Object –Oriented Software Development Using Java

Chap 1. Object-Oriented Software Development

한동대학교 전자전산학부
김 기석 교수

- peterkim@handong.edu
- http://www.handong.edu

"IT아키텍트 양성 급하다"

프로그래머 교육 편중…국내엔 100명 미만

정보시스템 개발때 '유기적 설계' 엄두 못내

소프트웨어(SW) 개발을 효율화·신속화하기 위해서는 건물의 설계사에 해당하는 정보시스템 아키텍트(architect) 양성이 시급한 것으로 지적되고 있다.

24일 관련업계에 따르면 최근 SW 개발의 효율성이 강조되면서 정보시스템 개발방식이 컴포넌트기반개발(CBD), 모델지향아키텍처(MDA) 등 전체 설계를 중시하는 방향으로 바뀌고 있지만 이를 현장에서 구현해 주는 전문 아키텍트가 거의 없어 효율적인 SW개발 취지를 제대로 살리지 못하고 있다.

특히 이같은 현상은 그동안 프로그래머 양성 위주로 진행돼 온 국내 IT 인력 육성 정책의 문제점과 방향을 함께 제시해준다는 점에서 주목된다.

실제로 미국의 경우 프로젝트매니저(PM)급은 물론 CIO나 CTO레벨에서도 아키텍트 능력을 기본으로 삼고 있지만 이에 관심이 높은 한국ITA협의회, 한국소프트웨어컴포넌트컨소시엄(KCSC) 등 유관기관과 한국솔루션센터 등 관련 업체들은 우리나라의 경우 아직까지 SW의 아키텍처 설계가 주먹구구식으로 이루어지는 것으로 파악하고 있다.

메타그룹의 경우 전체 IT인력 가운데 30%를 아키텍트로 가져갈 것을 권하고 있으나 이제까지 배출된 국내 아키텍트 인력은 100명 미만으로 전체 IT프로젝트 인력의 1%에도 못미친다는 것이 업계 전문가들의 설명이다.

아키텍트는 건물을 지을 때 설계사가 설계도면을 먼저 그려 기초공사의 뼈대를 잡는 것처럼 정보시스템 개발에서 전체 아키텍처를 설계하는 사람으로 미국에서는 아예 CSA(Chief Software Architecture)라는 별도 임원을 두고 있는 경우도 많으며 빌 게이츠 MS 회장 역시 CSA를 맡고 있다.

우리나라의 경우 아직까지 체계적이지는 않지만 이같은 상황을 개선하기 위한 움직임은 일고 있다.

〈조인혜기자 ihcho@etnews.co.kr〉

2면 'IT아키텍트'로 계속 ➡

➡ 1면에서 'IT아키텍트' 계속

지난해 말 발족한 한국ITA협의회(회장 황종선)는 올해 핵심 사업 가운데 하나로 교육사업을 진행한다. 우선 고급 아키텍트 양성을 위주로 교육을 진행할 방침이며 이를 위해 올해 두 번의 IT 아키텍트 교육과정을 계획하고 있다. KCSC(회장 오길록)는 올해 이뤄지는 CBD 교육에서 이 같은 내용을 결합해 선임 개발자들이 아키텍트로 옮겨갈 수 있도록 분위기를 조성할 방침이며 오는 3월말 개최하는 MDA 관련 국제 세미나를 통해서도 아키텍트의 중요성을 강조할 예정이다.

지난해부터 TA(Technical Architect) 컨설턴트 교육을 진행하고 있는 국내 유일의 IT아키텍처 전문업체인 한국솔루션센터 역시 올해 CIO, IT컨설턴트, 정보기획가, 7년 이상의 경력을 가진 선임 엔지니어 등을 대상으로 TA교육을 강화할 방침이다.

한국솔루션센터 박성범 사장은 "아키텍트가 아직까지 제대로 양성되지 않아 재사용보다는 무조건 갈아엎는 식의 정보시스템 구축이 만연해 있다"며 "점점 복잡해지면서 빠른 개발을 요구하는 정보시스템 개발 추세에 대응하기 위해서라도 아키텍트 양성이 시급하다"고 전했다.

- 소프트웨어 아키텍팅이란 단지 기존의 기술활동 이상의 것이며, 이는 프로젝트에서 이해관계자들의 비즈니스 요구사항과 기술적인 요구사항에 관해 균형을 맞추는 것입니다.
  - Randy Stafford (in 97 things every software architect should know)

- Software architect is a computer pro grammer that makes high-level desi gn choices and dictates technical st andards, including software coding s tandards, tools, or platforms.

표 1 아키텍트 기술 영역

기술 영역	필수 기술 요소	비고 (관련 자료)
아키텍처 수립	아키텍처 패턴 수립 방법론 (프로세스)	SA in Practice 1 POSA 1 2
설계	디자인 패턴 설계 표기법 / 설계 언어 설계 도구 (1가지)	Design Pattern [3] UML Distilled [4]
구현	프로그래밍 언어 (1가지) 개발 도구	

SW 아키텍트 평가모델 나온다

연합회, 내달 개발 완료… 4개 항목별 세부 콘텐츠 마련중

강동식 기자 dskang@dt.co.kr | 입력: 2011-01-31 20:24

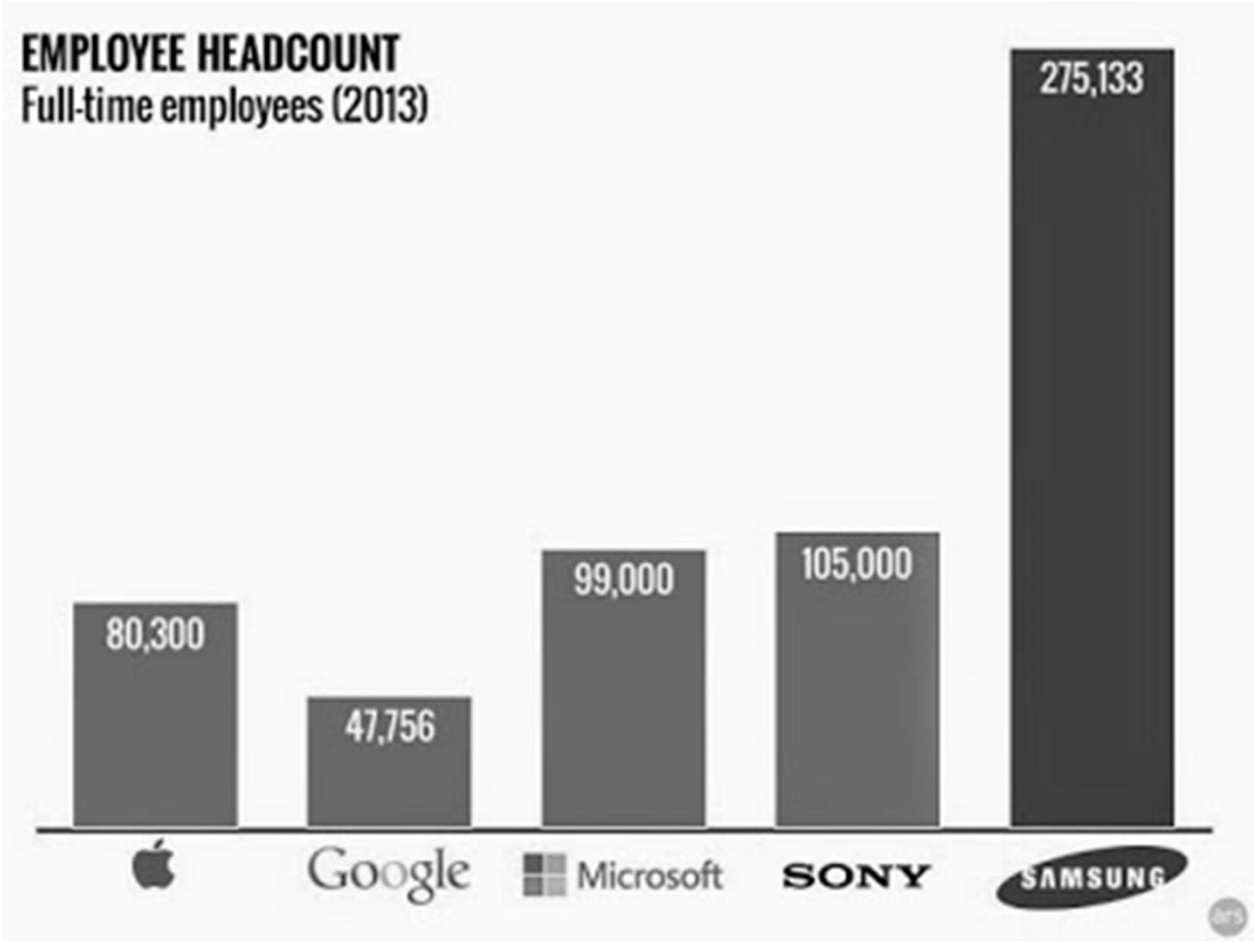
최상위 소프트웨어(SW) 기술 전문가인 SW 아키텍트(Architect) 평가인증모델이 나온다.

