# Ch 1: Software and Software Engineering

Moonzoo Kim
SoC. KAIST

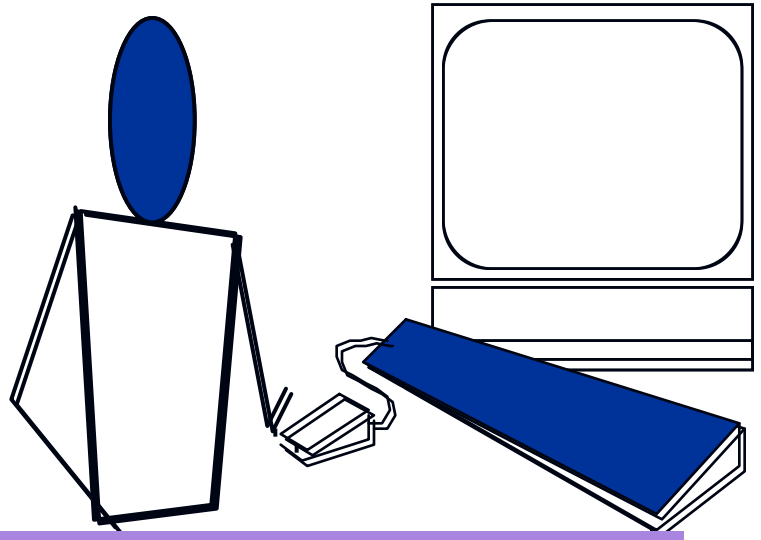
# Software's Dual Role

- 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
  - E.g. genetic engineering and nano technology

# What is Software?

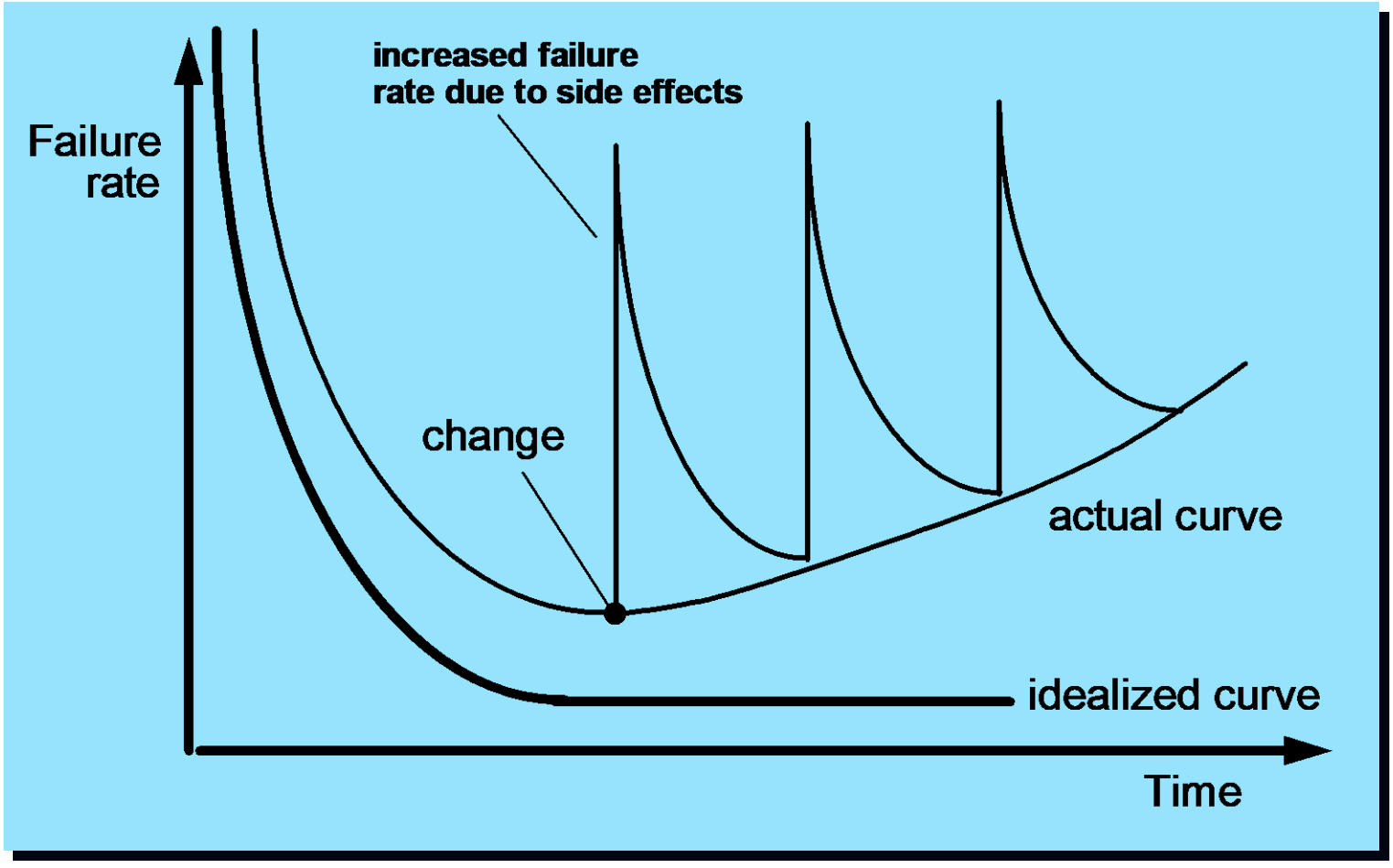
Software is a set of items or objects that form a “configuration” that includes

