소프트웨어 본질

소프트웨어 역할의 진화

- 1970s and 1980s
  - “새로운 산업혁명”
  - “제 3의 물결”
  - “정보 사회”
  - “전자 정부”
- 1990s
  - “권력 이동”
  - “지식의 민주화”
  - “Y2K 시간 폭탄”
- 2000s
  - “보다 지능적으로, 보다 유연하게”
  - “새로운 형식의 과학” (컴퓨터 시뮬레이션, 딥런닝)

Alvin Toffler

"작가/미래학자"
"제3의 물결"
"권력 이동"
"부의 미래"

"4차산업혁명?"

하드웨어

소프트웨어

마인드웨어

소프트웨어 역할의 진화

- 오늘날
  - 상호 이해와 소통
  - 소셜 네트워크, 사물 네트워크
  - 쓰레기 정보/지식 “Obsoledge” (Obsolete + Knowledge)

    ➔ 빅데이터 (데이터 → 정보 → 지식 → 마음 → 지혜)
- 초대형 소프트웨어 업체

  ➔ 아마존, 구글, 알리바바: 드론/자율자동차 …
- 개개의 프로그래머에서 소프트웨어 전문가 팀으로 재편

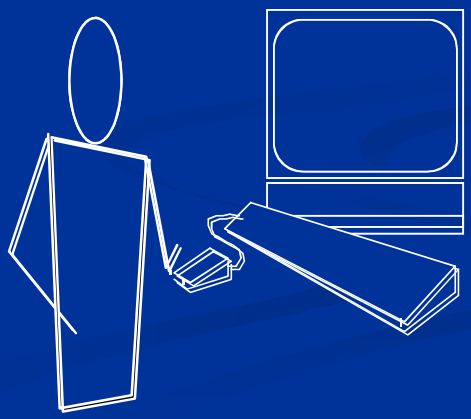
소프트웨어의 본질적 문제

- 소프트웨어가 완성되기까지 왜 그렇게 많은 시간이 필요한가?
- 소프트웨어 개발 비용이 왜 그렇게 많이 드는가?
- 소프트웨어를 고객에게 전달하기 전에 왜 모든 에러를 찾지 못하는가?
- 개발된 프로그램을 유지보수 하는데 왜 그렇게 많은 시간과 노력이 필요한가?
- 소프트웨어를 개발하고 유지보수 할 때 진척 정도를 측정하는 것이 왜 어려운가?

소프트웨어 정의

소프트웨어는 아래의 것들을 중심으로 구성되는 집합이나 객체를 말한다.

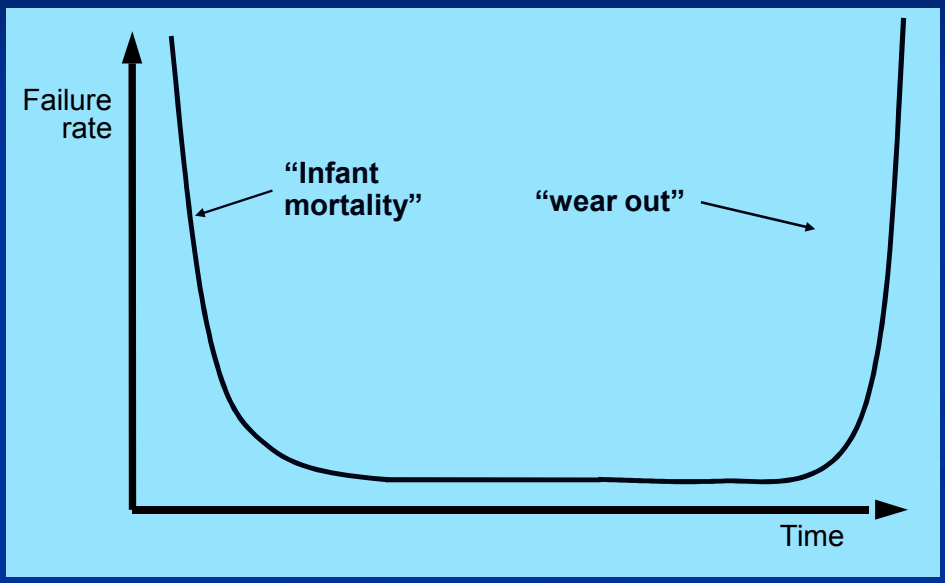
- 프로그램
- 문서
- 데이터 ...



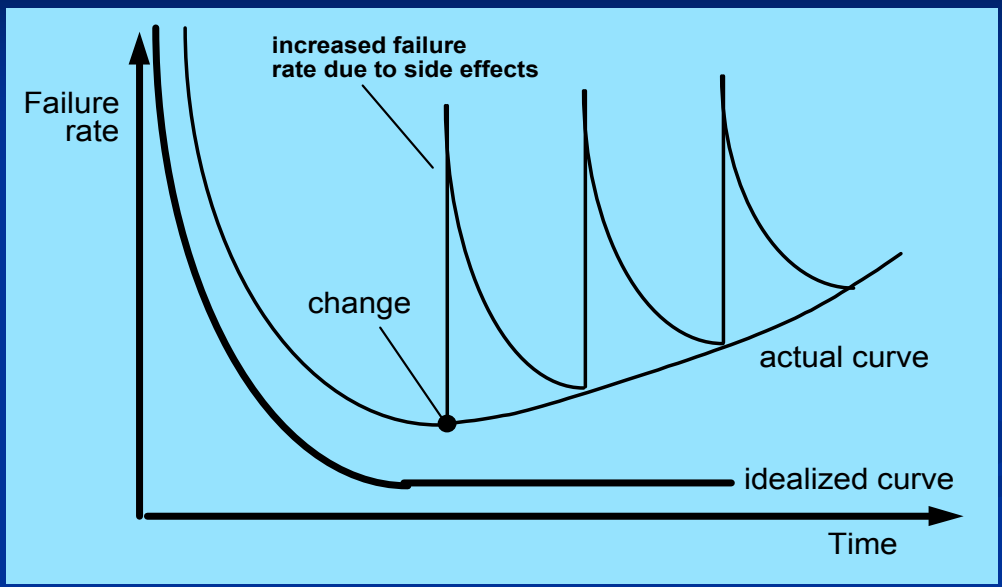
소프트웨어의 현실

1. Software is developed or engineered; it is not manufactured in the classical sense.
   - Similarities exist between software development and hardware manufacturing.
2. Software doesn't "wear out"
   - "bathtub curve"
   - Software doesn't wear out. But it does deteriorate!
   - There are no software spare parts.
   - Software maintenance involves considerably more complexity than hardware maintenance.
3. Although the industry is moving toward component-based construction, most software continues to be custom built.
   - In the hardware world, component reuse is a natural part of the engineering process.
   - In the software world, it has only begun to be achieved on a broad scale.
   - A Software component should be designed and implemented so that it can be reused in many different programs.