SW 아키텍트는 SW 프로젝트에서 다양한 요구사항을 분석해 시스템의 구조를 설계하는 SW기술 분야의 최상위 전문가이다. 국내는 SW 아키텍트 절대 부족이 SW 프로젝트나 SW 제품의 수준 향상을 가로막고 있어 체계적이고 장기적인 육성이 필요한 것으로 지적되고 있다.

한국SW아키텍트연합회는 최근 SW아키텍트|인증평가모델 초안을 개발했으며, 3월 초에 최종 결과물을 개발할 예정이라고 31일 밝혔다.

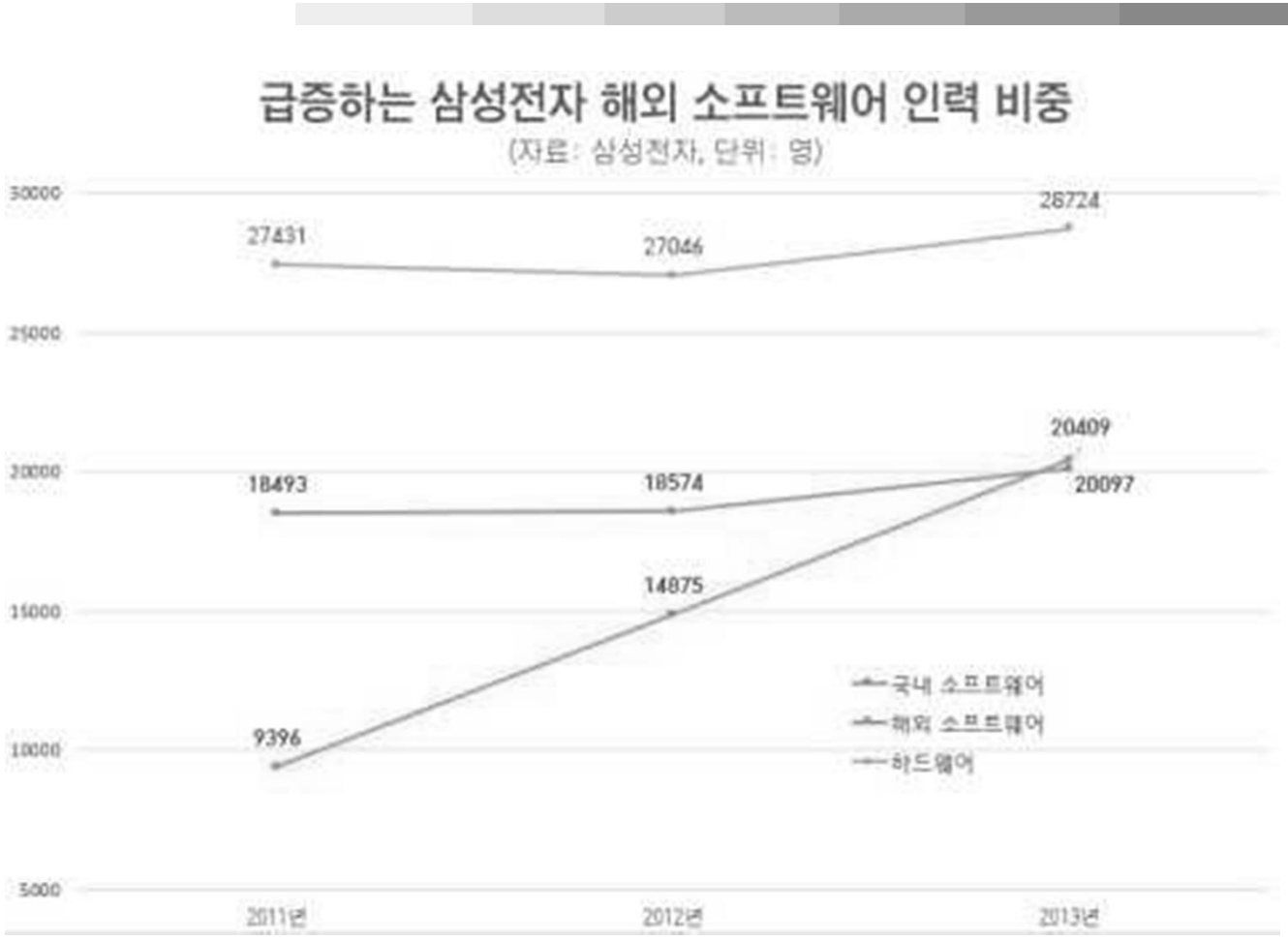
SW아키텍트연합회 관계자는 "아직까지 SW 아키텍트의 국내 수준, 범주, 역할이 제대로 정리된 것이 없어 SW아키텍트연합회를 중심으로 SW아키텍트 관련 정책과 기술, 역량, 시장을 정리하고 있다"며 "3월 초까지 SW아키텍트 평가인증모델 개발을 완료하는 것이 목표"라고 말했다.

삼성 & 애플 & 구글


EMPLOYEE HEADCOUNT
Full-time employees (2013)
80,300
47,756
99,000
105,000
275,133
Google
Microsoft
SONY
SAMSUNG

- 삼성이 이 많은 사람들로 무엇을 하고 있을까? 2013년에소프트웨어 엔지니어 수는 40,506명이다. 실제로 구글의 직원 중 18,593명만 "연구와 개발"(이라 쓰고 소프트웨어 개발이라 읽는다)을 담당하므로, 소프트웨어 엔지니어 수만 놓고 보면 삼성은 구글의 두 배다. 소프트웨어 군단은 삼성이 최근 확보한 자원이다. 소프트웨어 인력 수는 2011년 이후 45퍼센트 성장했다.

- 하지만 삼성의 2배 가까운 소프트웨어 엔지니어는 구글과같은 영향력을 미치지 못한다. 터치위즈나 삼성의 중복된 안드로이드 생태계 앱들을 생각해보자. 회사는 안드로이드와 터치위즈를 만들어내는 모든 신형 스마트폰에 이식한다. 매년 70개에 이르는 디바이스를 배포하고, 2년 동안 모든 것을 지원해야 한다. 엄청나게 큰 프로젝트다



전문가들은 소프트웨어 분야의 해외 연구·개발 인력 확대도 삼성전자의 '불가피한 선택'에 가깝다는 분석을 대체로 내놓고 있다.

