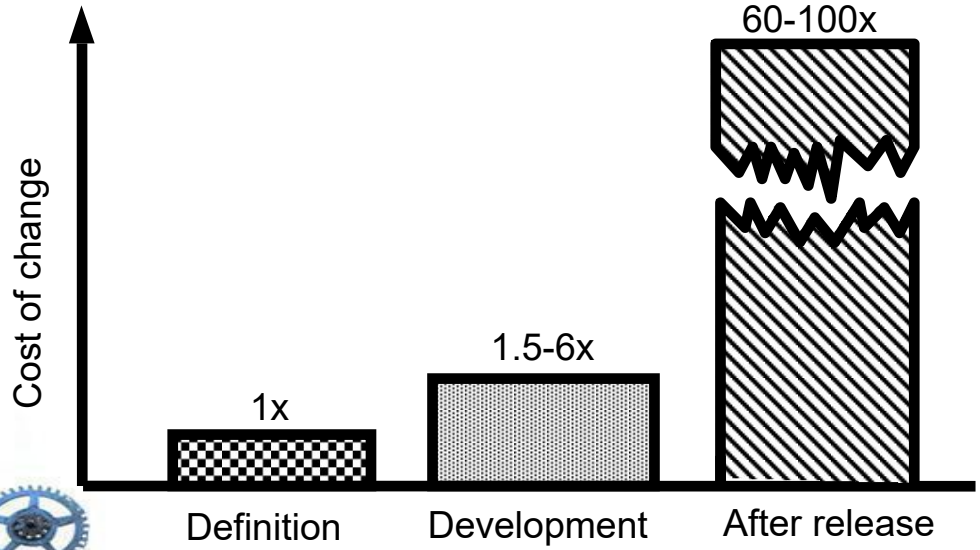
In 1996, the situation had not improved. The *Los Angeles Times* reported on March 29 that there was still no master plan for the modernization of IRS computers, only a six-thousand-page technical document. Congressman Jim Lightfoot called the project "a **$4-billion** fiasco（彻底失败） that is floundering(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

1.5 Software 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.**



1.5 Software Myths

- **Practitioner's myths**

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

Case 5. 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。

Algorithm:

```
Number = Total_time  = 0;
Get Message;
While ( ! End_of_stream ) {
   if (Code == S) {
      Number ++;
      Total_time  –= Start_time;
   else    Total_time  += End_time;
   Get Message;
}
Print Number;
If (Number)    Print Total_time / Number;
```

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。

1.5 Software Myths

- **Practitioner's myths**

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

Case 5. 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。

1.5 Software Myths

- **Practitioner's myths**

Myth: Until I get the program running, I have no way of assessing its quality.
Reality: Formal technical review is a kind of quality filter (See Ch 26).

Myth: The only deliverable work product for a successful project is the working program.
Reality: A working program is only one part of a **software 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.

1.5 Software Myths

- **Practitioner's myths**

Myth: Until I get the program ru

Reality: Formal technical review

Managers : evaluate, track progress, ...

Programmers : communicate to each other

Maintainers : *VITAL!*

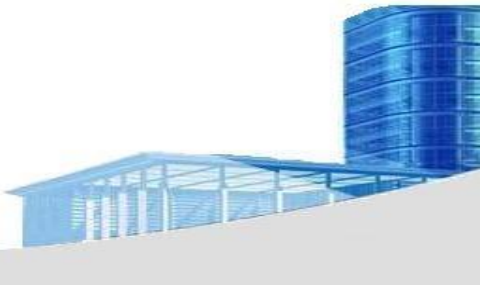
Myth: The only deliverable work product for a ul project is the working program.

Reality: A working program is only one pa a **software 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.