하드웨어의 고장율 곡선 "욕조 커브"



소프트웨어의 고장율 곡선



소프트웨어 응용 분야 (1/4)

- System S/W
  - System software is a collection of programs written to service other programs.
  - E.g., compliers, editors, and file to management utilities
  - E.g., operating system components, drivers, networking software, telecommunications processors
- Application S/W
  - Business or technical data
  - Management/technical decision-making
  - E.g., point-of-sale transaction processing, real-time manufacturing process control

소프트웨어 응용 분야 (2/4)

- Engineering/scientific S/W
  - “number crunching” algorithms
  - From astronomy to volcanology
  - From automotive stress analysis to space shuttle orbital dynamics
  - From molecular biology to automated manufacturing
  - Computer-aided design, system simulation, and other interactive applications have begun to take on real-time

소프트웨어 응용 분야 (3/4)

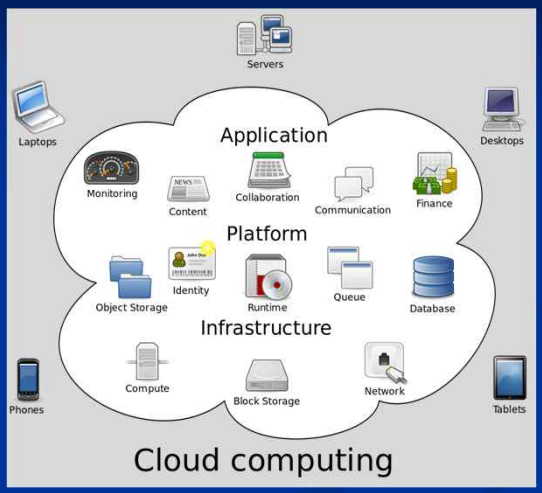
- Embedded S/W
  - Implement and control features and functions
  - Limited and esoteric functions (e.g., keypad control for a microwave oven)
  - Provide significant function and control capability (e.g., digital functions in an automobile such as fuel control, dashboard displayer, braking systems, etc.)
- Product-line S/W
  - Limited and esoteric marketplace (e.g., inventory control products)
  - Mass consumer markets (e.g., word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, personal and business financial applications).
- Artificial intelligence S/W
  - Robotics, expert systems, pattern recognition, artificial neural networks, theorem proving, and game playing Deep Learning (AlphaGo)

소프트웨어 응용 분야 (4/4)

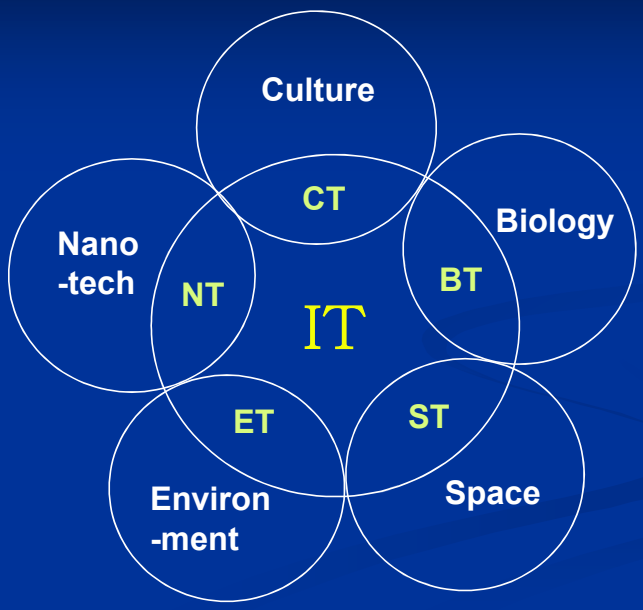
- Ubiquitous computing
  - Wireless networking
  - Distributed computing
- Web-application S/W
  - Simplest form
  - Sophisticated computing environments
  - Standalone features, Computing functions, and content to the end user
  - Database and business applications
- Open source
  - E.g., operating systems, database, and developing environment
  - So that customers can make local modifications
  - Self-descriptive
- The “new economy”
  - The challenge for software engineers is to build applications that will facilitate mass communication and mass product distribution using concept that are only now forming.

소프트웨어 본질의 변화

- Web Apps
  - 웹 기반 시스템과 응용프로그램
- Mobile Apps
  - 모바일 플랫폼
    ➔ iOS, 안드로이드, 윈도우 모바일
  - 모바일 웹 응용프로그램
- Cloud Computing
  - Front-end Service ➔ 클라이언트 디바이스용 응용 SW
  - Back-end Service ➔ 서버 관련 컴퓨팅 자원, 저장 및 관리, 미들웨어 포함 SW