LG전자 휴대폰 개발을 책임졌었던 정옥현 서강대 교수(전자공학)는 "신흥국을 겨냥한 모바일 기기 개발과 특급 소프트웨어 개발 인력 확보를 위해서는 해외에서 소프트웨어 인력을 대규모로 확충할 필요가 있다"고 설명했다. 정 교수는 "신흥국 스마트폰 사업이 확대되면 원가를 절감하기 위해 한국 본사 연구소에서 핵심 플랫폼만 만들고, 신흥국에서 나머지 개발 과정을 맡는 형태로 갈 수밖에 없다"고 말했다.

소프트웨어 경쟁력 확보에서 소수의 최고급 개발자 확보가 중요해지는 것도 이유다. 정 교수는 "정말 능력이 뛰어난 개발인력을 구하기 위해서는 해외에 연구소를 세우고, 글로벌IT기업과 전방위적인 인재 확보 경쟁을 벌여야만 한다"고 말했다.

삼성전자는 "글로벌 시장 공략을 위해서는 각국 실정에 맞춰 스마트폰을 조정하는 과정이 필수적"이라며 "해외 소프트웨어 인력이 늘 수밖에 없다"는 입장이다. 삼성전자는 또 "국내 소프트웨어 인력 규모가 상대적으로 협소해 다른 나라에서 인력을 뽑지 않을 수 없다"고 설명했다.

Preface

- Object Oriented Software Development
  - Areas played crucial roles in maturing OOSD
    - 1) UML (Unified Modeling Language) as the de facto standard. (no compiler)
    - 2) Publication of *the Design Patterns* (by Gamma) – object oriented frameworks
    - 3) The Evolution of OOPL culminated in the emergence and popularity of Java

Preface

- Objectives of This Lecture
  - Broad and coherent coverage of OO Tech
  - OO Modeling using UML
  - OO Design using design patterns
  - OO Programming using Java
    - primary focus on design and programming
  - (*) Design Practices Using Rational Rose
  - not for introductory course on programming in Java

Preface

- Programming and Design in OO Paradigm
  - Two Distinct tasks
  - however, more tightly intertwined than conventional Programming Paradigm
  - requires a new(OO) way of thinking
    - Through the use of design patterns
    - exploration of the design of Java Class Libraries
    - iterative(incremental) software development
  - OODS는 자바의 문법과 library를 배우는 것과는 차원이 다른 문제임

Preface

- Chap 1. General overview of OOSD
- Chap 2. OO Paradigm and UML
- Chap 3-5. Java Programming
- Chap 6-7. Key issues of OOPD(Design Patterns)
- Chap 8. Three important framework in Java
- Chap 9. Iterative development (using design patterns and refactoring)
- Chap 10. Examples of Design Patterns(with examples implemented in java)
- Chap 11-12. Concurrent and Distributed computing.

Chap 1. OO Software Development

- Chapter Review
  - overview of object oriented software development
  - 1) software development processes and the desirable qualities of software products.
  - 2) difficulties of software development
  - 3) iterative software development processes
    - Rational Unified Process (RUP) – 2003년 2월 IBM
    - Extreme Programming (XP)

Chap 1. OO Software Development

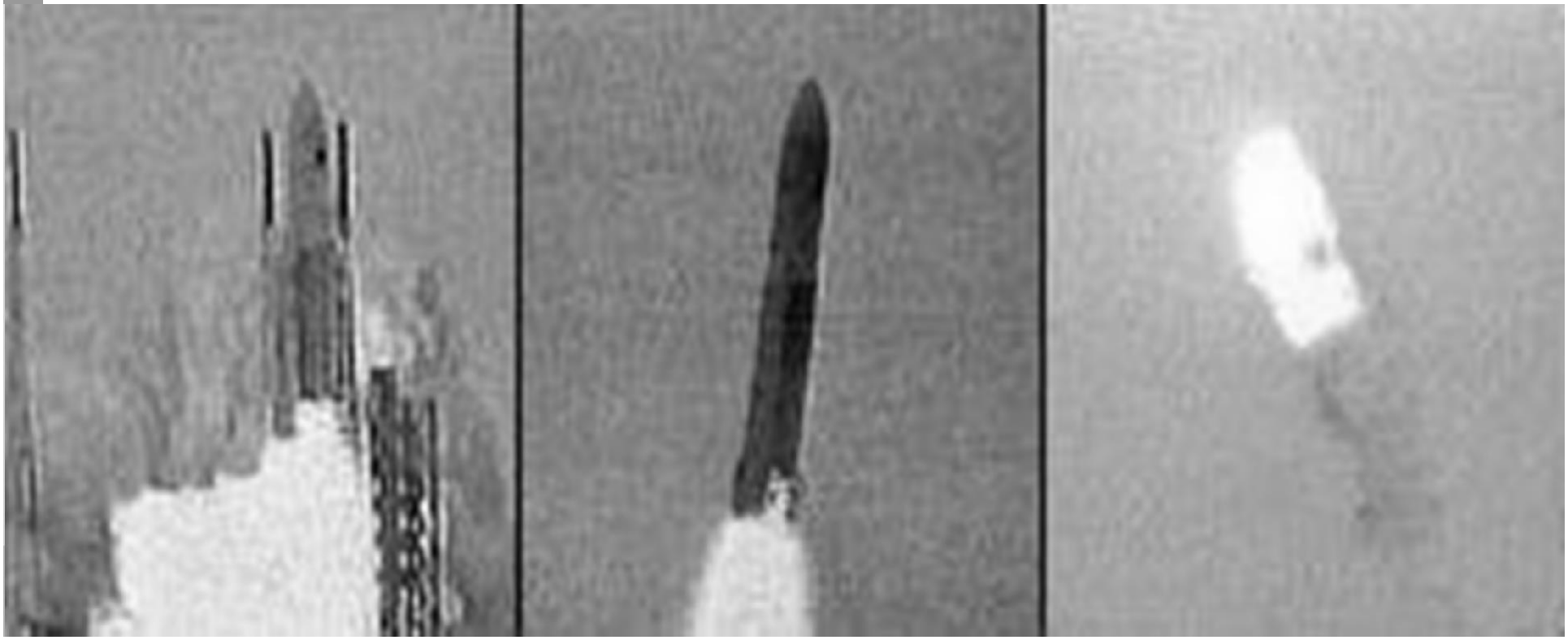
- Software Industry
  - the most successful industries during the 1980s and 1990s
    - its growth in market exponential
    - able to deliver technologically advanced and innovative products at an unrelenting pace
  - Problems : expensive
    - The Cost of purchasing, development, main taining and upgrading Software systems
    - ⓪ Object Oriented Software Development....

1.1 The Challenges of Software Development

- Process of Creating Software Products
  - 1) difficult, time-consuming, costly endeavor
    - Ex) initial version of Windows NT
      - 6 million lines of Code
      - cost $150 million to develop
      - took 200 developers,testers and technical writer 5 years to complete
  - 2) Buggy
    - Ex) 15 Jan 1990, AT&T long distance telephone network broke down – ill placed “Break” in the C language (more than 8 hours)
    - Ex) 4 June 1996, Ariane 5 communication satellite launcher – exploded 37 secs after lift-off, 64bit floating points to a 16 bit signed integer……

June 4, 1996

Debris falls to earth



1.1 Software Development

- Buggy software is the norm
  - objectives : in delivering high-quality software on time and under budget.
- Underlying causes of the difficulties of software development
  - 1) complexity(??)
  - 2) longevity(??) and evolution (??)
  - 3) high user expectations(??)

