

CPSC 461


# Software Engineering Techniques

*Ch.1 Software and Software Engineering*





Chang-Hyun Jo

Department of Computer Science  
California State University Fullerton  
jo@ecs.fullerton.edu



# Contents



 Ch.1 Software and Software Engineering

**Part 1: The Software Process**

- Ch.2 Process: A Generic View
- Ch.3 Prescriptive Process Models
- Ch.4 Agile Development

**Part 2: Software Engineering Practice**

- Ch.5 Practice: A Generic View
- Ch.6 System Engineering
- Ch.7 Requirement Engineering
- Ch.8 Analysis Modeling
- Ch.9 Design Engineering
- Ch.10 Architectural Design
- Ch.11 Component-Level Design
- Ch.12 User Interface Design
- Ch.13 Software Testing Strategies
- Ch.14 Software Testing Techniques
- Ch.15 Product Metrics for Software

**Part 3: Applying Web Engineering**

- Ch.16 Web Engineering
- Ch.17 Formulation and Planning
- Ch.18 Analysis Modeling for Web Applications
- Ch.19 Design Modeling for Web Applications
- Ch.20 Testing Web Applications


**Part 4: Managing Software Projects**

- Ch.21 Project Management Concepts
- Ch.22 Process and Project Metrics
- Ch.23 Estimation for Software Projects
- Ch.24 Software Project Scheduling
- Ch.25 Risk Management
- Ch.26 Quality Management
- Ch.27 Change Management

**Part 5: Advanced Topics in SE**

- Ch.28 Formal Methods
- Ch.29 Cleanroom Software Engineering
- Ch.30 Component-Based Software Engineering
- Ch.31 Reengineering
- Ch.32 The Road Ahead

*Software Engineering Techniques 2005 Chang-Hyun Jo*



# Ch.1 Software and Software Engineering

---

A framework for those who build computer software – people who must get it right

The framework encompasses a process, a set of methods, and an array of tools – software engineering



## Principles of Learning

---

- 5W1H
  - What
  - When
  - Where
  - Who
  - Why
  - How



## What Is Software?

---

- **Computer software** is the product that software professionals design and build.
- It encompasses
  - Programs
  - Content
  - Documents



## Who Does It?

---

- **Software engineers build it.**
- **Virtually everyone** in the industrialized world **uses it either directly or indirectly.**



## Why Is It Important?

---

- Software affects **nearly every aspect of our lives.**



## What Are The Steps?

---

- You build software like you build any successful product, by applying a **process** that leads to a high-quality result.
- You apply a **software engineering** approach.



## What Is the Work Product?

- From the point of view of a **software engineer**, the work product is the **programs, documents, and content (data)**.
- From the **user's** viewpoint, the work product is the **resultant information** that somehow makes the user's world better.



## The Product

- **Software** is the **engine** that drives business decision making.
- **Software** serves as the **basis** for modern scientific investigation and engineering problem solving.
- It is **embedded in systems of all kinds**: transportation, medical, telecommunications, military, industrial processes, entertainment, office products ...



## The Evolving Role of Software

- Software takes on a **dual role**:
  - A **product**
    - Delivers the computing potential embodied by computer hardware
  - The **vehicle** for delivering a product
    - Acts as the basis for the control of the computer (OS), the communication of information (networks), and the creation and control of other programs (software tools and environments)
    - The most important product: **Information**



## Evolution of Software

- **Software development** was **virtually unmanaged** - until schedules slipped or costs began to escalate.



## Software Maintenance

- As the number of computer-based systems grew, libraries of computer software began to **expand**.
- Most of software had to be
  - **corrected** when faults were detected,
  - **modified** as user requirements changed, or
  - **adapted** to new hardware that was purchased.



## Software Crisis

- Effort spent on **software maintenance** escalated **high**.
- Worse yet, the personalized nature of many programs made them virtually **unmaintainable**.



## Software-Related Problems

- **Hardware** advances continue to **outpace** our ability to build **software** to tap hardware's potential.
- Our ability to build new programs **cannot** keep pace with the demand for new programs, **nor** can we build programs rapidly enough to meet business and market needs.



## Software-Related Problems

- The widespread use of computers has made society increasingly dependent on reliable operation of software.
  - **Enormous economic damage and potential human suffering can occur when software fails.**
- We struggle to **build computer software that has high reliability and quality.**
- Our ability to support and enhance existing programs is **threatened by poor design and inadequate resources.**





## An Industry Perspective

- In the early days of computing, computer-based systems were developed using hardware-oriented management.
  - Project managers focused on hardware.
  - Project managers applied the controls, methods, and tools that we recognize as hardware engineering.