- 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

# The Laws of SW Evolution (Ch. 27) (1/2)

- The Law of Continuing Change:
  - E-type systems must be continually adapted
  - They become progressively less satisfactory otherwise
- The Law of Continuing Growth:
  - The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.
- The Law of Increasing Complexity:
  - As an E-type system evolves, its complexity increases unless work is done to maintain or reduce it (refactoring)

KAIST

**Source: Lehman, M., et al, "Metrics and Laws of Software Evolution—The Nineties View," *Proceedings of the 4th International Software Metrics Symposium (METRICS '97),* IEEE, 1997, can be downloaded from: http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf**

# The Laws of SW Evolution (Ch. 27) (2/2)

- The Law of Conservation of Familiarity:
  - 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.
    - Thus, the average incremental growth remains invariant as the system evolves

- The Law of Declining Quality:
  - The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.



Source: Lehman, M., et al, "Metrics and Laws of Software Evolution—The Nineties View," *Proceedings of the 4th International Software Metrics Symposium (METRICS '97),* IEEE, 1997, can be downloaded from: http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf

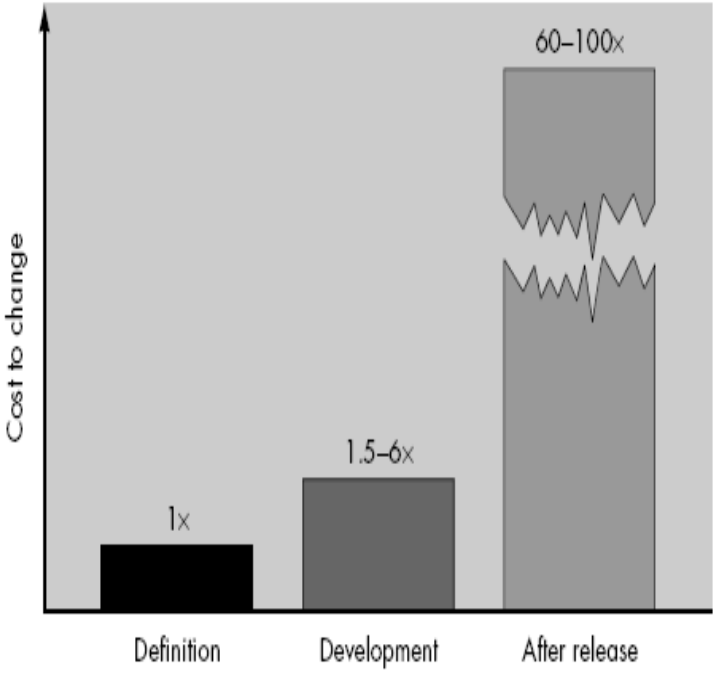
# Management Myths

- ***Myth1:*** 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?
- ***Myth2:*** 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“
- ***Myth3:*** 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.

KAIST

# Customer Myths

- ***Myth4:*** 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.
- ***Myth5:*** 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

- ***Myth6:*** 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.
- ***Myth7:*** 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.
- ***Myth8:*** 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.
- ***Myth9:*** 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.

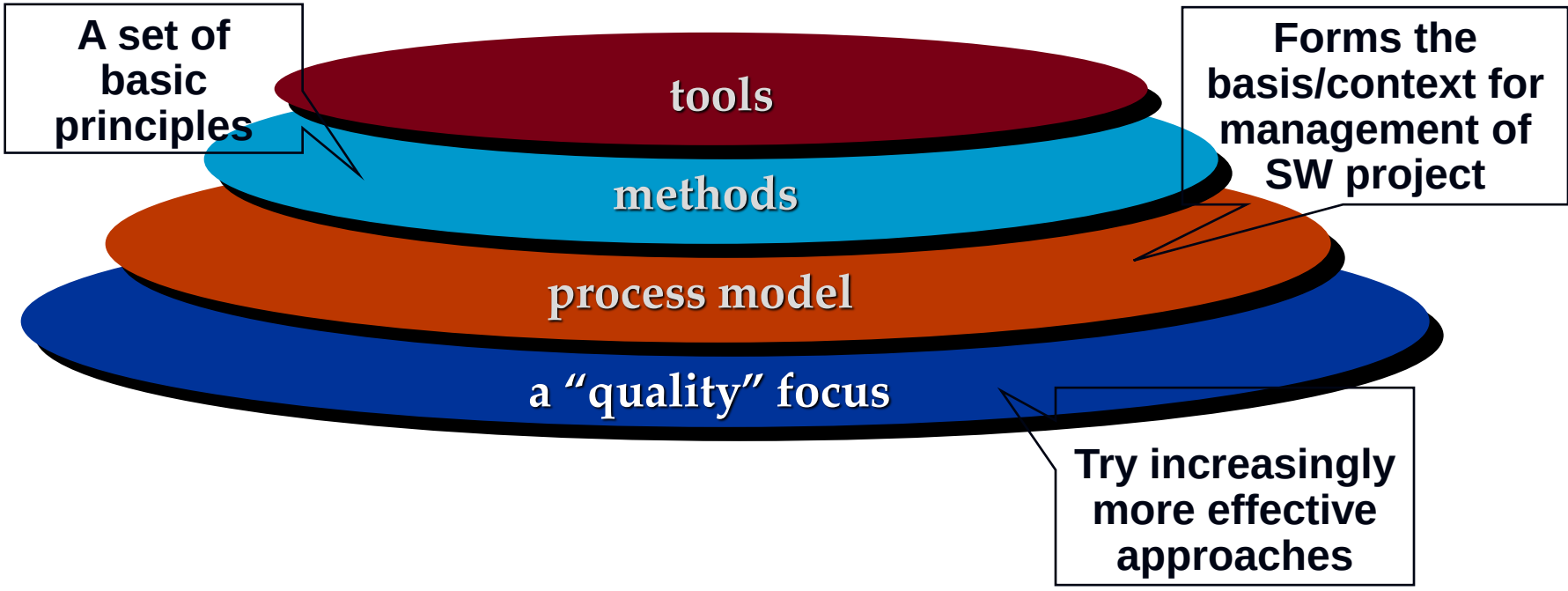
# Why Is Software Process Important?