Ch.2 Software Engineering



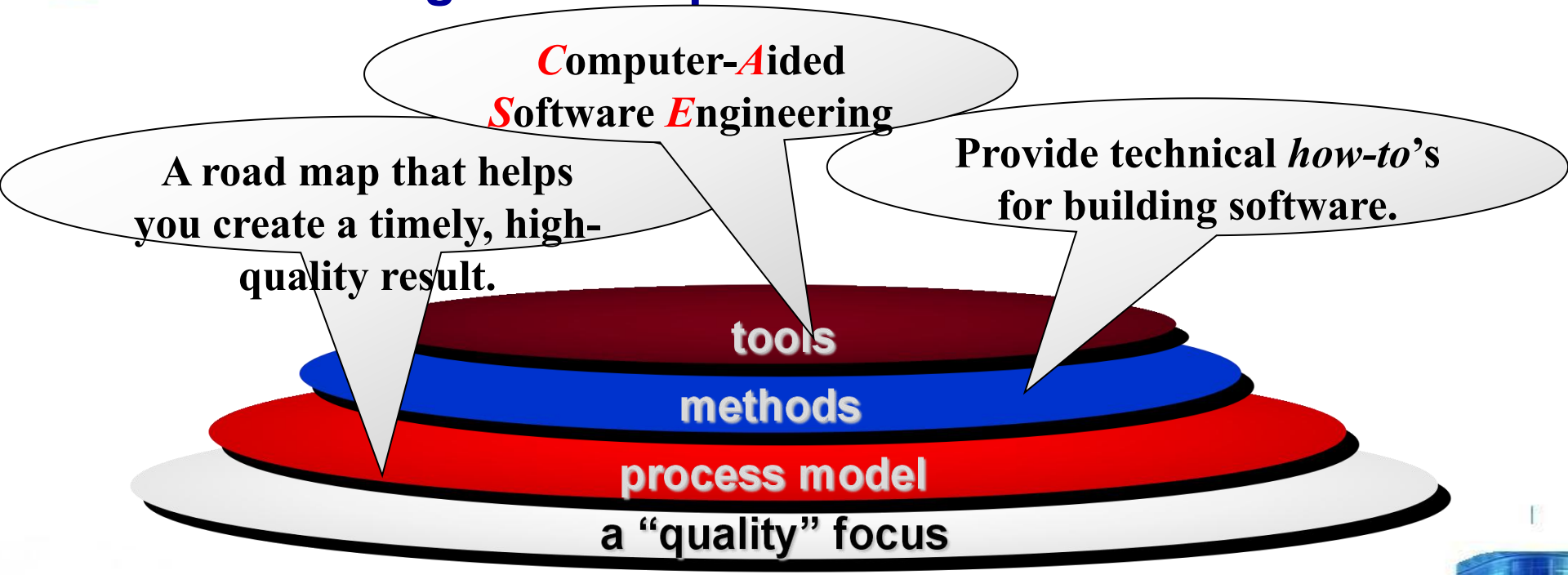
2.1 Defining the Discipline（学科）

- The IEEE Definition – Software Engineering

1. The application of a systematic, disciplined（受过训练的）, 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).
   Ex. Waterfall(瀑布模型) or Agile（敏捷开发模型）?

2.1 Defining the Discipline



A layered technology

2.2 The Software Process

Common Process Framework

Framework Activities

- work tasks
- work products
- milestones & deliverables
- QA checkpoints

Umbrella (普适的)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 Ex. STSS (智能教学服务系统)**

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(活动), actions(行动),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**（刚度） with which the process is described
  - – degree to which the **customer and other stakeholders (共同利益者)** are involved with the project
  - – level of autonomy(自治性) given to the **software team** degree to which **team organization and roles** are prescribed

2.3 Software Engineering Practice

- **The Essence of Practice**

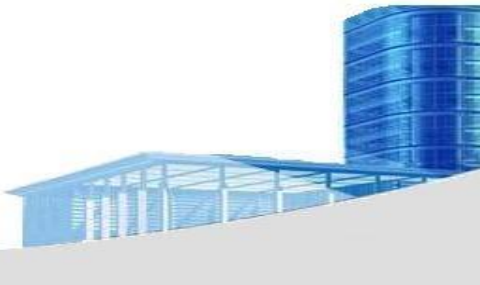
1. **Understand the problem** (communication and analysis).
2. **Plan a solution** (modeling and software design).
3. **Carry out the plan** (code generation).
4. **Examine the result for accuracy** (testing and quality assurance).

