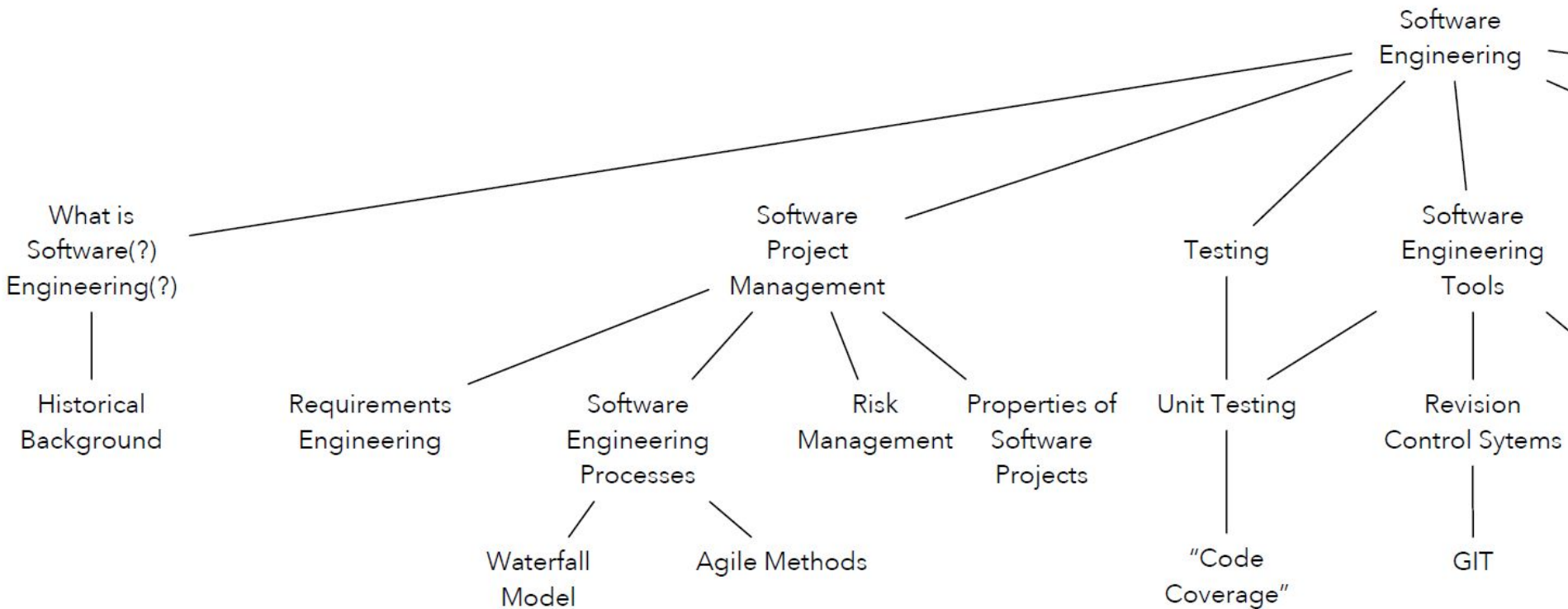
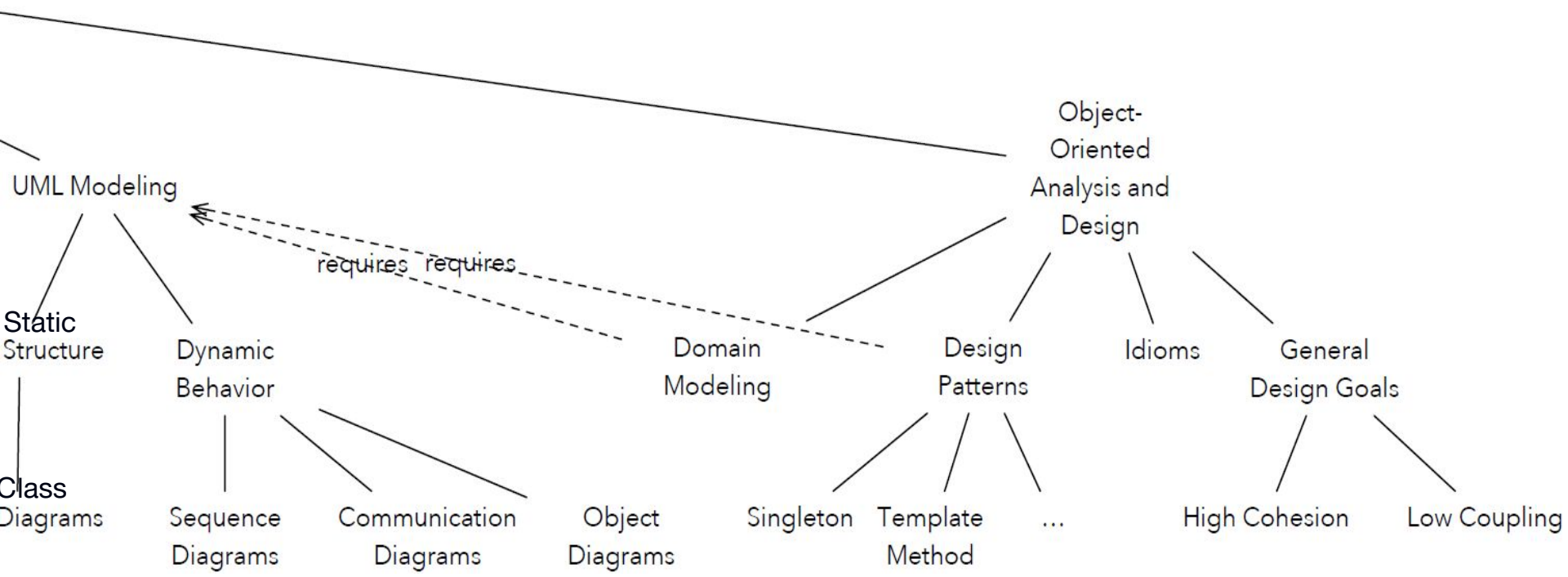