1.1 Software Development

- 1) Complexities
  - software systems are very large and complex
  - no individual can comprehend every details of the systems.
  - large systems must be developed by teams
  - (Requirements 만도 너무 많은. 별도 관리 Tool 필요)-Top 20 Requirement Management Software - http://www.capterra.com/requirements-management-software/
  - (그래서 Communication skill 중요함, documentation 기술 중요함,(일본의 문서화문화)

1.1 Software Development

2) Longevity and Evolution

- Longevity : in service for very Long Lifetime
  - Some legacy system(??):operating for more than 20 years
- Evolution
  - must constantly evolve to accommodate changes in users' needs and environments
  - making changes to software is very difficult
  - Ex) Y2K Problems
- Maintenance
  - costly and time consuming
  - usually degrades the quality of the systems being maintained.
  - the maintenance costs is far greater than initial costs.

1.1 Software Development

- 3) High User Expectations
  - In the pasts, the users of computers are experts.
  - But, Now, nontechnical, ordinary people
  - Expected to be “Bug Free”

1.2 An Engineering Perspective

- Software Engineering
  - coined at a NATO workshop in 1968
    - to build the practice of software development on a solid scientific foundation
    - to attain the level of reliability and productivity associated with well –established engineering principles
    - 기존 engineering discipline 으로 부터 software engineering 의 특성을 살림.
  - 1) define the various activities
  - 2) defines the software development processes
- Software(??)

1.2.1 Software Development Activities

- Software means
  - not only the software source code
  - but also, associated documentation,
    - requirements specification
    - architecture and design documents
    - configuration data
    - installation and user manuals

1.2.1 Software Development Activities

- Software Development Activities
  - 1) Requirements Analysis
    - to establish the functions, services, and constraints of software to developed.
    - two types of users
      - 1) custom software : specific customer
      - 2) commercial software (Shrink-wrapped) : potential software
    - two categories of requirements
      - 1) functional requirements
      - 2) nonfunctional requirements (constraints : response time, memory consumption, and user friendliness)

1.2.1 Software Development Activities

- 2) Design
  - establish an overall architecture of the software
  - by partitioning the software into components or subsystems (modules)
  - by identifying the relationship and dependencies among them.
  - 1) system design
  - 2) detail design

1.2.1 Software Development Activities

- 3) Implementation and Unit Testing
  - Implementation : realization of source code
  - Unit Testing : test each individual components
- 4) Integration and System Testing
  - integrated and tested as a whole
- 5) Maintenance
  - 1) correcting bugs
  - 2) improving performance
  - 3) enhancing functions or services
  - 4) adapting to new environments

1.2.2 Software Development Processes

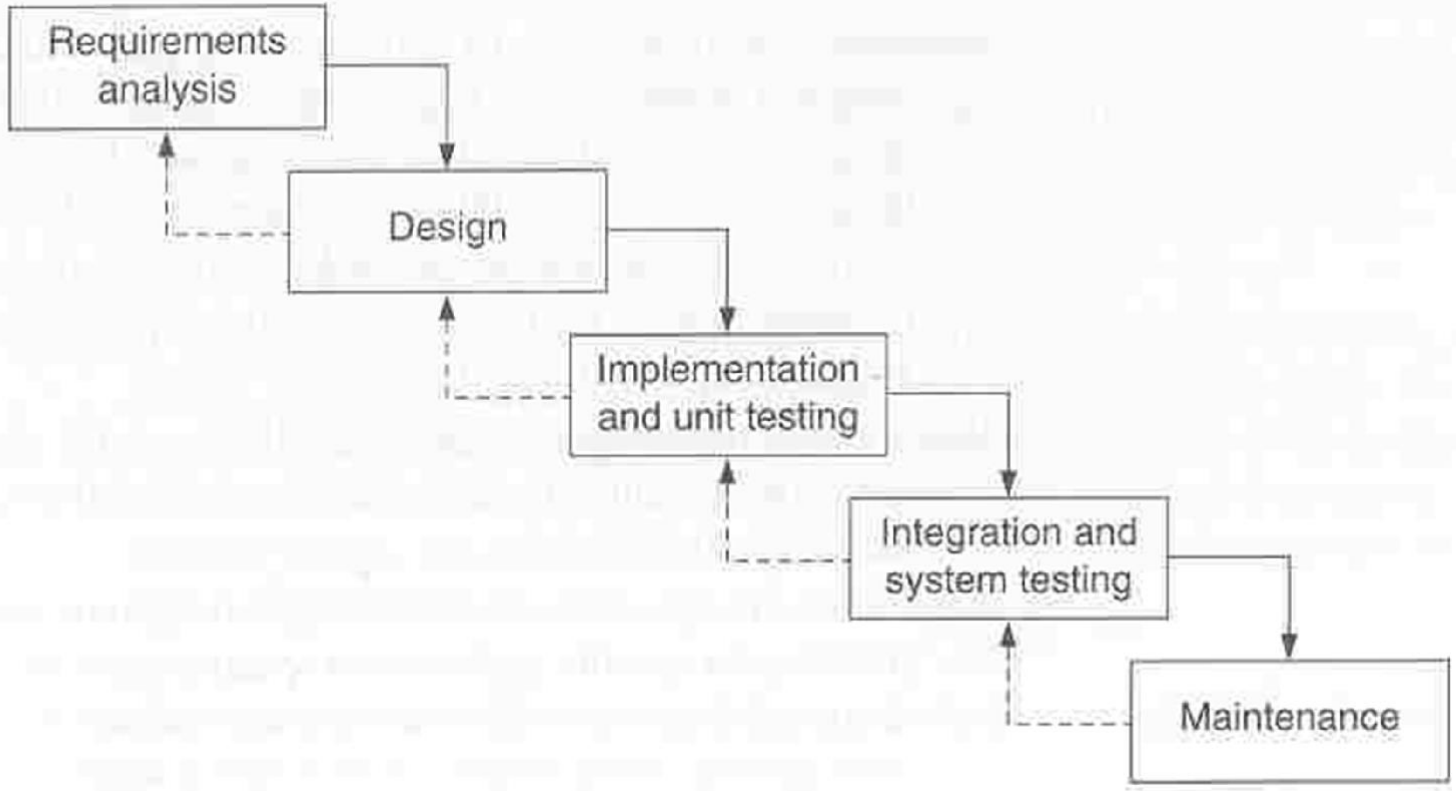
- Waterfall Model
  - the most well-known software development process
  - De facto(??) standard
  - to minimize the changes after the documents are delivered
    - the tasks of each phase be completed thoroughly
    - The deliverables of each phase be frozen once they are delivered and approved(=="signed off")
  - But, not realistic (does not facilitate such iterations)
  - ⓪ iterative fashion

1.2.2 Software Development Processes

- Waterfall Model

Figure 1.1

The waterfall model of software development.



1.2.3 Desirable Qualities of Software Systems

- Usefulness (??)
- Timeliness (??)
- Reliability (??)
- Maintainability
- Reusability
- User Friendliness
- Efficiency

1.2.3 Desirable Qualities of Software Systems

- 1) Usefulness
  - Address the needs of their intended users
- 2) Timeliness
  - Software systems should be completed and shipped in a timely manner
- 3) Reliability
  - correctness of the functions being performed
  - the availability of services , an acceptable level of failures.
- 4) Maintainability
  - should be easily maintained
  - making corrections, adaptations, and extensions without undue costs.
- 5) Reusability
  - Components – not as ad hoc solutions but as general solution in different context.
  - can be adapted and reused many times

1.2.3 Desirable Qualities of Software Systems

- 6) User Friendliness
  - should provide user friendly interface .. To facilitate easy use and access to the full extent of the systems' capabilities
- 7) Efficiency
  - should not make wasteful use of system resources
- Trade – Off
  - Not all of these desirable qualities are attainable at the same time. ⓞ 이러한 trade-off를 잘 다루도록 하는 것이 SE의 이슈.
  - Ex) the Maintainability and timeliness

- Why Maintainability ??