## In the early days ...

- In the early days, programming was viewed as an "art form."
- The programmer often learned his/her craft by trial and error.
- The software world was virtually undisciplined.



## Today ...

---

- Today, the distribution of costs for the development of computer-based systems has changed dramatically.
- **Software**, rather than hardware, is **the largest single cost item**.



## Questions

---

- For almost two decades **managers and many technical practitioners** have asked the following **questions**:
  - Why does it take so **long** to get programs finished ?
  - Why are development costs so **high** ?
  - Why can't we find all **errors** before we give the software to our customers?
  - Why do we continue to have **difficulty in measuring progress** as software is being developed.



## An Aging Software Plant

- Like the other industry, we have an aging “software plant” - there are thousands of critical software-based applications that are in dramatic need of refurbishing.



## Example: An Aging Software Plant Information System

- Information system applications written twenty years ago that have undergone many changes and are now virtually unmanintainable.
- Even the smallest modification can cause the entire system to fail.



## Example: An Aging Software Plant Engineering Applications

---

- Engineering applications that are used to produce critical design data, and yet, because of their age and state of repair, no one really understands their internal structure.



## Software Competitiveness

---

- Software is now an extremely competitive business.
- Cost, timeliness, and quality are primary drivers that will lead to intense competition for software work over the next few decades.



## Software Competitiveness

---

- Feigenbaum and McCorduck [1983]
  - “*Knowledge* is power, and the *computer* is an amplifier of that power ...”



## Software

---

- **Software** is
  - (1) **instructions (computer programs)** that when executed provide desired function and performance, (2) **data (contents)** that enable the programs to adequately manipulate information, and (3) **documents** that describe the operation and use of the programs.



## Software Characteristics

- To gain an understanding of SW, it is important to **examine the characteristics of software** that make it different from other things that human being built.
- When **hardware** is built, the human creative process (analysis, design, construction, testing) is ultimately translated into a **physical** form.
- **Software** is a **logical** rather than a physical system element.
- Therefore, **software** has characteristics that **differ** considerably from those of **hardware**:



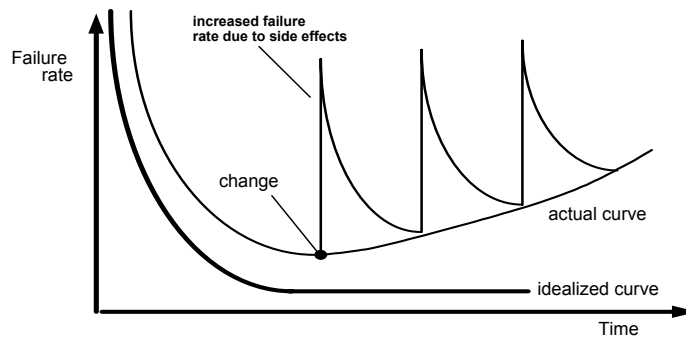
## Software Characteristics

- Software is **developed or engineered**, it is **not manufactured** in the classical sense.
- Software does **not wear out**.
  - Figure 1.1 Failure curve for hardware
  - Figure 1.2 Failure curve for software  
(idealized and actual curves)
- Most software is **custom-built**, rather than being assembled from existing components.



## Software Characteristics

- Figure 1.2 Failure curve for software (idealized and actual curves)



*Software Engineering Techniques 2005 Chang-Hyun Jo*

29



## Software Components

- In the **hardware** world, **component reuse** is a natural part of the engineering process.
- In the **software** world, it is something that has yet to be achieved on a broad scale.
- **Reusability** is an important characteristic of a high-quality software component.
- A software component should be designed and implemented so that it can be **reused** in many different programs.

*Software Engineering Techniques 2005 Chang-Hyun Jo*

30



## Software Components

- In the 1960s, we built **scientific subroutine libraries** that were **reusable** in a broad array of engineering and scientific applications.
- These subroutine libraries reused well-defined algorithms in an effective manner, but had a **limited domain** of application.
- Today, we have extended our view of reuse to encompass not only **algorithms**, but also **data structures**.



## Software Components

- Modern **reusable components** **encapsulate both data and the processing** that is applied to the data, enabling the software engineer to **create new applications from reusable parts**.