Overview on Software Engineering Topics





Ch 1: The Nature of Software

Ch 2: Software Engineering

Moonzoo Kim
CS Dept. KAIST

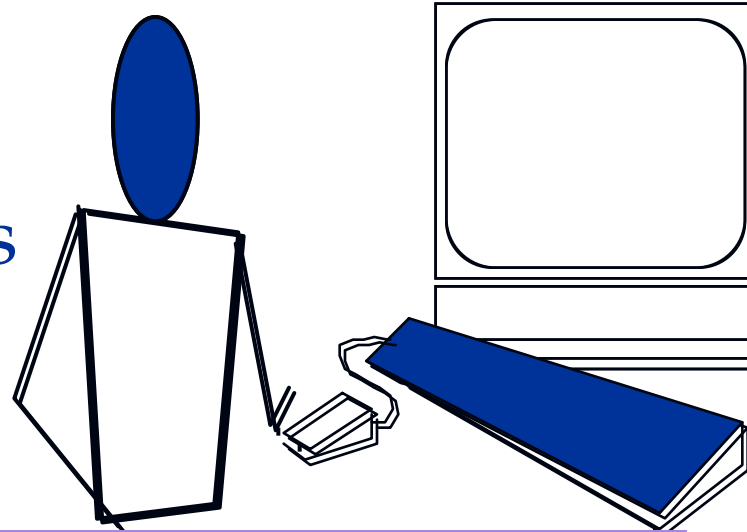
Software's Dual Role

- Software is a product
 - Delivers computing potential
 - Produces, manages, acquires, modifies, displays, or transmits **information**
- Software is a **vehicle** for delivering a product
 - Supports or directly provides system functionality
 - Controls other programs (e.g., an operating system)
 - Effects communications (e.g., networking software)
 - Helps build other software (e.g., software tools)
- Even Software can enable the creation of new technologies and new markets
 - E.g. genetic engineering and nano technology
 - E.g. Fintech (e.g., p2p banking, Bitcoin, etc.)

What is Software?