1.2.3 Desirable Qualities of Software Systems

- Maintainability should be focused for three reasons
  - 1) First, with long lifetimes, maintenance costs will far exceed initial development costs.
  - 2) Second, current development technologies cannot yield high reliability in the initial release of Software System.
    - repeated corrections
  - 3) third, high maintainability requires flexibility in the design and implementation of software systems
    - ⓞ This flexibility facilitates incremental development.

- How maintainability ??
  - Flexibility(??), Simplicity(??), Readability(??)

1.2.3 Desirable Qualities of Software Systems

- Factors that contribute to the maintainability of software systems
  - 1) Flexibility
    - Various aspects of software systems should be easily changeable
    - The correctness of the changes can be verified locally
    - The impact of the changes can be confined to small regions
  - 2) Simplicity
    - if things are simple, people are much less error-prone
    - Complex software systems can be simplified by the effective use of the divide-and conquer technique
  - 3) Readability
    - prerequisite for maintainability
    - understandability
    - Clarity and simplicity of the design and program code, accompanying documentation, a simple and consistent style of design, implementation and documentation

1.2.4 Is Software Development an Engineering Process?

- Software Engineering
  - little consensus on the definition
  - using software engineer as a professional title..still being debated.
- [Shaw and Garlan 1996] Software engineering ....
  - refers to a collection of management processes, software tooling, and design activities for softw are development.
  - differs significantly from the practice of older forms of engineering(??).

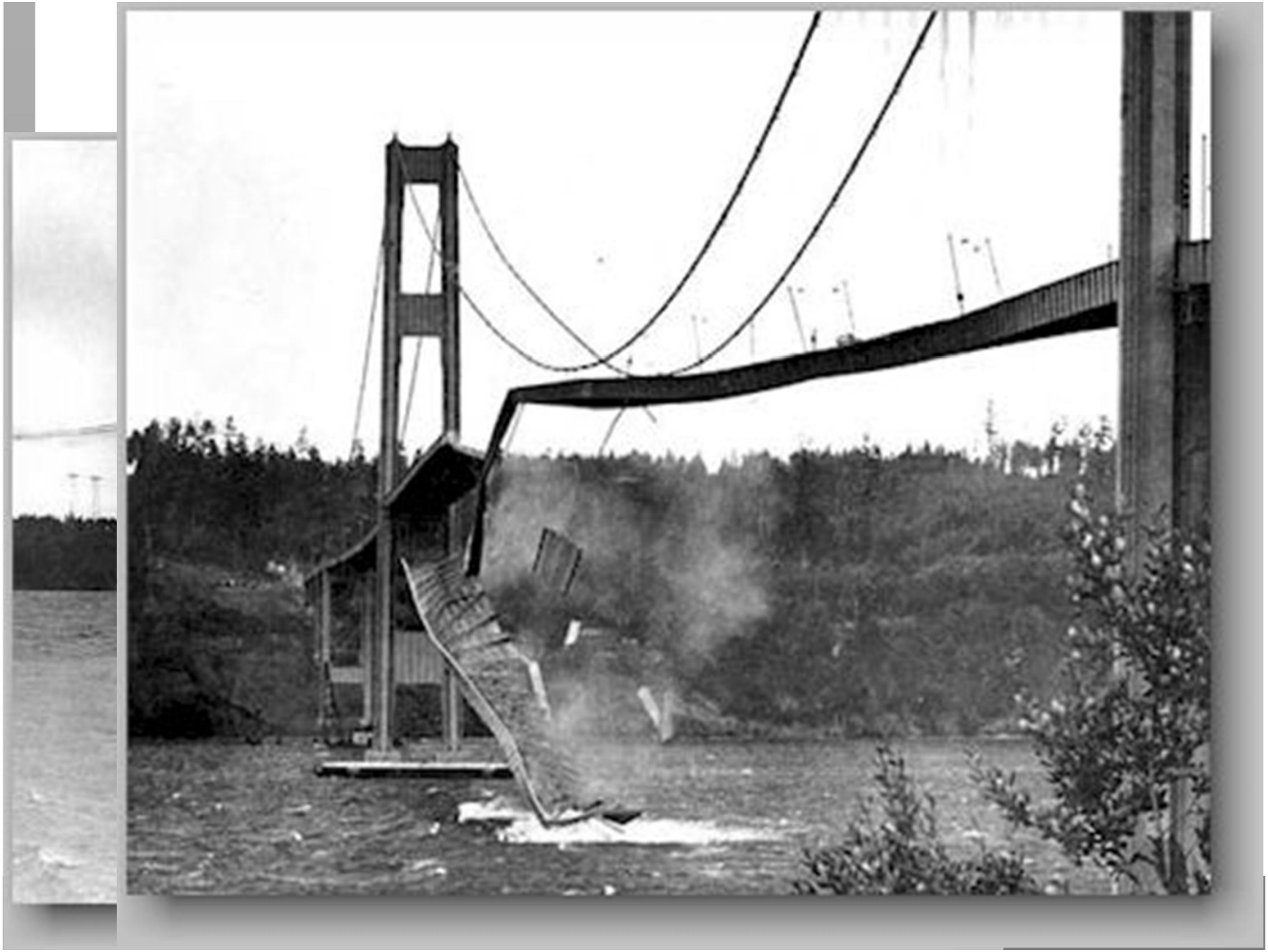
1.2.4 Is Software Development an Engineering Process?

- A. Analysis of Designs
  - craftsmanship
    - through trail and error during current and prior construction
  - But Modern engineering
    - offers assurance, predictability and efficiency that craftsmanship cannot match.
    - can be assured beforehand through scientific analysis of their designs
  - Software development
    - is like building modern skyscrapers with craftsmanship....
    - depending on testing and debugging (trial and error)

1.2.4 Is Software Development an Engineering Process?

- B. Nonrecurrence of Failures
  - Failures
    - ex) collapse of the Tacoma Narrows Bridge in 1940
      - did not foresee that the slender bridge deck would act like an airplane wing...
      - 바로 결과를 분석하여, 같은 에러가 발생치 않도록...
      - 동영상 : 공진(Resonance)
      - 공진 : 강제진동의 진동수**(Forced frequency)**와 물체의 고유진동수**(Natural frequency)**가 일치할 때 진폭이 점점 커지는 현상
      - **http://youtu.be/_uJb7T7L5II**
      - https://youtu.be/lXyG68_caV4 동영상