6대 융합 분야: 6T

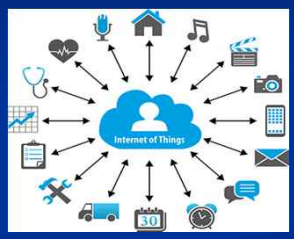


4차 산업혁명 신기술 분야

무인자율차

인공지능

IOT



드론

VR

소프트웨어공학

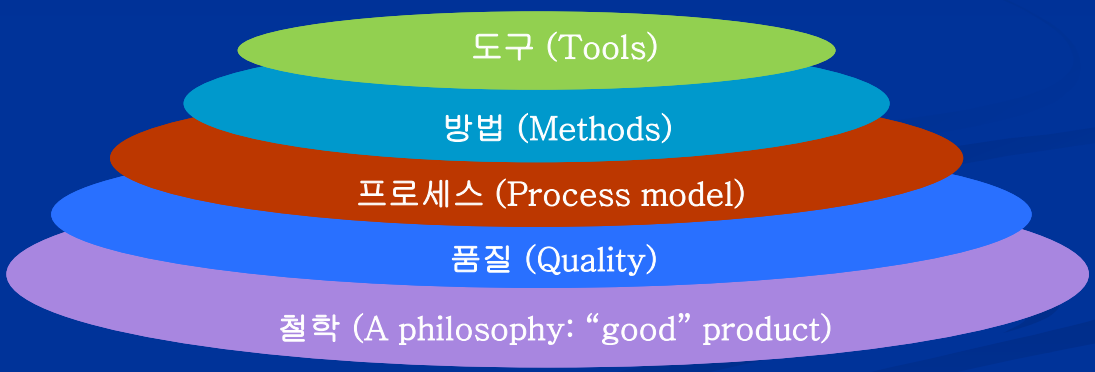
개요

소프트웨어공학 정의

- [Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines

- Software engineering;
  1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
  2) The study of approaches as in (1)

소프트웨어공학
– 단계별 접근



소프트웨어공학
– 단계적 접근

- Philosophy
- Quality
  - Bedrock
- Process
  - Framework
  - Effective delivery of software engineering technology
  - Management control of software projects
  - Establishes the context in which technical methods
  - Work products (models, documents, data, reports, forms, etc.)
  - Milestones are established
  - Quality is ensured
  - Change is properly managed
- Methods
  - "how to's"
- Tools
  - Automates or semi-automated support for the process and the methods
  - Computer-aided software engineering – CASE tool

프레임워크 활동

보호 활동

소프트웨어 프로세스:
프레임워크 활동 (Framework Activities)

- 대화(Communication)
- 계획수립(Planning)
- 모델링(Modeling)
  - Analysis of requirements
  - Design
- 구축(Construction)
  - Code generation
  - Testing
- 설치(Deployment)

소프트웨어 프로세스: 프레임워크 활동 (Framework Activities)

- 대화(Communication)
  - This framework activity involves heavy communication and collaboration with the customer (and other stakeholders) and encompasses requirements gathering and other related activities.
- 계획수립(Planning)
  - This activity establishes a plan for the software engineering work that follows. it describes the technical tasks to be conducted, the risk that are likely, the resources that will be required, the work products to be produced, and a work schedule.
- 모델링(Modeling)
  - This activity encompasses the creation of models that allow the developer and the customer to better understand software requirements and the design that will achieve those requirements.
- 구축(Construction)
  - This activity combines code generation (either manual or automated) and the testing that is required the uncover errors in the code.
- 설치(Deployment)
  - The software (as a complete entity or as a partially completed increment) is delivered to the customer who evaluates the delivered product and provides feedback cased on the evaluation.

소프트웨어 프로세스: 보호 활동 (Umbrella Activities)

- 통제 (Software project tracking & control)
- 위험관리 (Risk management)
- 품질보증 (Software quality assurance)
- 기술검토 (Formal technical reviews)
- 측정 (Measurement)
- 형상관리 (Software configuration management)
- 재사용 (Reusability management)
- 산출물 (Work product preparation & production)

보호 활동(1/2)

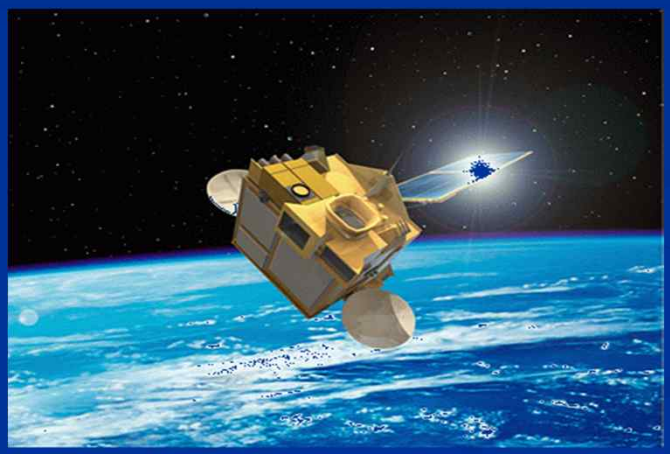
- 통제 (Software project tracking and control)
  - Allows the software team to assess progress against the project plan and take necessary action to maintain schedule.
- 위험관리 (Risk management)
  - Assesses risks that may effect the outcome of the project or the quality of the product.
- 품질보증 (Software quality assurance)
  - Defines and conducts the activities required to ensure software quality.
- 기술검토 (Formal technical reviews)
  - Assesses software engineering work products in an effort to uncover and remove errors before they are propagated to the next action or activity.

보호 활동(2/2)

- 측정 (Measurement)
  - Define and collects process, project, and product measures that assist the team in delivering software that meets customers' needs; can be used in conjunction with all other framework and umbrella activities.
- 형상관리 (Software configuration management)
  - Manages the effects of change throughout the software process.
- 재사용 (Reusability management)
  - Define criteria for work product reuse (including software components) and establishes mechanisms to achieve reusable components.
- 산출물 (Work product preparation and production)
  - Encompasses the activities required to create work products such as models, documents, logs, forms, and lists.
  - Umbrella activities are applied throughout the software process and are discussed in detail later in this book.

