Overview of Software Engineering

Moonzoo Kim
KAIST

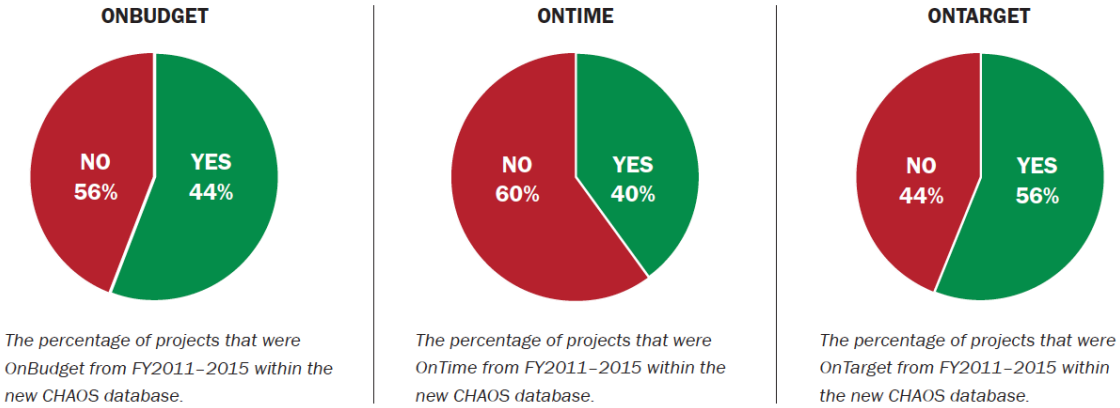
(slides from CS550 '06 taught by prof. D. Bae)

Challenges

- In 2015, only 36% of software projects were successful.

The total number of software projects is 25,000-plus, with an average of 5,000 per yearly period.

	2011	2012	2013	2014	2015
SUCCESSFUL	39%	37%	41%	36%	36%
CHALLENGED	39%	46%	40%	47%	45%
FAILED	22%	17%	19%	17%	19%



The percentage of projects that were OnBudget from FY2011–2015 within the new CHAOS database.

The percentage of projects that were OnTime from FY2011–2015 within the new CHAOS database.

The percentage of projects that were OnTarget from FY2011–2015 within the new CHAOS database.

[Chaos Report 2015, Standish Group International, Inc.]

Challenges

- Impact of project size:

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	100%	100%	100%

- Success rates by industry:

	SUCCESSFUL	CHALLENGED	FAILED
Banking	30%	55%	15%
Financial	29%	56%	15%
Government	21%	55%	24%
Healthcare	29%	53%	18%
Manufacturing	28%	53%	19%
Retail	35%	49%	16%
Services	29%	52%	19%
Telecom	24%	53%	23%
Other	29%	48%	23%

Why is it hard to make good software?

- Software is intangible
- There are no formal methods to develop software
- Software needs to be changed over time
- There are many different stakeholders who provide requirements
- Software needs to be built by a group of people
- Technology changes
- Limited budget and schedule (high competition)
- …

The Nature of Software

- **Complexity**
  - Due to a large number of interacting parts
  - Causes difficulty in finding defects
- **Conformity**
  - There are no underlying natural principles that it must conform to.
  - Makes it difficult to reuse existing software components
- **Changeability**
  - Because software is malleable
  - However, changing software is not easy…
- **Invisibility**
  - Requirements/design documents, and source code are “representations” of software
  - Software resides in computing devices and environments.
- **Uniqueness** (Brooks)
  - Software has no physical properties
  - Software is the product of intellect-intensive teamwork
  - …
  - Software alone is useless, as it is always a part of a larger system

https://sebokwiki.org/wiki/The_Nature_of_Software

Software Engineering is about …

- Understanding what to develop
- Developing software the right way
- Developing the right software

by effectively collaborating with stakeholders, including customers, users, and developers.

https://www.linkedin.com/pulse/go-digital-why-diverse-inputs-multi-stakeholder-feedback-chris-leong/