1940.7.1



Same type of Failures

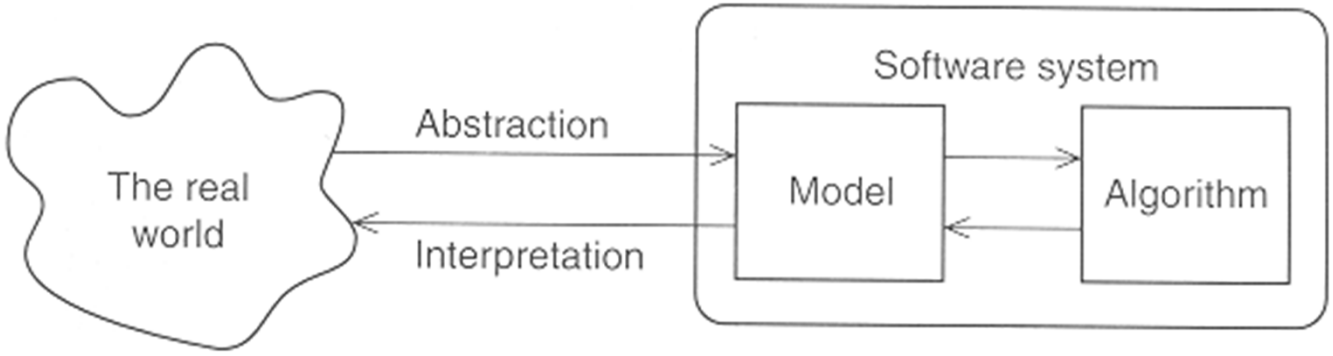
- In software Development, the Same type of failures recur all the time.
- The sad truth about software development is that no one can ensure that the type of failure in Ariane 5 will never occur again

1.2.4 Is Software Development an Engineering Process?

- C. Codification of Knowledge
  - The success of well-established engineering fields is due largely to the accumulation and codification of knowledge and the reuse of prior solutions.
  - But, In software development, very little has been systematically codified.
  - Each design is treated as an original
  - ⓪ Software design is difficult, time-consuming, and unreliable

1.3 OBJECT ORIENTATION
1.3.1 Modeling the Real World

- Two components of Software system
  - 1) model : representation of a pertinentpart of the real world;
  - 2) Algorithm : which captures the comp utations involved in manipulating or pro cessing the model



1.3.1 Modeling the Real World

- Abstraction/Interpretation
  - 1) Abstraction
    - the real world : fuzzy, unknown, intangible
    - programming model
      - must be precise and relative small
      - necessarily an abstraction of the real world
      - captures “only” the essential and relevant characteristics of the real world
      - From a particular perspective

1.3.1 Modeling the Real World

- 2) Interpretation
  - Models are intended to be manipulated or processed
  - Interpretation : assignment of meanings to the entities in the models.
  - the results of manipulations can be fed back to the real world through interpretation

- PL(Programming Language)
  - main tools used by software developers to describe computer models

1.3.2 Evolution of Programming Models

- How Does someone model the real world ? – through the changing views of computer m odel
  - 1) Computation-oriented models
    - 1950,60... Focus was on the algorithm
    - main concern
      - 1) solving computation models
      - 2) Designing efficient algorithms
      - 3) controlling the complexity of computation
    - Decomposition of complex systems was primary on (e x. Gotoless, subroutine, block structure)

1.3.2 Evolution of Programming Models

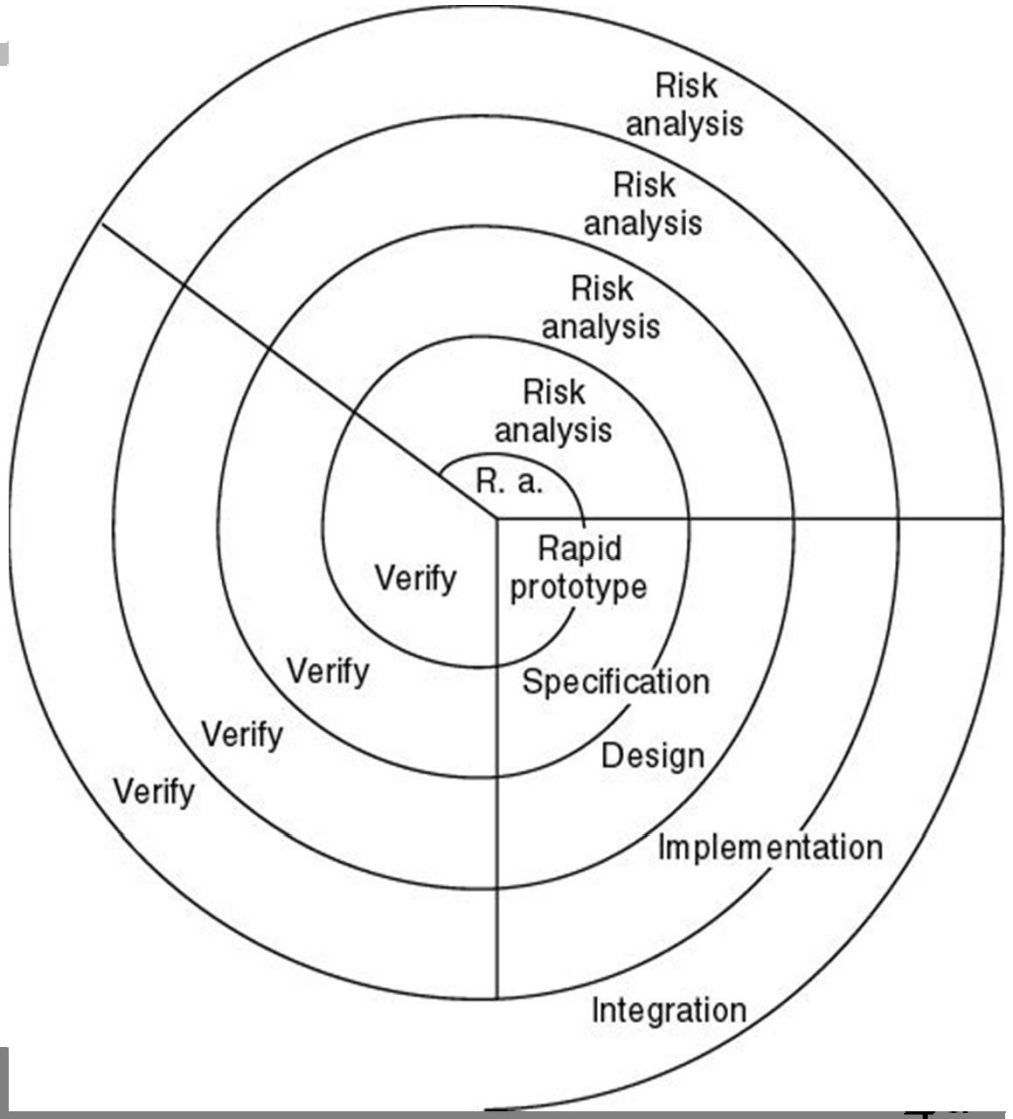
- 2) Data-Oriented Model
  - in the 1970s, 1980s
  - to address the complexity of the data being processed.
  - centered on data entities and data flows
  - decomposition of complex systems was primarily based on data flow
- 3) Objected-Oriented Model
  - a balanced view of the data and computation .
  - composed of objects (== data and the associated computations)
  - decomposition of complex systems is based on the structure of objects, classes, the relationships among them

1.3.3 A Brief History

- History of OO Development
  - late 1960s, Simula (Simulation language)
    - first PL included oo feature, ex) class
  - Smalltalk (Xerox PARC in the 1970s)
    - First full-blown OOPL
  - 1980s
    - OOPL (C++, Objective-C, Eiffel)
  - Recently
    - only recently OO Development approach matured
    - OOPL(C++,Java), OO analysis and modeling notation(ex. UML), frameworks, design patterns, iterative development process.

1.4 Iterative Development Processes

- Spiral Model [Boehm, 1988]
  - first iterative software development process



1.4 Iterative Development Processes

- Booch's iterative oo development process [Micro Processes]
  - 1) identifying the classes
  - 2) identifying the semantics (attributes and behaviors of the classes)
  - 3) identifying the relationships among the classes
  - 4) defining the class interface
  - 5) implementing the classes

  - Grady booch (부치) RATIONAL 소프트웨어 수석과학자. UML 개발자

1.4 Iterative Development Processes

- Booch's iterative oo development process[Macro Processes]
  - to serve as the controlling framework of the micro process.
  - analysis, modeling, design, implementation, maintenance
  - ⓪ RUP(Rational Unified Process) – complete
  - ⓪ XP(Extreme Programming) – lightweight..