천리안 인공위성 개발 예

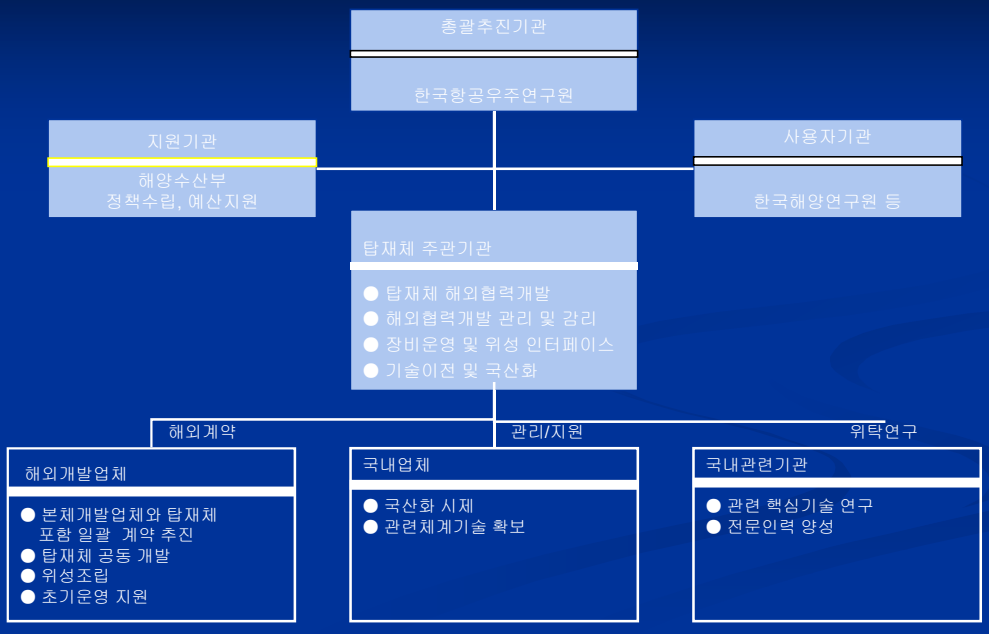
정지궤도용 통신해양기상위성 개발사업



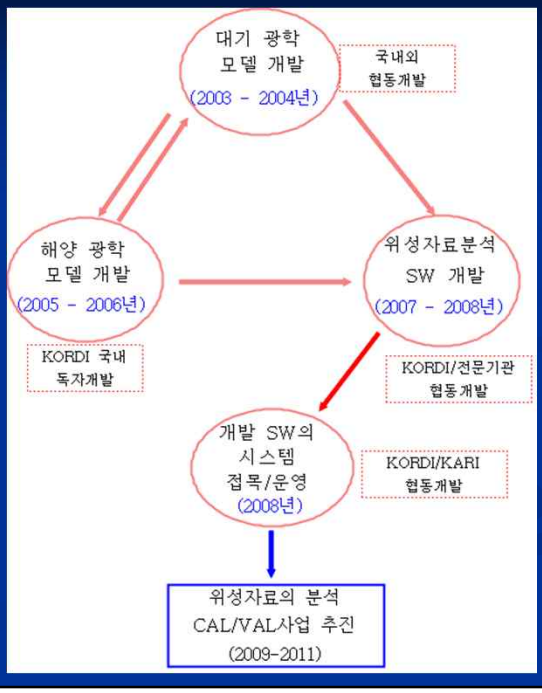
○ 사업기간 : **03 – 08**

○ 총사업비 : **2,880억원**

사업 추진 체계



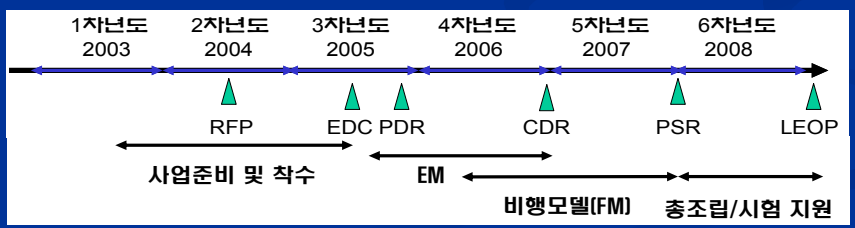
해양위성 자료처리 시스템의 연구개발 추진 체계도



인공위성 개발 일정 예

- **연구개발 진행 현황 및 일정**

구분	연구개발목표
1차년도 ('03.9 - '04.4)	해양탑재체 요구사항수립 및 해외협력개발기관 평가
2차년도 ('04.5 - '05.4)	해외협력개발기관 선정/ 해양탑재체 시스템 설계 ('05. 7 GOCI KO 개최)
3차년도 ('05.5 - '06.4)	해양탑재체 예비설계 ('06. 1. GOCI SDR 개최)
4차년도 ('06.5 - '07.2)	해양탑재체 상세설계
5차년도 ('07.3 - '08.2)	해양탑재체 총조립/시험 및 납품
6차년도 ('08.3 - '08.12)	해양탑재체 위성체와의 총조립/시험, 발사 및 초기운영지원



※EDC : 계약발효일 PDR : 예비설계검토회 CDR : 최종설계검토회
PSR : 납품전검토회 LEOP : 발사 및 초기운영

소프트웨어공학 실무 (1/5)

< 문제해결 방법론 >

1. *문제 이해* (대화와 분석)
2. *해결방안 계획* (모델링과 소프트웨어 설계)
3. *실행* (코드 생성)
4. *검토* (테스트와 품질보증)

소프트웨어공학 실무 (2/5)

< 문제해결 방법론 >

- *Understand the problem* (communication and analysis)
  - Who has a stake in the solution to the problem? That is, who are the stakeholders?
  - What are the unknowns? What data, functions, features and behavior 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?

소프트웨어공학 실무 (3/5)
< 문제해결 방법론 >

- *Plan a solution (modeling and software design)*
  - 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, features, and behavior that are required?
  - Has a similar problem been solved? If so, are elements of the solution reusable?
  - Can sub-problems be defined? If so, are solutions readily apparent for the sub-problems?
  - Can you represent a solution in a manner that leads to effective implementation? Can a design model be created?

소프트웨어공학 실무 (4/5)
< 문제해결 방법론 >

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

소프트웨어공학 실무 (5/5)
< 문제해결 방법론 >

- *Examine the result for accuracy (testing and quality assurance)*
  - 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, features, and behavior that are required? Has the software been validated against all stakeholder requriements?