Coding vs. Programming vs. Software Engineering

Coding ⊂ Programming ⊂ SW Engineering

- Implementing an algorithm by using a programming language

- Designing an algorithm and data structures
- Integrating libraries and modules
- Debugging and testing
- …

- Modeling and specifying requirements
- Designing software architecture
- Project planning and management
- …

Done by a **Coder**

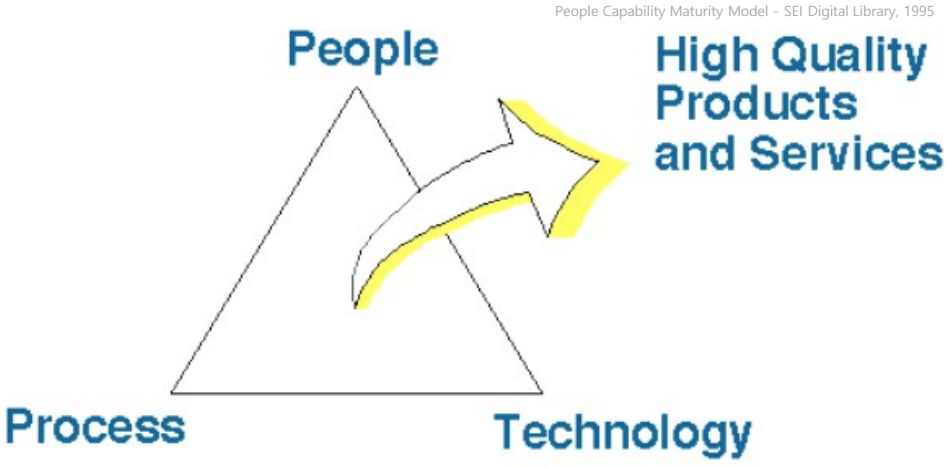
Done by a **Programmer**

Done by a **SW Architect**

Software Developer

Software Engineer

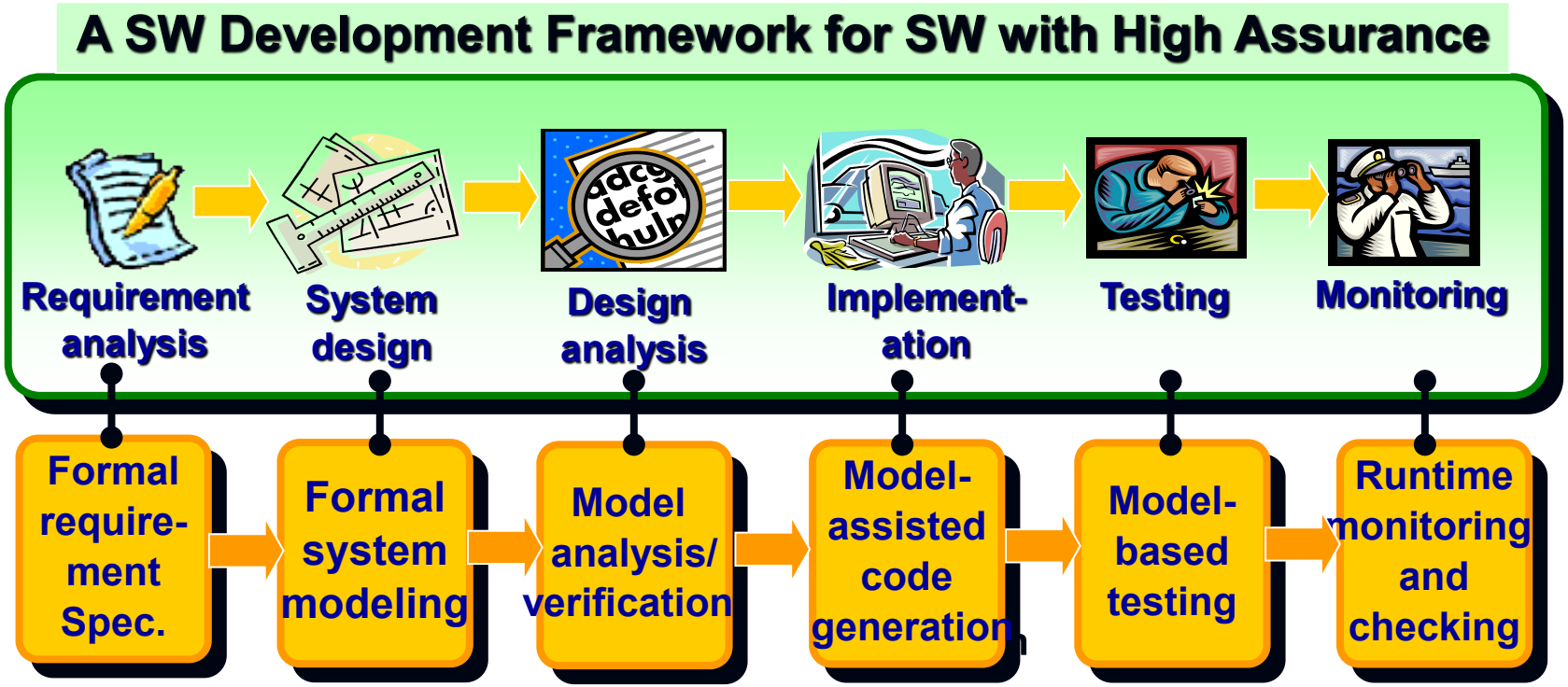
Three Aspects of Software Engineering (SE)