2.3 Software Engineering Practice

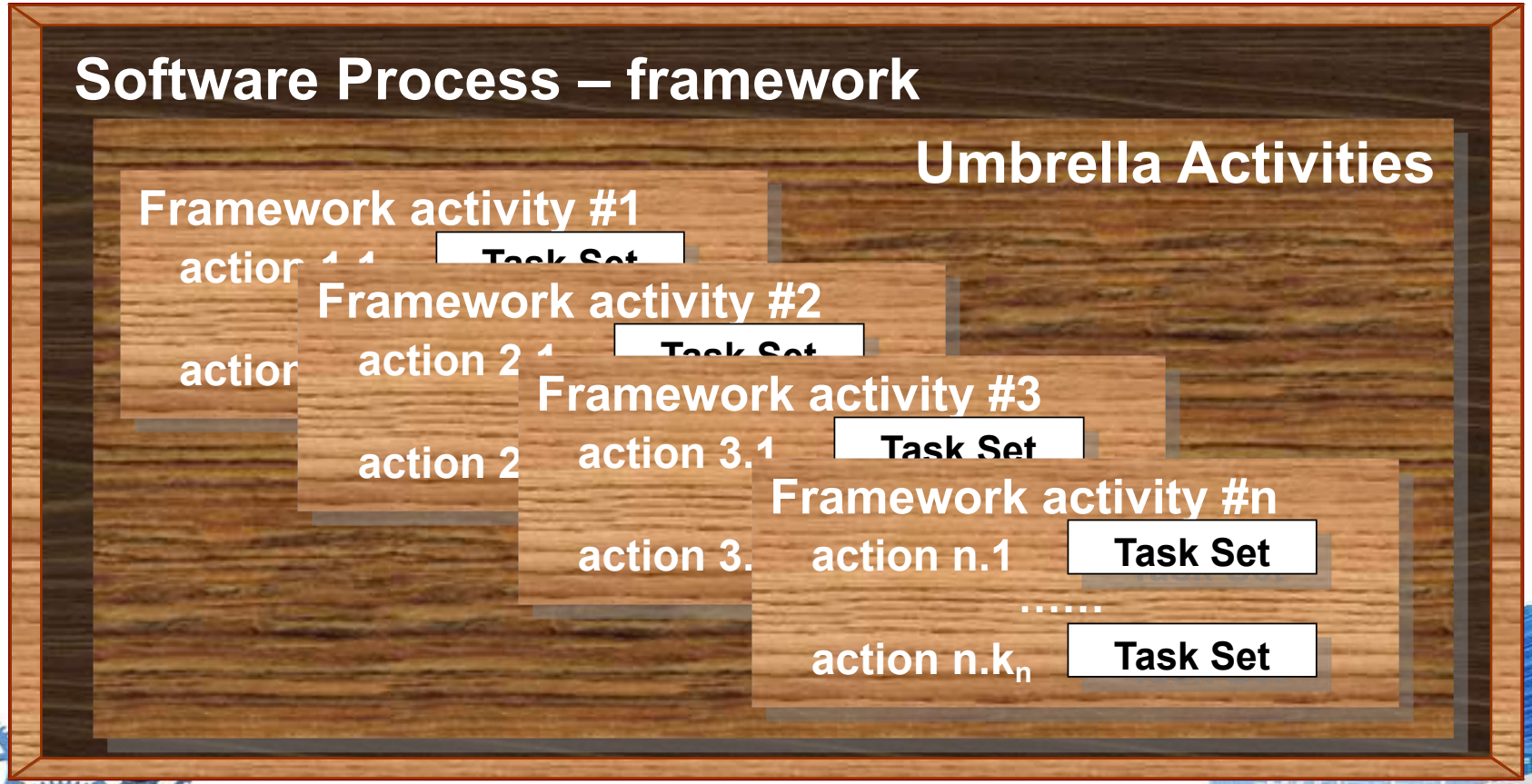
- **General Principles**

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

Ch.3 Software Process Structure

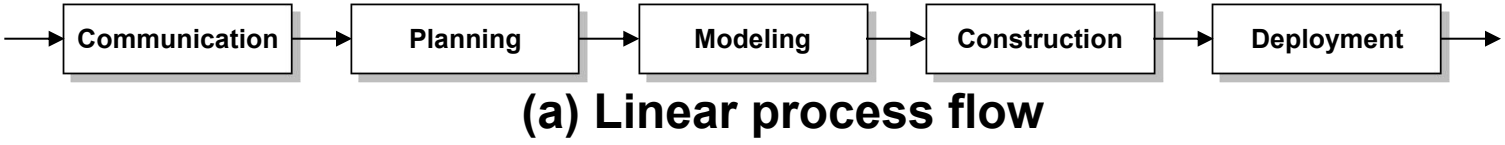


3.1 A Generic Process Model

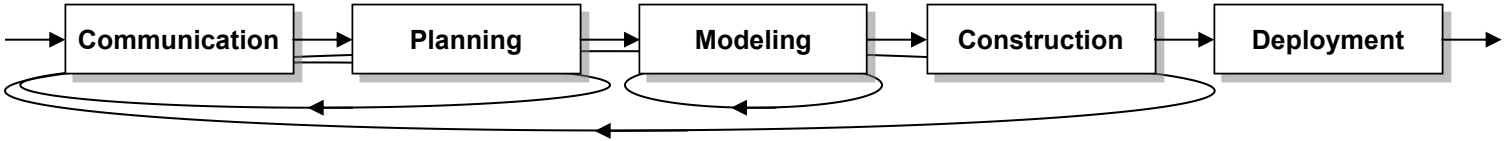


3.1 A Generic Process Model

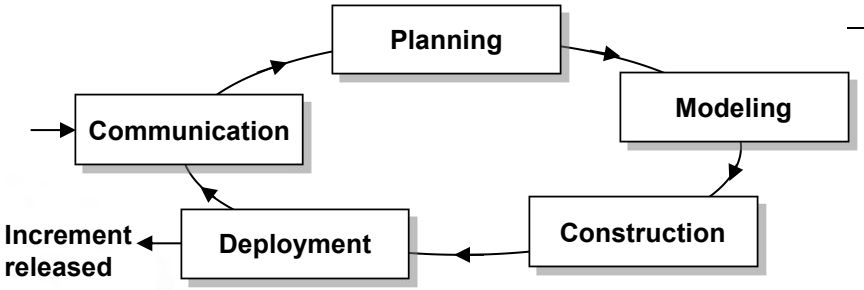
- **Process flow**



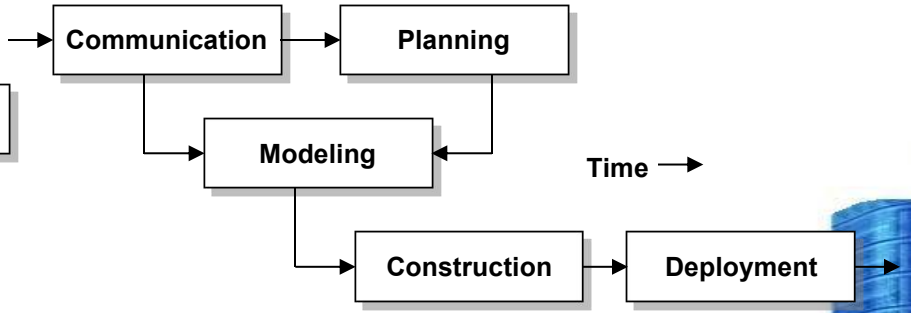
(a) Linear process flow



(b) Iterative process flow



(c) Evolutionary process flow



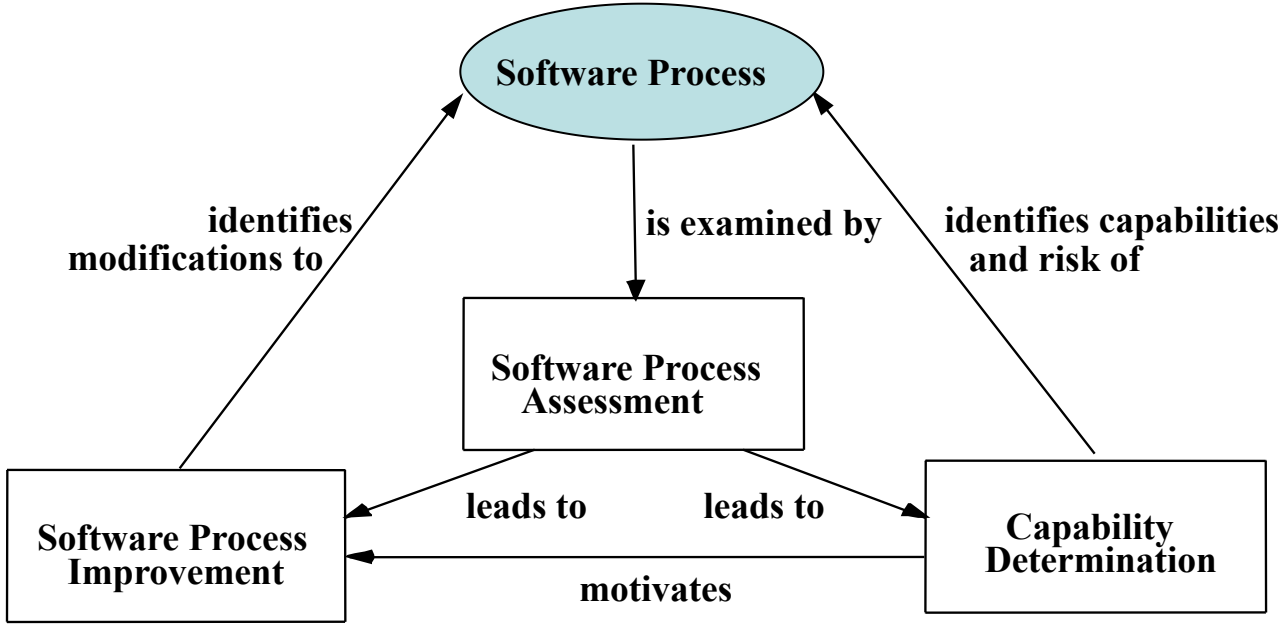
(d) Parallel process flow

3.4 Process Patterns

- **Process patterns** define a set of activities, actions, work tasks, work products and/or related behaviors

- A **template** is used to define a pattern
- **Generic software pattern elements**
  - Meaningful **pattern name** (e.g. Customer Communication)
  - **Intent** (objective of pattern)
  - **Type** 步骤 阶段