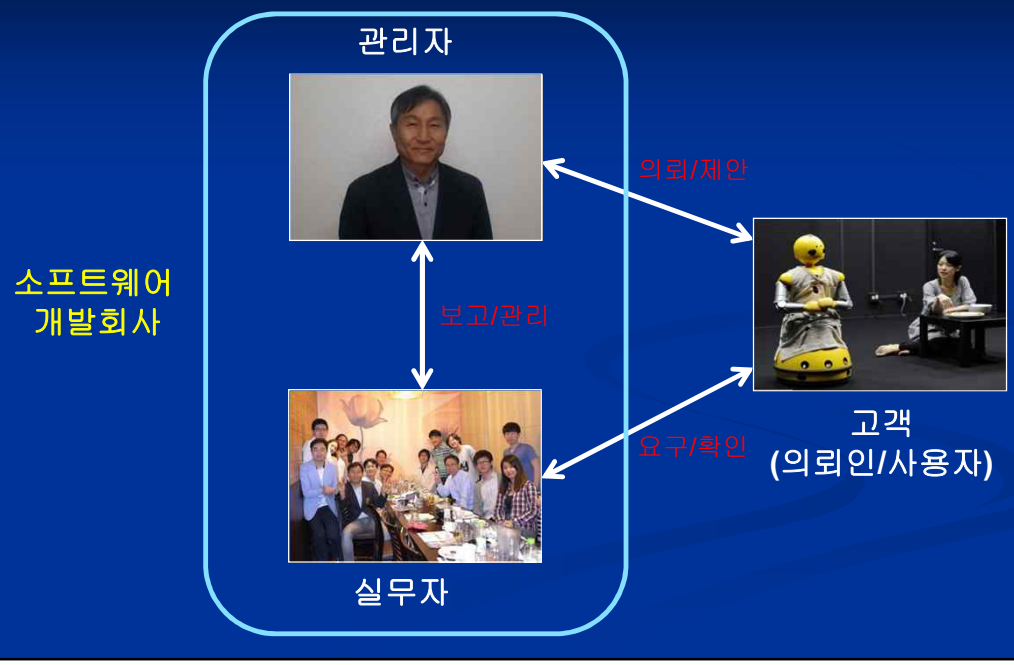
일반 원칙 (1/2)

- 제1원칙: The Reason It All Exists
  - To provide value to its users
- 제2원칙: KISS (Keep It Simple, Stupid!)
  - All design should be as simple as possible, but no simpler.
  - This is not to say that features, even internal features, should be discarded in the name of simplicity.
- 제3원칙: Maintain the Vision
  - A clear vision is essential to the success of a software project.
- 제4원칙: What You Produce, Others Will Consume
  - Always specify, design, and implement knowing someone else will have to understand what you are doing.

일반 원칙 (2/2)

- 제5원칙: Be Open to the Future
  - Never design yourself into a corner
  - Solve the general problem, not just the specific one
- 제6원칙: Plan Ahead for Reuse
  - Planning ahead for reuse reduces the cost and increases the value of both the reusable components and the systems into which they incorporated.
- 제7원칙: Think!
  - Placing clear, complete thought before action almost always produces better results.

소프트웨어에 대한 편견



소프트웨어에 대한 편견

- 관리자의 편견
  - Myth: Standards, Procedures
  - Reality: No!
  - Myth: If behind schedule, then more programmers and catch up
  - Reality: Software developments is not a mechanistic process like manufacturing
  - Myth: Outsource
  - Reality: Invariably struggle

소프트웨어에 대한 편견

- 고객의 편견
  - Myth: A general statement of objectives is sufficient
  - Reality: Ambiguous statement makes disaster
  - Myth: Change can be easily accommodated
  - Reality: Impact of change; time, (if early, then small), but as time passes, cost impact grows rapidly

소프트웨어에 대한 편견

- 실무자의 편견
  - Myth: If it works, Job is done
  - Reality: Sooner you begin writing code, the longer it'll take you to get done
  - Myth: No way of assessing its quality
  - Reality: Formal technical review

소프트웨어에 대한 편견

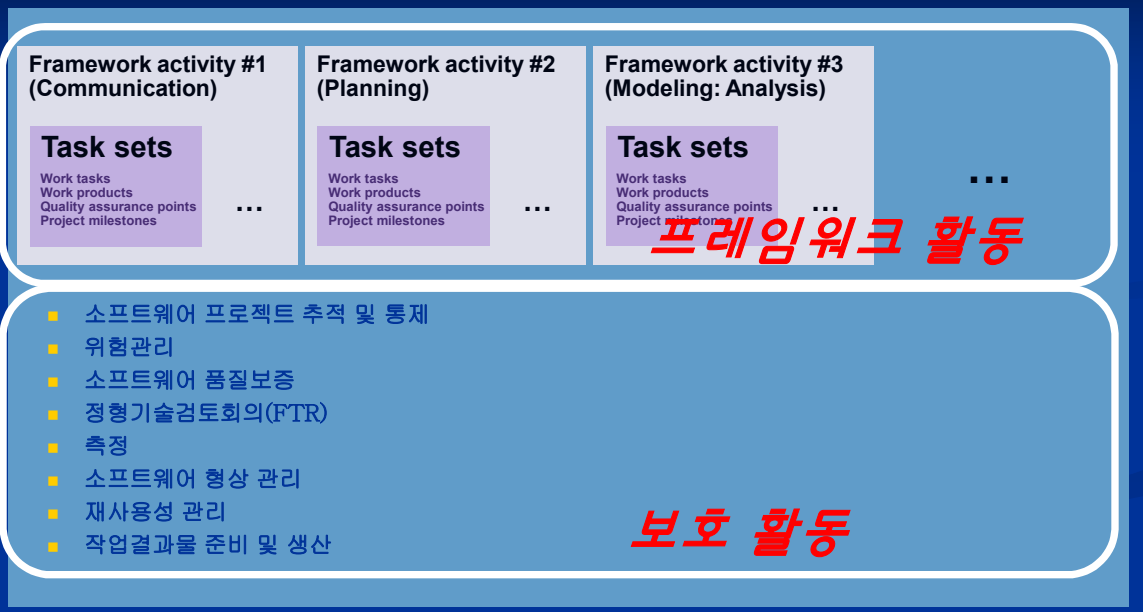
- 실무자의 편견
  - Myth: Only deliverable work is working program
  - Reality: Software configuration
  - Myth: Creating voluminous and unnecessary documentation will invariably slow us down
  - Reality: Creating quality, reduce rework, faster delivery times