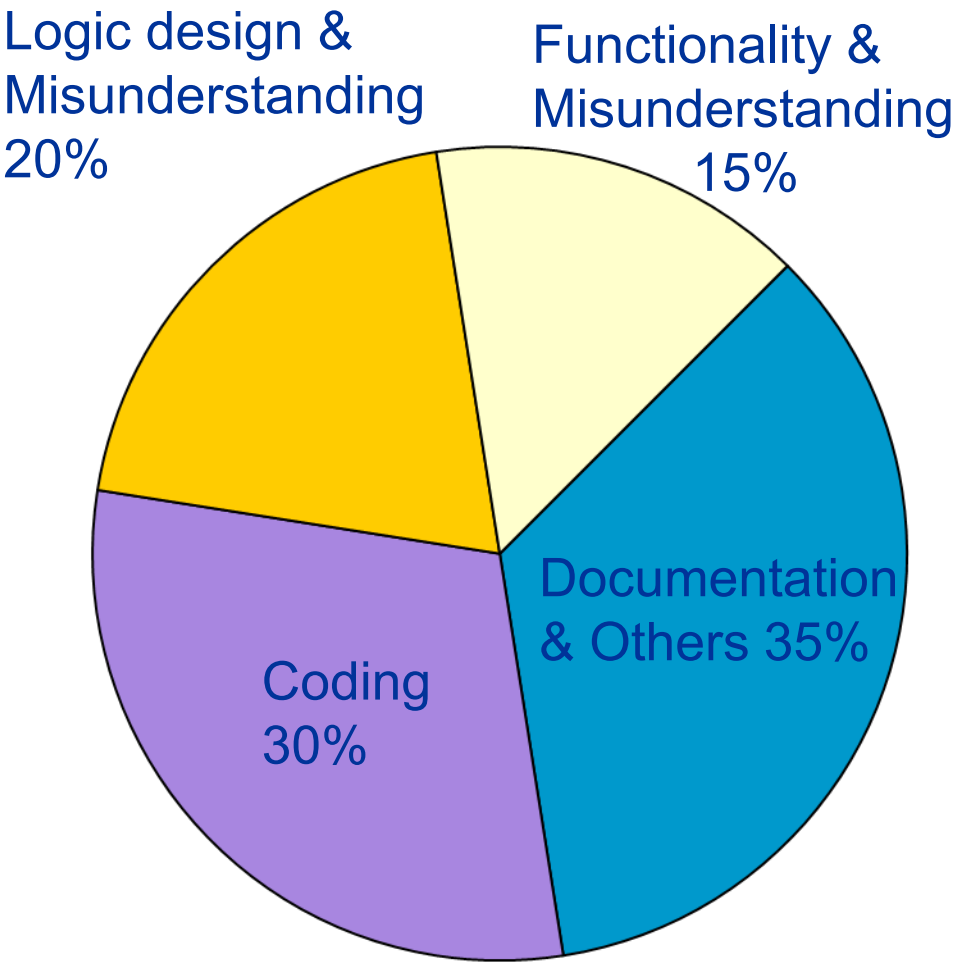
- What are examples of each aspect?
- Which one is more important than the others?