## Examples: Software Components

---

- **Interactive interfaces** are built using **reusable components** that enable the creation of graphics windows, pull-down menus and a wide variety of interaction mechanisms.
- The data structures and processing detail required to build the interface are contained within a library of **reusable components** for interface construction.



## Software Applications

---

- Software may be applied in any situation for which **a pre-specified set of procedural steps (i.e., an algorithm)** has been defined.



## Information Content

- Information content and determinacy are important factors in determining the nature of a software application.
- Information content refers to the meaning and form of incoming and outgoing information.



## Example: Information Contents

- Many business application make use of highly structured input data (a database) and produce formatted "reports".
- Software that controls an automated machine (e.g., a numerical control) accepts discrete data items with limited structure and produces individual machine commands in rapid succession.



## Information Determinacy

---

- **Information determinacy** refers to the **predictability of the order and timing** of information.



## Examples: Information Determinacy

---

- An engineering analysis program **accepts data that have a predefined order, executes the analysis algorithm(s) without interruption, and produces resultant data in report or graphical format.**



## Examples: Information Determinacy

- Example for the **indeterminate** application
- A multi-user OS **accepts inputs that have varied content and arbitrary timing, executes algorithms that can be interrupted by external conditions, and produces output that varies as a function of environment and time.**



## The Change Nature of SW Seven Broad Categories of SW

- System software
- Application software
- Engineering and scientific software
- Embedded software
- Product-line software
- Web-applications
- Artificial intelligence software



## The Change Nature of SW New Challenges

---

- Ubiquitous computing
- Netsourcing
- Opensource
- The “new economy”



## Software: A Crisis on the Horizon

---

- The term “software crisis” alludes to a set of problems that are encountered in the development of computer software.
  - The problems are not limited to software that does not function properly.
  - Rather, the affliction encompasses problems associated with how we develop software, how we maintain a growing volume of existing software, and how we can expect to keep pace with a growing demand for more software.



## Quality of Legacy Software

---

- Poor quality
  - Legacy systems sometimes have inextensible designs, convoluted code, poor or nonexistent documentation, test cases and results that were never achieved, a poorly managed change history, and etc.
  - An yet, these systems support core business functions and are indispensable to the business.
  - Nothing can be done until the legacy system must undergo some significant change.



## Software Evolution

---

- Regardless of its application domain, size, or complexity, computer software will evolve over time.
- Change (software maintenance) drives this process and occurs when error are corrected, when the software is adapted to a new environment, when the customer requests new features or functions, and when the application is reengineered to provide benefit in a modern context.



## Software Evolution

A Unified Theory for SW Evolution [Lehman 1997]

- The Law of Continuing Change
- The Law of Increasing Complexity
- The Law of Self-Regulation
- The Law of Conservation of Organizational Stability
- The Law of Conservation of Familiarity
- The Law of Continuing Growth
- The Law of Declining Quality
- The Feedback System Law



## Software Myths

- Unlike ancient myths, **software myths propagates misinformation and confusion** that have caused serious problems for managers, technical people, and customers.



## 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
  - The book of standards may very well exist, but is it used? **Is it complete?** In many cases, the answer is **no**.



## Customer Myths

---

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





## Practitioner's Myths

---

- Myth
  - Once we write the program and get it to work, our job is **done**.
- Reality
  - "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 software will be expended after it is delivered to the customer for the first time.



## Summary

---

- **Software** has become the key element in the evolution of computer-based systems and products.
- **Software** is composed of **programs**, **data (contents)**, and **documents**.
  - Each of these items comprises a configuration that is created as part of the **software engineering process**.



## Summary

---

- Software has evolved from a specialized problem-solving and information analysis tool to an industry in itself.
- Yet we still have **trouble** developing high-quality SW on time and within budget.
  - Early programming culture and history have **created a set of problems** that persist today.



## Summary

---

- The intent of **software engineering** is to provide a framework for building software with higher quality.



## References

- Pressman, Roger S. *Software Engineering: A Practitioner's Approach*, 6th Edition, McGraw-Hill, 2005. (ISBN: 0-07-285318-2)
- SEPA Web Site
  - <http://www.mhhe.com/pressman>



## Stanzas To The Po

Time may have somewhat tamed them, - not for ever;  
Thou overflow'st thy banks, and not for aye  
Thy bosom overbils, congenial river!  
Thy floods subside, and mine have sunk away;  
But left long wrecks behind, and now again,  
Borne on our old unchanged career, we move;  
Thou tendest wildly onwards to the main,  
And I - to loving one I should not love.

George Gordon Byron (1788-1824)