    - **Task pattern** (defines engineering action or work task, e.g. Requirement Gathering)
    - **Stage pattern** (defines **framework** activity for the process, e.g. Communication)
    - **Phase pattern** (defines sequence or flow of framework activities that occur within process, e.g. Spiral model or Prototyping<--原型)

3.5 Process Assessment



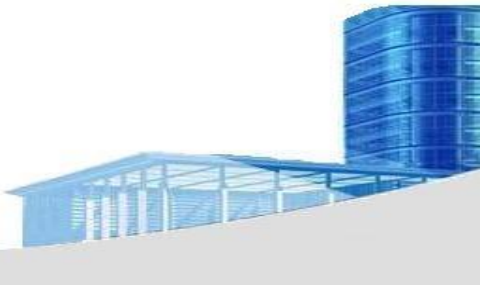
- SCAMPI
- SPICE (ISO/IEC15504)
- CBA IPI
- ISO 9001:2000 for Software

The Capability Maturity Model Integration

— by Software Engineering Institute (SEI) of Carnegie Mellon University (CMU)

- **Level 0: Incomplete** (process is not performed or does not achieve all goals defined for this level)
- **Level 1: Performed** (work tasks required to produce required work products are being conducted)
- **Level 2: Managed** (people doing work have access to adequate resources to get job done, stakeholders are actively involved, work tasks and products are monitored, reviewed, and evaluated for conformance to process description)
- **Level 3: Defined** (management and engineering processes documented, standardized, and integrated into organization-wide software process)
- **Level 4: Quantitatively Managed** (software process and products are quantitatively understood and controlled using detailed measures)
- **Level 5: Optimizing** (continuous process improvement is enabled by quantitative feedback from the process and testing innovative ideas)

Ch.4 Process Models



4.1 Prescriptive Models

- **Prescriptive （惯例） process models advocate an orderly approach to software engineering, e.g. Windows 2025**

- **Questions:**

1. If prescriptive process models strive （兴盛 / 持续） for structure and order, are they inappropriate for a software world that thrives on change?
2. Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured（e.g. Agile，敏捷开发）, do we make it impossible to achieve coordination （协调） and coherence （一致） in software work?