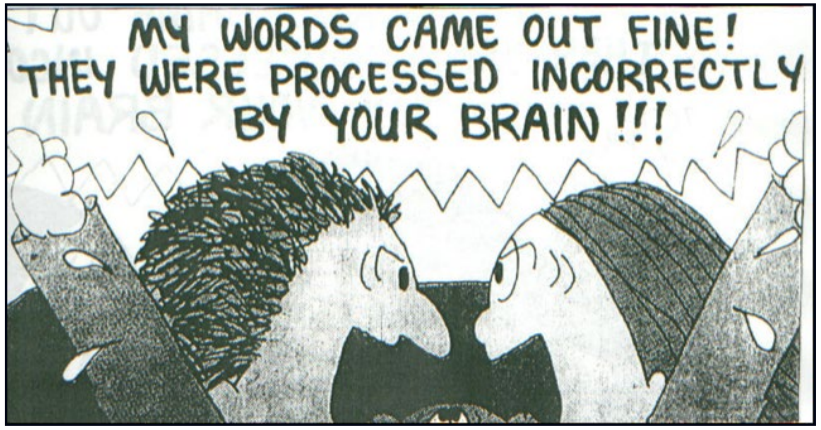
Adopted from Prof. Doo-Hwan Bae's CS350 lecture material

Software Development Process



Sources of Errors in S/W Developments





KAIST

Ex. Requirement on Retail Chain Management Software

- Find ambiguous points in the following requirement
  - If the sales for the current month are below the target sales, then a report is to be printed,
    - unless the difference between target sales and actual sales is less than half of the difference between target sales and actual sales in the previous month
    - or if the difference between target sales and actual sales for the current month is under 5 percent.

Scope of S/W Engineering

- Historical Aspects
- Economic Aspects
- Maintenance Aspects
- Specification & Design Aspects
- Team Programming Aspects