- Every organization tried to “get the fat” out of industrial processes for more than a century
  - Ex. Toyota’s cost reduction for vehicle manufacturing
- 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



# The Essence of Practice

- Polya suggests:
  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).

# Understand the Problem

- *Who has a stake in the solution to the problem?* That is, who are the stakeholders?
- *What are the unknowns?* What data, functions, and features are required to properly solve the problem?
- *Can the problem be compartmentalized?* Is it possible to represent smaller problems that may be easier to understand?
- *Can the problem be represented graphically?* Can an analysis model be created?

# Plan the Solution

- *Have you seen similar problems before?* Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- *Has a similar problem been solved?* If so, are elements of the solution reusable?
- *Can subproblems be defined?* If so, are solutions readily apparent for the subproblems?
- *Can you represent a solution in a manner that leads to effective implementation?* Can a design model be created?

# Carry Out the Plan

- *Does the solution conform to the plan?* Is source code traceable to the design model?
- *Is each component part of the solution provably correct?* Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

# Examine the Result

- *Is it possible to test each component part of the solution?* Has a reasonable testing strategy been implemented?
- *Does the solution produce results that conform to the data, functions, and features that are required?* Has the software been validated against all stakeholder requirements?

# Hooker's General Principles

- 1: The Reason It All Exists – *to provide value to its users*
- 2: KISS (Keep It Simple, Stupid!) – *a lot of thought and work over multiple iterations to simplify the design*
- 3: Maintain the Vision – *w/o conceptual integrity, a system become a patchwork of incompatible designs*
- 4: What You Produce, Others Will Consume – *someone else will have to understand what you are doing*
- 5: Be Open to the Future - *never design yourself into a corner. Always ask "what if"*
- 6: Plan Ahead for Reuse – *achieving a high level of reuse is arguably the hardest goal*
- 7: Think! – *placing clear, complete thought before action*

# How a Project Starts (pg 16)

- **The scene:**
  - Meeting room at CPI Corporation, a (fictional) company that makes consumer products for home and commercial use.
- **The players:**
  - Mal Golden, senior manager, product development;
  - Lisa Perez, marketing manager;
  - Lee Warren, engineering manager;
  - Joe Camalleri, executive VP, business development.

**The conversation:**

- **Joe:** Okay, Lee, what's this I hear about your folks developing a what? A generic universal wireless box?
- **Lee:** It's pretty cool, about the size of a small matchbook. We can attach it to sensors of all kinds, a digital camera, just about anything. Using the 802.11 n wireless protocol. It allows us to access the device's output without wires. We think it'll lead to a whole new generation of products.
- **Joe:** You agree, Mal?
- **Mal:** I do. In fact, with sales as flat as they've been this year, we need something new. Lisa and I have been doing a little market research, and we think we've got a line of products that could be big.
- **Joe:** How big... , bottom-line

- **Mal: (avoiding a direct commitment):** Tell him about our idea, Lisa.
- **Lisa:** It's a whole new generation of what we call "home management products." We call 'em *SafeHome.* They use the new wireless interface, provide homeowners or small business people with a system that's controlled by their PC--home security, home surveillance, appliance and device control. You know, turn down the home air conditioner while you're driving home, that sort of thing.
- **Lee: (jumping in)** Engineering's done a technical feasibility study of this idea, Joe. It's doable at low manufacturing cost. Most hardware is off the shelf. Software is an issue, but it's nothing that we can't do.
- **Joe:** Interesting. Now, I asked about the bottom line.
- **Mal:** PCs have penetrated 70 percent of all households in the USA. If we could price this thing right, it could be a killer-app. Nobody else has our wireless box--it's proprietary. We'll have a 2-year jump on the competition. Revenue? Maybe as much as $30-40 million in the second year.
- **Joe (smiling):** Let's take this to the next level. I'm interested.