1.4.1 Object Oriented Development Activities

- 1) Conceptualization
  - to establish the vision and the core requirements of the software system.
  - Establishing the complete requirements of the system.

- 2) Object-Oriented analysis and modeling
  - to build models of the system's desired behavior, using ex.UML
  - use cases and class diagrams

1.4.1 Object Oriented Development Activities

- 3) Object-oriented design
  - to create an architecture for implementation
  - in terms of objects, classes, the relationships among them.
  - key concern of OOD
    - 1) satisfy all the stated requirements and constraints
    - 2) flexible for future changes and enhancements
    - 3) feasible for implementation , can it be implemented efficiently ?

1.4.1 Object Oriented Development Activities

- 4) Implementation
  - using OOPL(ex. Java)
  - coding, unit testing, debugging
  - key issues
    - 1) correct?
    - 2) efficient and maintainable?
    - 3) robust ? (capable of tolerating faults and recovering from failures?)

1.4.1 Object Oriented Development Activities

- 5) Maintenance
  - to manage post delivery evolution
  - removing bugs
  - enhancing functionalities
  - adapting to evolving needs and environments

1.4.1 Object Oriented Development Activities

- Iterative Development Processes
  - try to facilitate and manage changes
  - 1) Each iteration is relative small and can be completed in a relative short period of time
  - 2) Each iteration results in a release of an executable product or component, which is a part of the final product.

1.4.2 Rational Unified Process

- RUP
  - Complete Software Engineering Process
  - Provides guidelines for every phase
  - Goal : ensure the production of high quality s oftware that meets the needs of its end users within a predictable schedule and budget.
  - not one process
  - but a process framework that can be adapted and extended to different organizations and projects

- RUP
  - IBM의 래셔널 소프트웨어 부서에서 만든 객체 지향 개발 방법론
  - RUP는 하나로 고정되어 쓰인 프로세스가 아니라, 적응이 가능한 프로세스 프레임워크
  - 개발 조직과 소프트웨어 프로젝트 팀이 필요한 바에 따라서 프로세스의 요소들을 선택하여 조절할 수 있도록 설계됨
  - 래셔널 소프트웨어사에서 개발
  - IBM에 2003년 2월에 합병
  - 샘플 산출물과 다양한 활동에 대한 자세한 설명을 바탕으로 한 서로 연결된 지식-베이스를 포함
  - RUP는 사용자가 쉽게 개발 과정을 수정할 수 있는 IBM Rational Method Composer (RMC) 라는 제품에 포함되어 있음

1.4.2 Rational Unified Process

- The Key Practices of the RUP
  - 1) develop software iteratively
  - 2) systematically elicit, organize, and manage changing requirements.
  - 3) use component-based architecture
  - 4) visually model software using UML
  - 5) continuously verify software quality
  - 6) control changes to software

1.4.2 Rational Unified Process

- Emphasis of RUP is On Building models rather than paper documents.
- 9 models (collectively cover all the important decisions)
  - 1) Business Model : Establishes an abstraction of the organization
  - 2) Domain Model : Establishes the context of the system.
  - 3) Use Case Model : Establishes the system's functional requirements
  - 4) analysis model(optional) : an idea design

1.4.2 Rational Unified Process

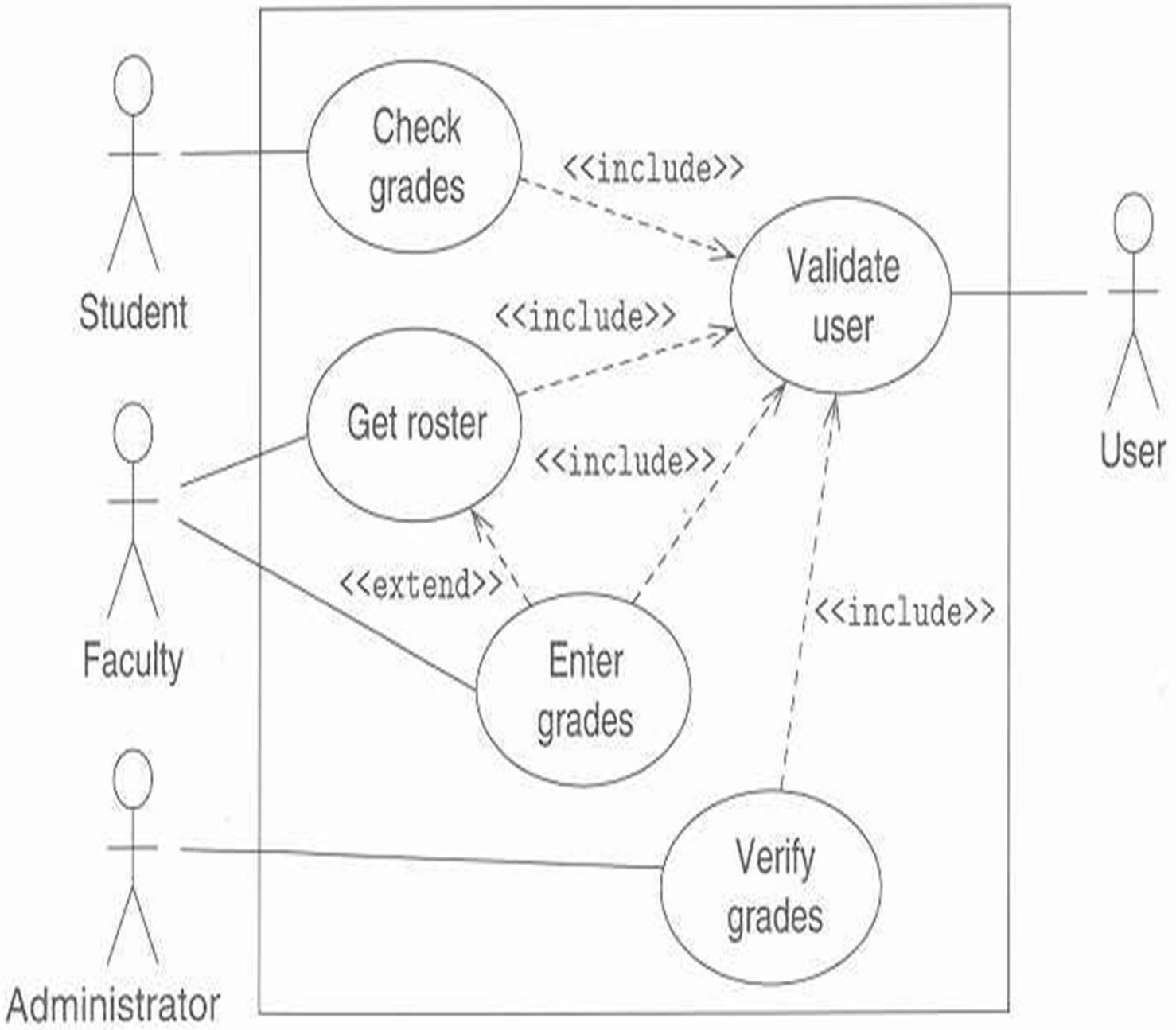
- 5) Design Model : establishes the vocabulary of the problem and its solution.
- 6) Process Model(optional) : establishes the system's concurrency and synchronization mechanisms
- 7) Deployment Model : the hardware topology on which the system is executed
- 8) implementation model : establishes the parts used to assemble and release the physical system
- 9) Test Model : establishes the paths by which the system is validated and verified.