Historical Aspects

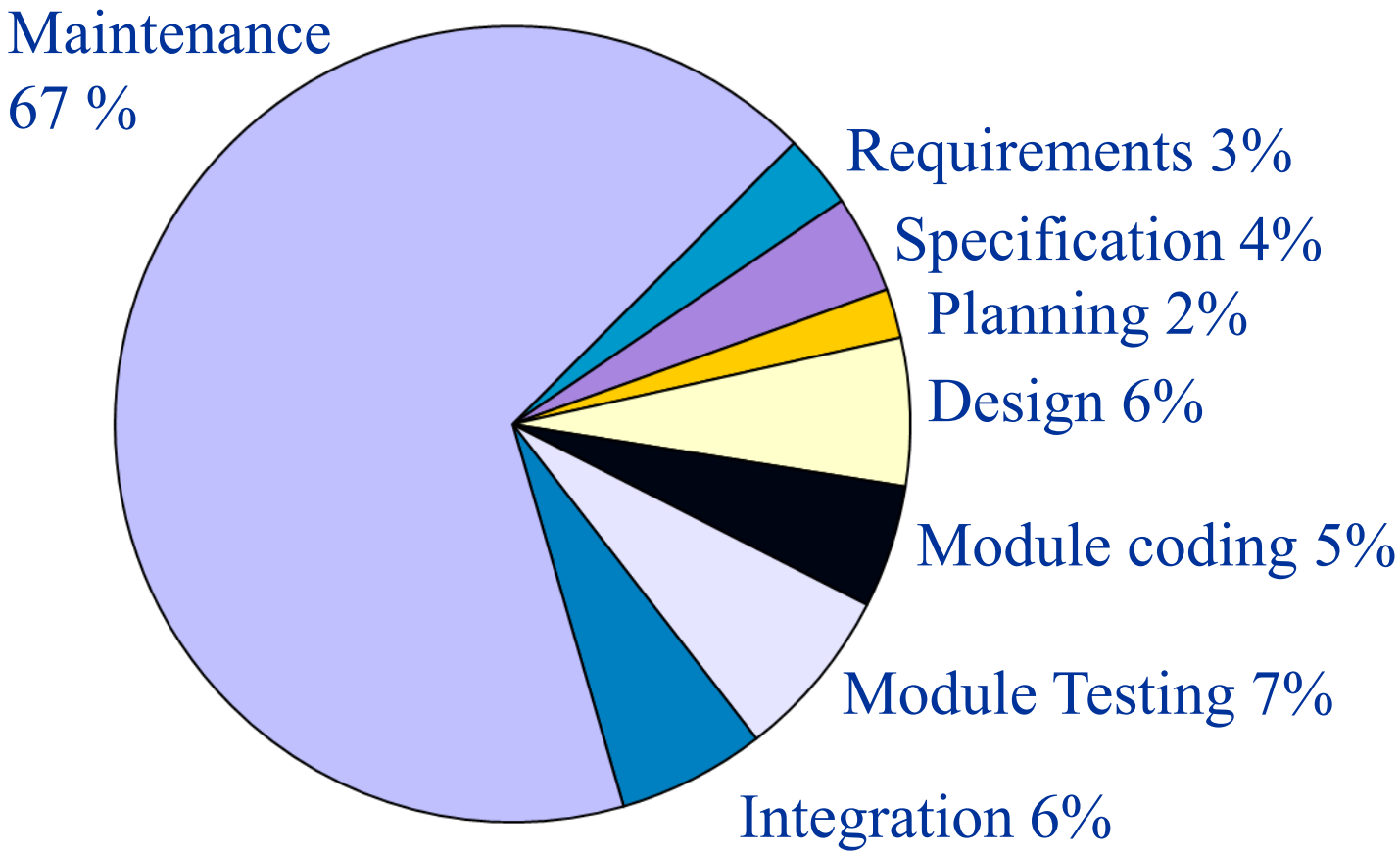
- 1967, A NATO group coined the term " Software Engineering"
- 1968, NATO conference concluded that software engineering should use the philosophies and paradigms of established engineering disciplines, to solve the problem of software crisis