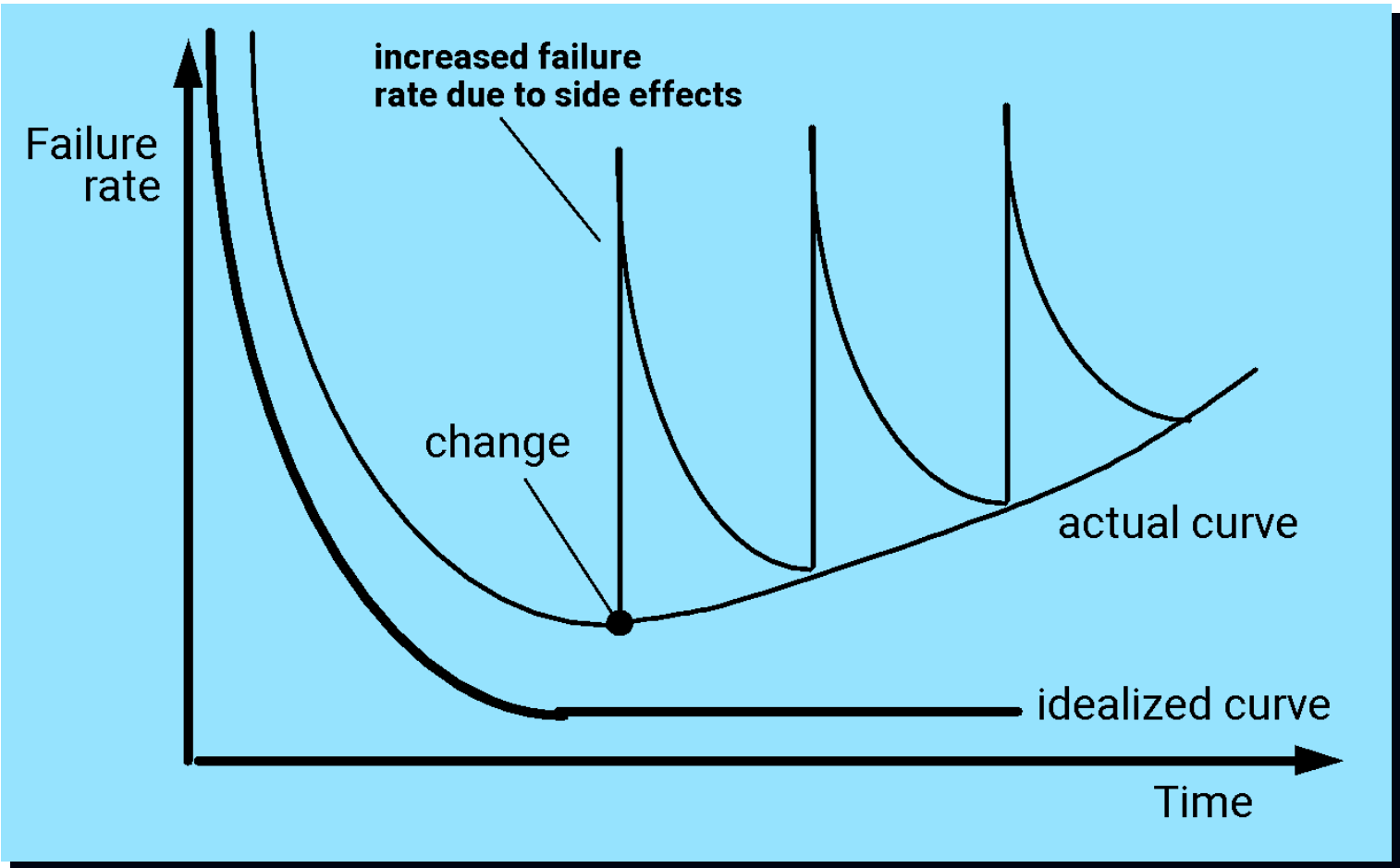
Software is a **set** of items or objects that form a “configuration” that includes

- programs
- documents
- data ...



- software is engineered
- software doesn't wear out
- software is complex

Wear vs. Deterioration



Legacy Software

Why must it change?

- software must be **adapted** to meet the needs of new computing environments or technology.
- software must be **enhanced** to implement new business requirements.
- software must be **extended to make it interoperable** with other more modern systems or databases.

The Laws of SW Evolution (Ch. 36) (1/2)

- The Law of **Continuing Change** (1974):
 - E-type systems must be continually adapted
 - They become progressively less satisfactory otherwise
- The Law of **Increasing Complexity** (1974):
 - As an E-type system evolves, its complexity increases unless work is done to maintain or reduce it (refactoring)
- The Law of **Conservation of Familiarity** (1980):
 - As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behavior to achieve satisfactory evolution.
 - Therefore, the average incremental growth remains invariant as the system evolves

The Laws of SW Evolution (Ch. 36) (2/2)

- The Law of **Continuing Growth** (1980):
 - The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.
- The Law of **Declining Quality** (1996):
 - The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.