프로세스

프로세스 프레임워크

- 프레임워크 활동
  - 작업 태스크
  - 작업 산출물
  - 이정표 및 부산물
  - 품질보증 점검 항목
- 보호 활동

프로세스 프레임워크



프레임워크 활동
(Framework Activities)

- 대화
- 계획
- 모델링
  - 요구사항 분석
  - 설계
- 구현
  - 코드 생성
  - 테스팅
- 배치

보호 활동 (Umbrella Activities)

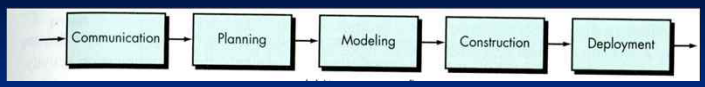
- 소프트웨어 프로젝트 추적 및 통제
- 위험 관리
- 소프트웨어 품질 보증
- 정형기술검토 (Formal technical reviews)
- 측정
- 소프트웨어 형상 관리
- 재사용성 관리
- 작업 산출물 준비 및 생산

프로세스 모델: 딜레마

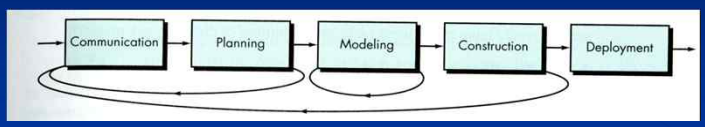
- If prescriptive process models strive for structure and order, are they inappropriate for a software world that thrives on change?
- Yet, if we reject traditional process models (and the order they imply) and replace them with something less structured, do we make it impossible to achieve coordination and coherence in software work?

프로세스 흐름

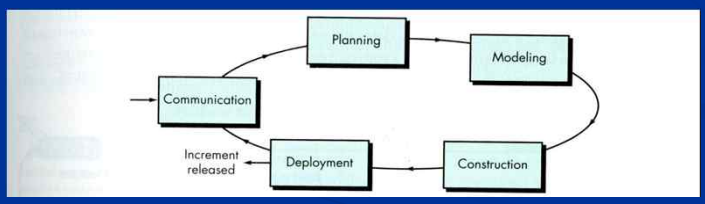
- Linear Process Flow



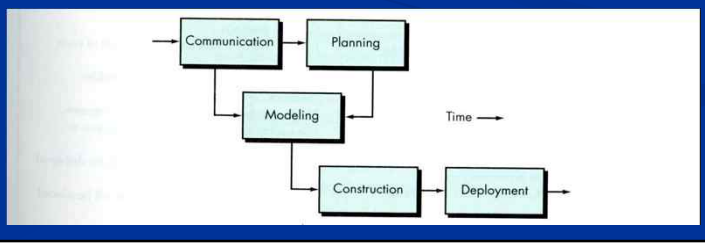
- Iterative Process Flow



- Evolutionary Process Flow

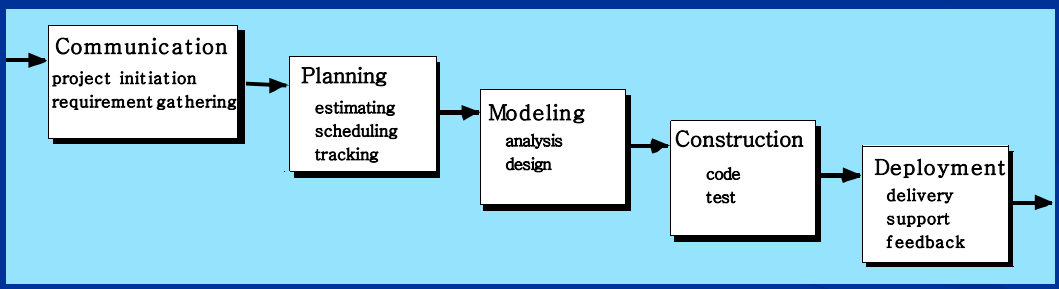


- Parallel Process Flow

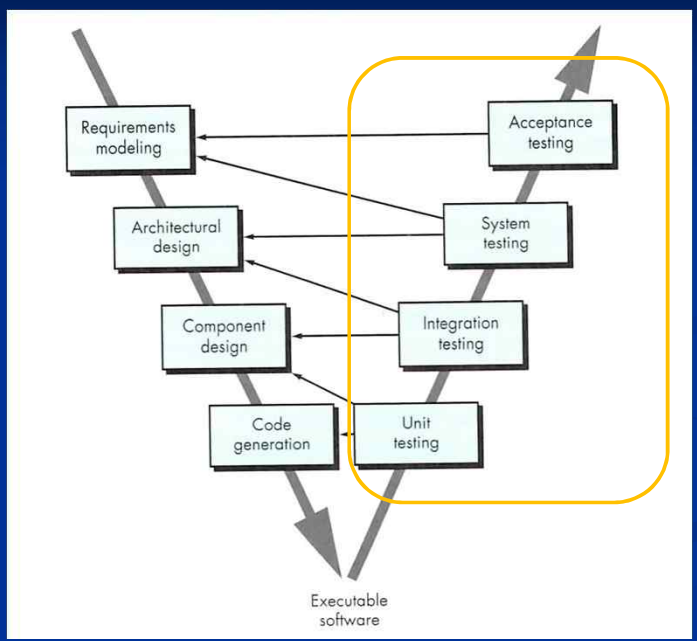


프로세스 방법론

폭포수 모델 (1/3)