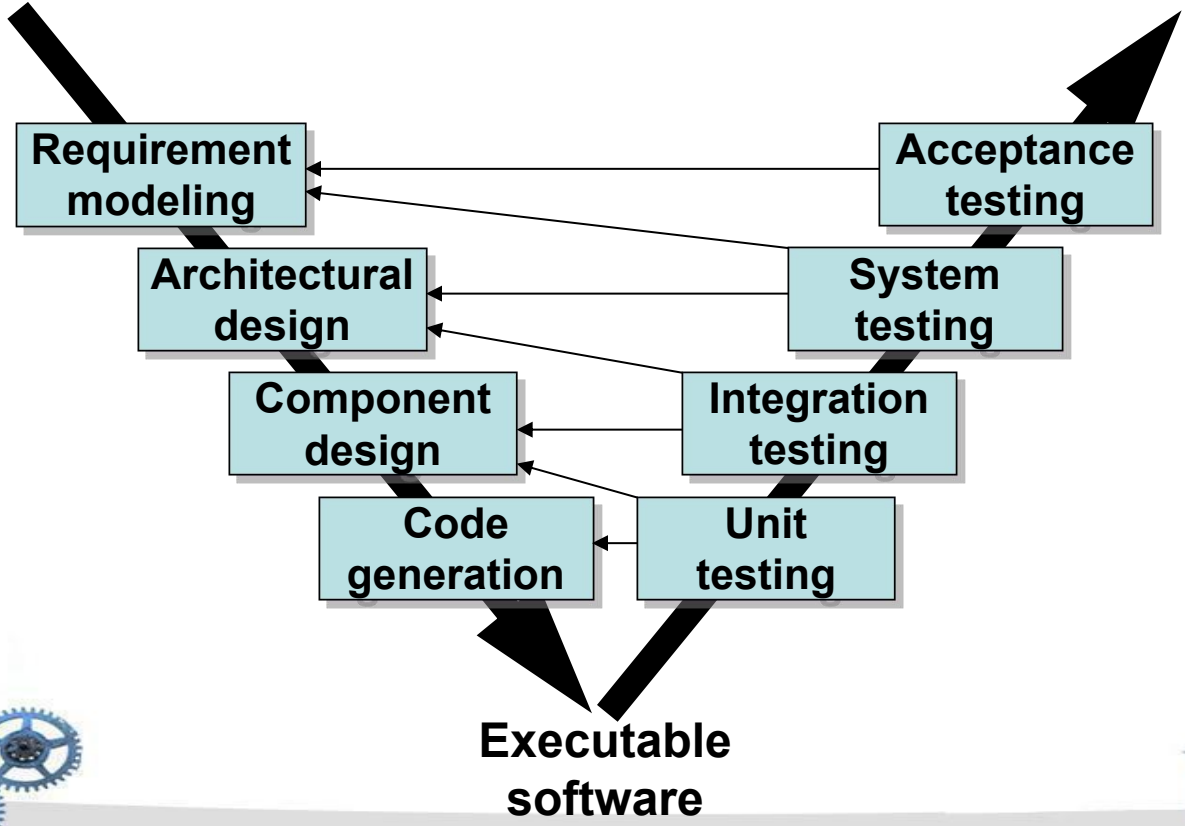
4.1.1 The Waterfall Model

- Real projects rarely follow the sequential flow.
- Customers usually can't state all requirements explicitly.
- A working version will not be available until late in the project time-span.

Classic Life Cycle

4.1.1 The Waterfall Model

- The V-model



Task

- **Review** Ch. 2-4
- **Finish** “Problems and points to ponder” in Ch. 2-4
- **Hold team / group meeting!**
- **Preview** Ch. 31,5,6,7