1.4.2 Rational Unified Process

- RUP
  - use case driven
    - use cases
      - defined for system requirements
      - the foundation for all other development activities, including design, implementation and testing
  - architecture centric
    - the main focus of early iteration of the develop ment process is to produce and validate an ex ecutable architecture prototype

Figure 2.17

Dependency relationships among use cases.

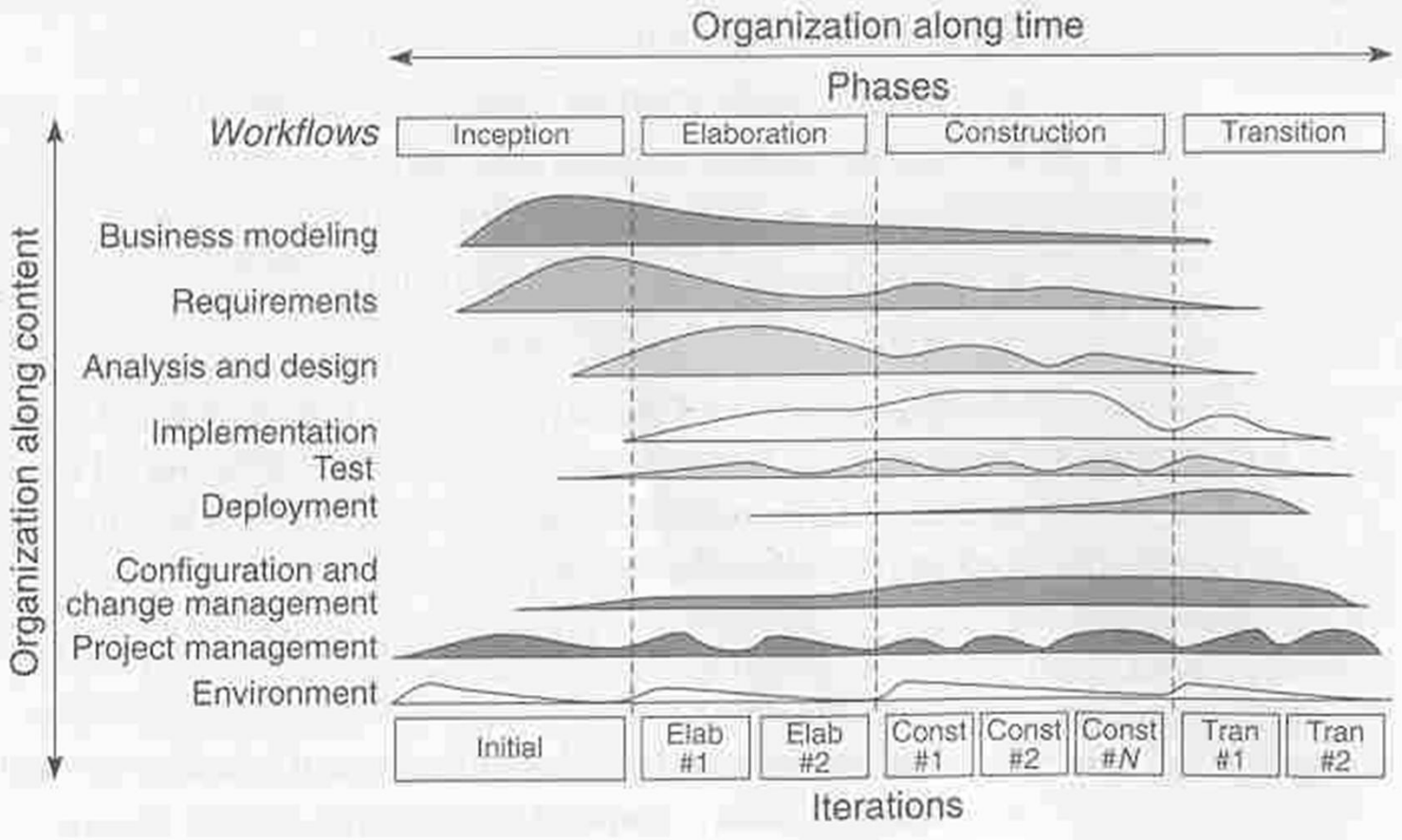


1.4.2 Rational Unified Process

- Process structure of the RUP

Figure 1.2

Rational Unified Process. (From Kruchten [2000] The Rational Unified Process, An Introduction. Addison-Wesley.)



1.4.2 Rational Unified Process

- 1) First Dimension : Workflow
  - a workflow
    - consists of a sequence of activities that produce a set of artifacts, or deliverables, which can be project plans,design models, source code, tests and documentations
  - Nine process workflow
    - 1) business modeling : the structure and dynamics of the organization
    - 2) Requirements : the use case-based method for eliciting requirements
    - 3) Analysis and design : the multiple architectural views.

1.4.2 Rational Unified Process

- 4) Implementation
- 5) Test : test cases, procedures, and defect-tracking metrics
- 6) Deployment : all the deliverable system configurations
- 7) Configuration management
- 8) Project management
- 9) Environment : covers the necessary infr astructure required to develop a system.

1.4.2 Rational Unified Process

- 2) Second Dimension : Phases and iterations
  - four major Phases
    - 1) Inception : Establishes the Business case for the project
    - 2) Elaboration : Establishes a project plan and a sound architecture
    - 3) Construction : grows the system.
    - 4) Transition : supplies the system to its end users
  - more iterations
    - iterations in different phases have different emphases on process workflows.

1.4.3 Extreme Programming

- Extreme Programming
  - lightweight process for producing high-quality executable code throughout the development process.
  - focuses on executable code from the very beginning
  - iterative process with small iteration
    - each iteration : a few days, a few weeks
    - first iteration : produce a minimum, skeletal, and executable implementation
    - focus of each iteration : enhancement or refactoring

1.4.3 Extreme Programming

- emphasizes maintaining high quality in the code delivered by each and every iteration
  - 1) enhancements : new functionalities or features
  - 2) refactoring : restructure the code to improve the quality , including extensibility and maintain ability, and the structure of the software system (p. 252 – 255)

1.4.3 Extreme Programming(*)

- Step of Extreme Programming
  - 1) Development Team determines the various features(stories)
  - 2) for each such features, Team informs the clients how long & how cost to implement
  - 3) the clients selects the features using cost-benefit analysis
  - 4) the proposed build is broken down into smaller pieces(tasks)
  - 5) A programmer first draws up test cases for a task

1.4.3 Extreme Programming(*)

- 6) working with a partner on one screen, the programmer implements the tasks
- 7) test all the test cases
- 8) the task is integrated into the current version of the product
- (all members of the XP team work on sp ecifications, design, code and testing.)

1.4.3 Extreme Programming

- Key Concepts of XP
  - Planning Game (start with a simple a plan for each iteration, and continually refine the plan as necessary)
  - Frequent and small releases
  - use Metaphor (with the customers)
  - Simple design(refactor later if changes are necessary).
  - Test First (write unit test before writing code)
  - Refactoring (refactor to make the system simpler and clearer or to reduce duplication) p 253.
  - Pair Programming(write all production code in pairs)

1.4.3 Extreme Programming

- Collective ownership (Anyone may change code anywhere)
- Continuous integration
- 40-hour week
- On-Site customer (have a customer available on-site and full time)
- Coding Standards (adopt common standards and conventions for naming, source code formatting, documentation, and so on)

■ RUP vs. XP

- RUP : emphasizes building OO Models with UML
- XP : emphasizes producing executable code.