Management Myths (1/2)

- **Myth:** We already have standards and procedures for building software, won't that provide my people with everything they need to know?

- **Reality:** The book of standards may very well exist, but is it used? In many cases, the answer to the following questions is "no."

- Are software practitioners aware of its existence?
- Does it reflect modern software engineering practice?
- Is it complete?
- Is it streamlined to improve time to delivery while still maintaining a **focus on quality**?

Management Myths (2/2)

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

- **Reality:** Software development is **not** a mechanistic process like manufacturing. In the words of Brooks [BRO75]: "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.

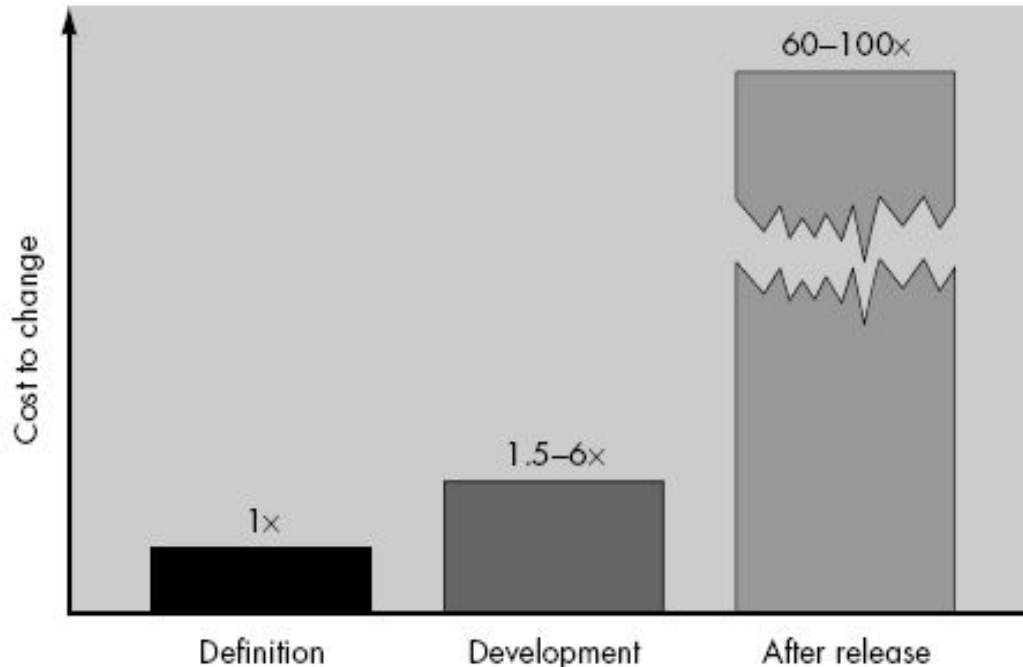
- **Reality:** If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer Myths (1/2)

- **Myth:** A general statement of objectives is sufficient to begin writing programs—we can fill in the details later.
 - **Reality:** A poor up-front definition is the major cause of failed software efforts. **A formal and detailed description** of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough **communication** between customer and developer.

Customer Myths (2/2)

- **Myth:** Project requirements continually change, but change can be easily accommodated because software is flexible.
 - **Reality:** It is true that software requirements change, but the impact of change varies with the time at which it is introduced.



Practitioner's Myths (1/2)

- **Myth:** Once we write the program and get it to work, our job is done.
 - **Reality:** Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done." Industry data ([LIE80], [JON91], [PUT97]) indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.
- **Myth:** Until I get the program "running" I have no way of assessing its quality.
 - **Reality:** One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the *formal technical review*. Software reviews are more effective than testing for finding certain classes of software defects.

Practitioner's Myths (2/2)

- **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 many elements. **Documentation** provides a foundation for successful engineering and, more important, 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.