폭포수 모델 (2/3)



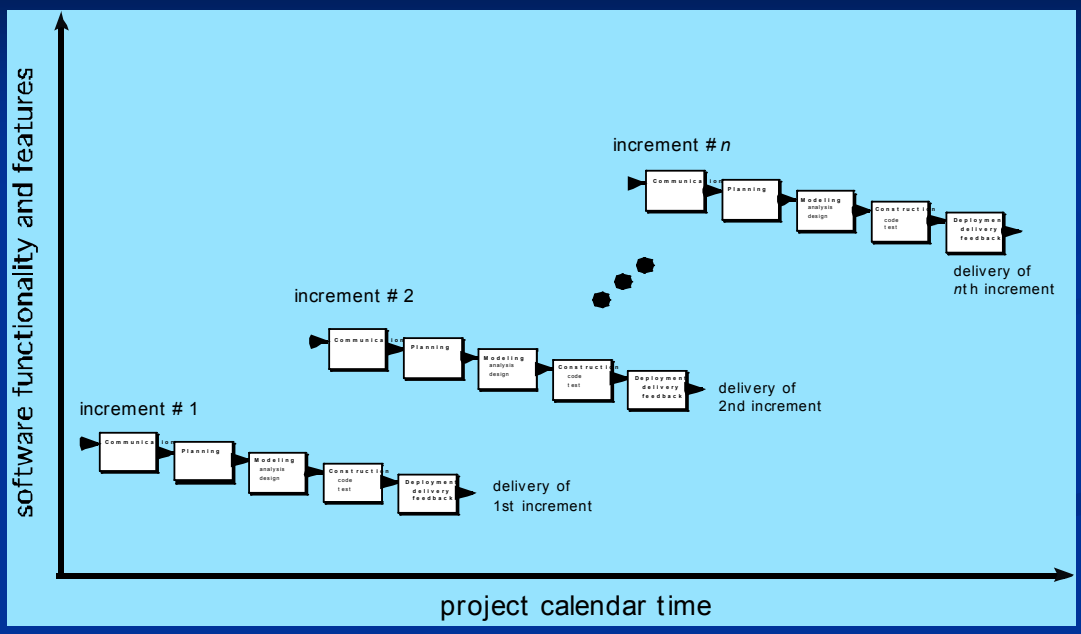
"V 모델"

➔ Verification &Validation 강조

폭포수 모델 (3/3)

- 폭포수 모델 – 고전적 생명주기(classic life cycle)
  - Systematic
  - Sequential approach
  - Oldest paradigm
  - Problems;
    1. Real projects rarely follow the sequential flow
    2. Difficult to state all requirements explicitly
    3. A working version will not be available until late
- It can serve as a useful process model in situations where requirements are fixed and work is to proceed to completion in a linear manner.

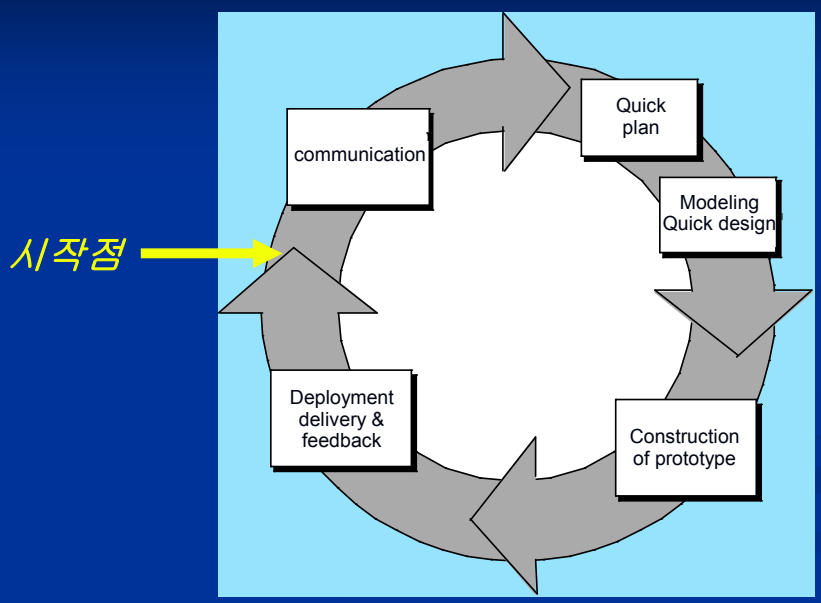
점증적 모델 (1/2)