Economic Aspects

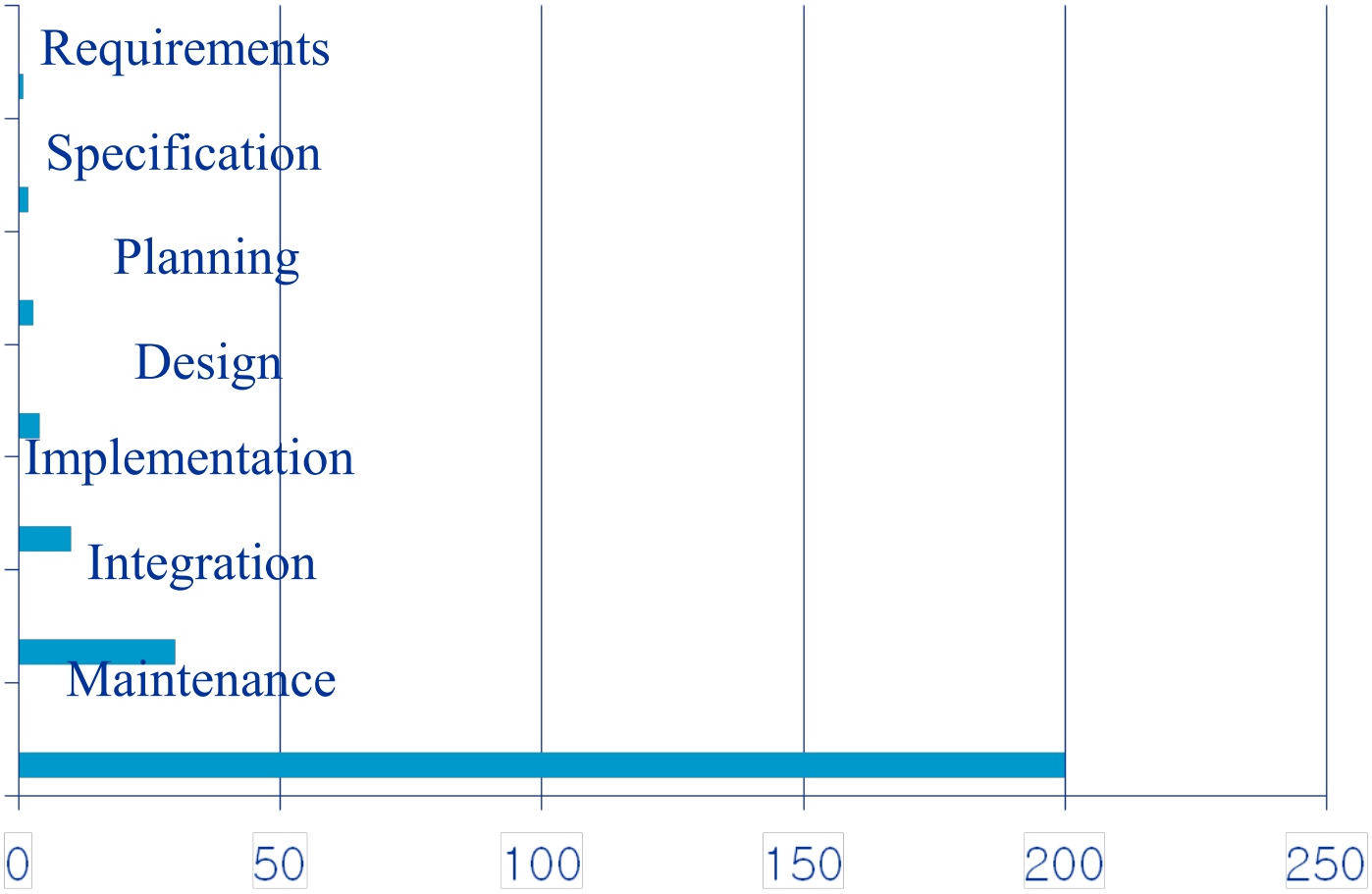
- Relationship between computer science and software engineering
  - cf: chemistry and chemical engineering

- Software engineer is intended in only those techniques which make sound economic sense, while computer scientists investigate a variety of ways of producing software, some good and some bad

Maintenance Aspects



Specification and Design Aspects



Approximate relative cost
to detect and correct a fault

Team Programming Aspect

- Parnas, "Multi-person construction of multiversion software."
  - Programming : personal activity
  - S/W engineering : team activity

Team Programming Aspect (Cont.) (From programming to sw engineering)

- Programming in early days
  - The problem is well understood.
  - Mostly scientific applications.
  - By a person, who is an expert in that area.
  - User = programmer = maintainer

- User and programmer separation
  - User: specify the problem(tasks)
  - Programmer: interpret and translate into code

Team Programming Aspect (Cont.)