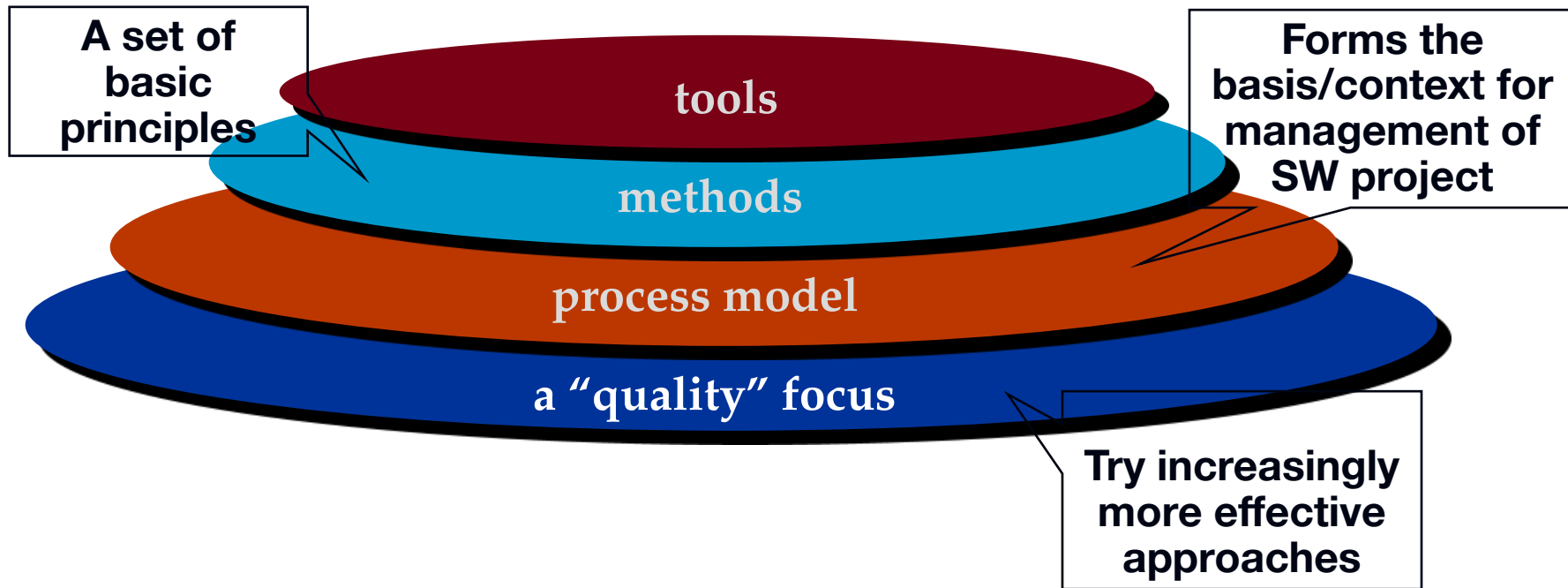
Why Is Software Process Important?

- Software process v.s. food recipe
- A **process** is a collection of *activities*, *actions*, and *tasks* to perform when some work product is to be created.
- Process helps us **order** our thinking by defining **common activities** and **artifacts**
 - Process is a means to **capture** and **transfer** the **knowledge** we gain in developing a particular product
 - Process improvement identify and deploy knowledge **over large groups**.

Why Process Improvement Helps

- A process is about incorporating **discipline** into **routine activities** to check everything that was supposed to be done was done
 - Making sure
 - There was sufficient **repeatability** in the tasks to make future work **predictable**
 - This process repeatability and predictability are called “**capability maturity**”
- Informally speaking, process improvement is to incorporate **individual wisdom/guidance** into the way the organization works

Software Engineering Layers



A SW Process Framework

Process framework

Framework activities

work tasks

work products

milestones & deliverables

QA checkpoints

Umbrella Activities

Software process

Process framework

Umbrella activities

framework activity # 1

software engineering action #1.1

Task sets

work tasks
work products
quality assurance points
project milestones

⋮

software engineering action #1.k

Task sets

work tasks
work products
quality assurance points
project milestones

⋮

framework activity # n

software engineering action #n.1

Task sets

work tasks
work products
quality assurance points
project milestones

⋮

software engineering action #n.m

Task sets

work tasks
work products
quality assurance points
project milestones

5 Framework Activities

- Communication
- Planning
 - Technical tasks
 - The risks
 - The resources
 - Work products
 - Work schedule
- Modeling
 - To better understand the requirements and the design
- Construction
 - Code generation
 - Testing
- Deployment

Umbrella Activities

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Software configuration management
- Reusability management
- Work product preparation and production