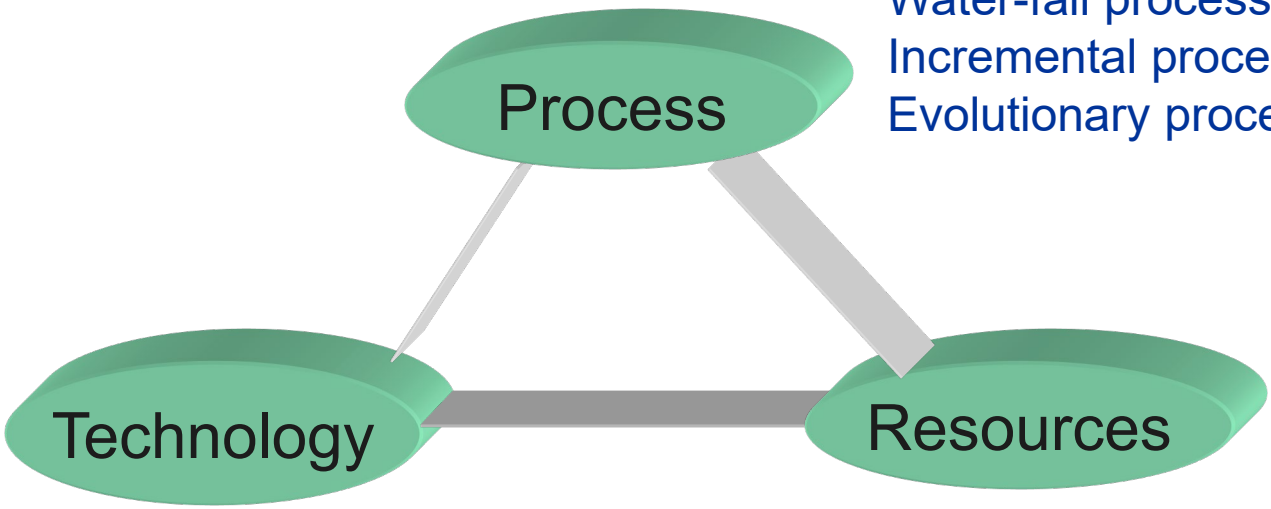
- Team project started in late 1960's
  - IBM360 Operating system
  - Software crisis observed
  - ``Software Engineering'' coined
- Solutions to software crisis
  - Management techniques
  - Team organization
    - Chief programmer team
    - Democratic team
    - Hierarchical team
  - Better languages and tools
  - Standards
  - ==> Applying engineering approach

Team Programming Aspect (Cont.)

- Requirements in the programming-in-the-small
  - Good programming skill
  - Skilled in data structures and algorithms
  - Fluent in programming languages
- Requirements in the programming-in-the large
  - Needs communication skills and interpersonal skills
  - Be familiar with design approaches (i.e. system abstraction)
    - Top-down design
    - Divide and conquer paradigm
    - Component based integration
  - Be able to translate vague requirements and desires into precise spec.
  - Be able to build and use a model of the application
  - Needs ability to schedule work

KAIST

Three Elements of S/W Development

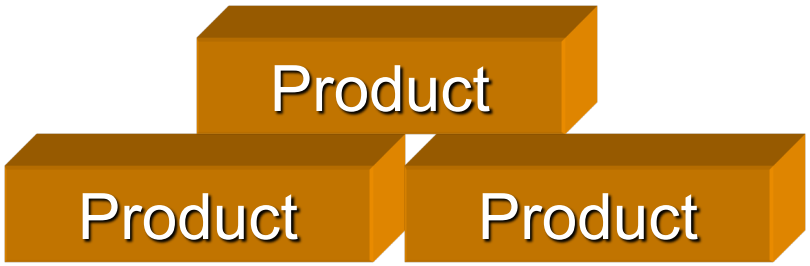


Water-fall process
Incremental process
Evolutionary process

OOP
Model
based
testing

Facilities
Staff
Budget

Quality/Productivity Improvement



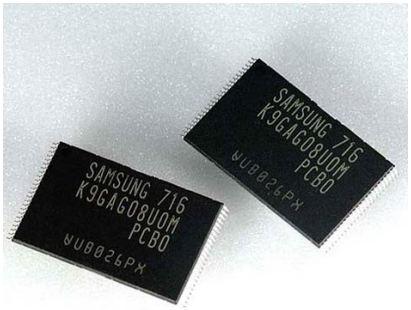
Special Software Domain:Commercial Electronics and Embedded System

Strong IT Industry in South Korea

Time-to-Market?

SW Quality?

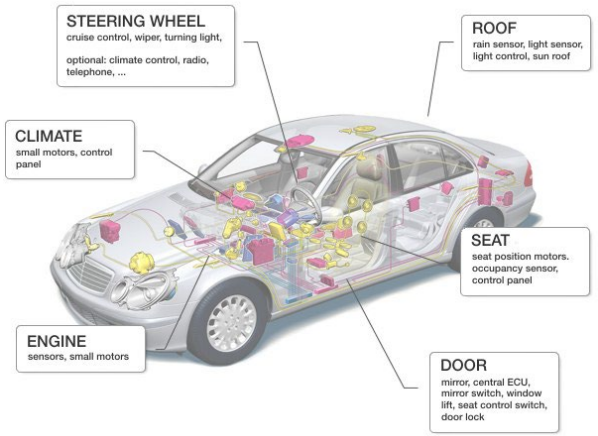
SAMSUNG



LG

KIA

KIA MOTORS



HYUNDAI

What's Different About Embedded Systems

- Embedded systems have different design constraints than general purpose systems
  - Cost may matter more than speed
  - Long life cycle may dominate design decision
    - Since ubiquitous computing paradigm occurred, this aspect is changing
  - Reliability/safety may constraint design choice
- Because applications are often unique, software development may wait for hardware to become available
  - need for simulator/emulators/etc
- Time to market constraints
  - Rapid redesign for changing form factors
  - Rapid redesign for changing control algorithms

Software Characteristics by Domain

- Ordinary IT Software System(e.g. systems developed by SI organizations)
  - Size : Very Large
  - Domain consistency: Low
  - New technology sensitivity: High
  - Hardware dependency: Low
  - Time-to-market pressure: Low

Software Characteristics by Domain

- Commercial Software(e.g. systems developed by software vendors)
  - Size : Large
  - Domain consistency: High
  - New technology sensitivity: High
  - Hardware dependency: Low
  - Time-to-market pressure: Moderate

Software Characteristics by Domain

- Controller Systems/Automation Systems
  - Size : Medium
  - Domain consistency: High
  - New technology sensitivity: Low
  - Hardware dependency: Moderate
  - Time-to-market pressure: Moderate

Software Characteristics by Domain

- Embedded Systems /Commercial Electronics
  - Size : Small
  - Domain consistency: High (-> Moderate)
  - New technology sensitivity: High
  - Hardware dependency: High
  - Time-to-market pressure: High

Software Engineering Applicability

- In general, Controller Systems/Automation Systems (and Embedded Systems /Commercial Electronics) can give much higher rewards for software engineering activity
  - Domain consistency is high and new technology sensitivity is low
    - Ease of accumulating empirical data
    - High reusability in accumulated developments assets(e.g. requirements specification, domain model, test cases, modules)
    - Ease of continuous improvement

General Obstacles

- Hardware dependency is high
  - Software development may wait for hardware to become available
  - Confident testing environment is not supported even until complete hardware is ready
  - May need for effective simulator/emulator for testing
- Time-to-market pressure is high
  - High schedule pressure causes difficulties in software engineering activities
    - Overcome the hardware dependency as much as possible
    - Set up process to reduce redundant time consumption
    - Asset reuse

Embedded Software in Two Different Domains

Conventional Testing | Concolic testing | Model checking

	Consumer Electronics	Safety Critical Systems
Examples	Smartphones, flash memory platforms	Nuclear reactors, avionics, cars
Market competition	High	Low
Life cycle	Short	Long
Development time	Short	Long
Model-based development	None	Yes
Important value	Time-to-market	Safety