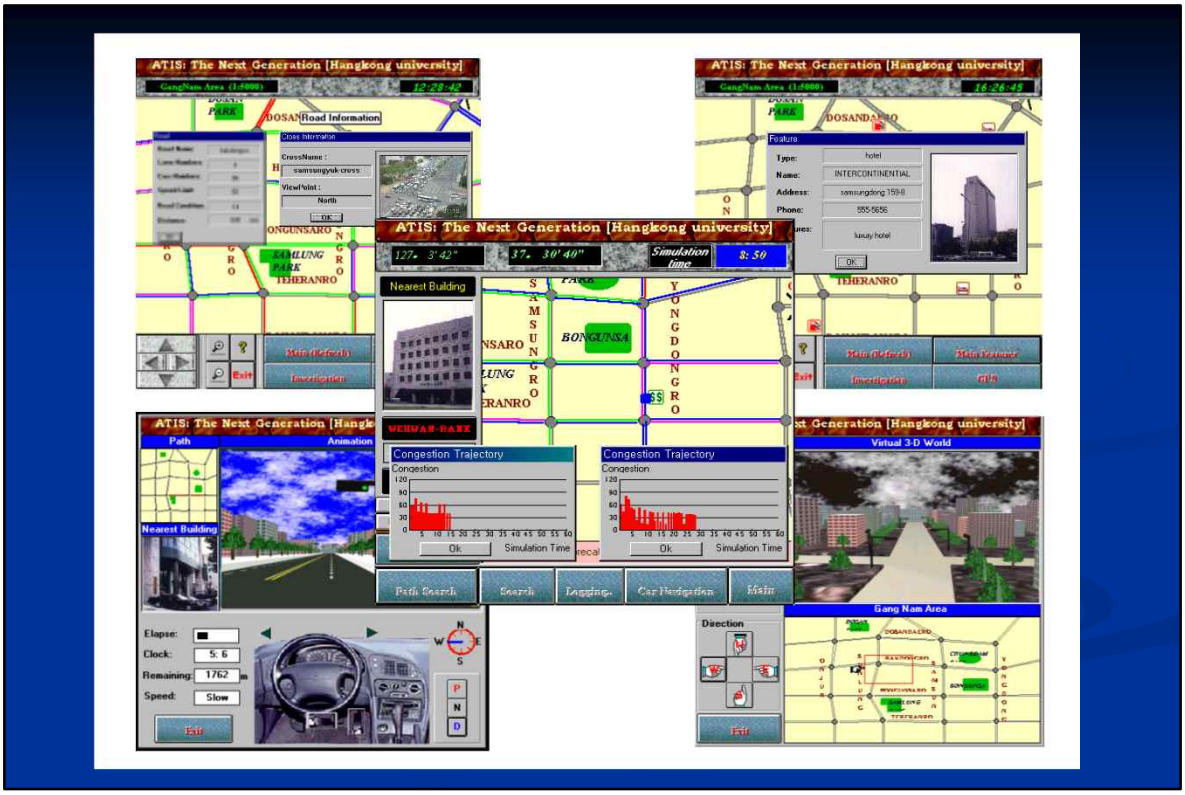


점증적 모델 (2/2)

- Combines elements of the waterfall model applied in an iterative fashion
- Core product – The first increment
- Unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment.
- Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project

프로토타입 모델 (1/3)




ATIS: The Next Generation [Hangkong university]
Road Information
Cross Information
Feature
Nearest Building
Congestion Trajectory
Virtual 3-D World
Gang Nam Area
Path Search
Search
Logging
Car Navigation
Main
Exit


I3D2 - Traffic Simulation System
Specification
Road component
Road name
Road ID
Number of lane
Road length
Cross policy
Cross Name
Cross ID
Help
Close
Length of latency of links
Total Number of running vehicles
Average Speed of links
Time-Space Diagram
Cancel

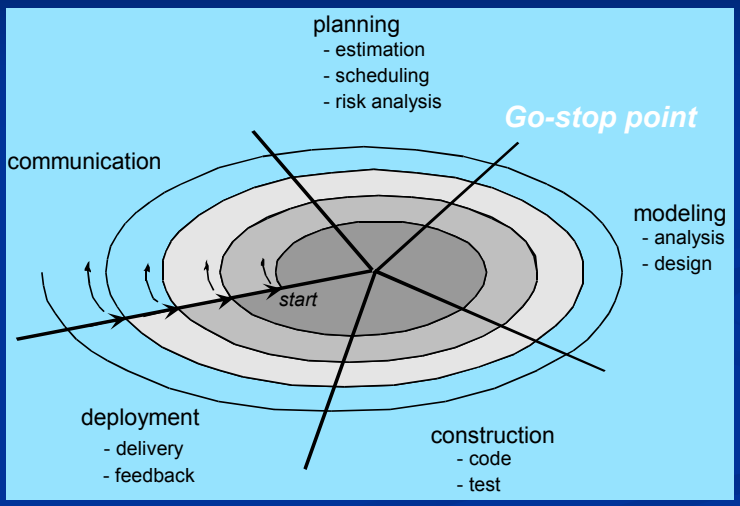
프로토타입 모델 (2/3)

- Prototyping
  - Human-machine interaction
  - Quick design
  - Ideally, the prototype serves as a mechanism for identifying software requirements.
  - "Throw away!!!"
  - The first system that we throw away

프로토타입 모델 (3/3)

- Problems;
  1. Sees what appears to be a working version
     - haven't considered overall software quality or long-term maintainability
  2. Implementation compromises in order to get a prototype work quickly
     - An inappropriate operation system or programming language
     - An inefficient algorithm
- Although problems, effective paradigm

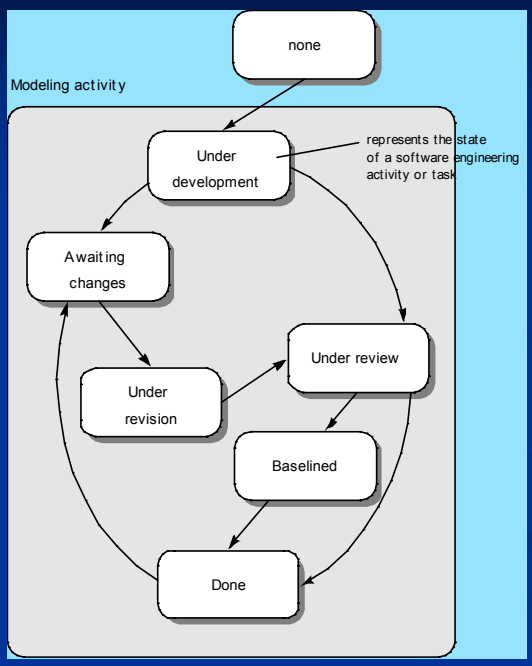
나선형 모델 (1/2)



나선형 모델 (2/2)

- Couple the iterative nature of 프로토타입 모델 with the controlled and systematic aspects of the 폭포수 모델.
  - A risk-driven process model
- During early iterations, it might be a paper model or prototype.
- During later iterations, increasingly more complete versions of the engineered system are produced.
- The spiral model is a realistic approach to the development of larger-scale systems and software.
- Risk
  - It demands considerable risk assessment expertise and relies on this expertise for success.
  - If a major risk is not uncovered and managed, problems will undoubtedly occur.

동시성 모델 (1/2)



동시성 모델 (2/2)

- Emphasizes an activity network
- Called ‘concurrent engineering’
- Schematically as a series of framework activities, software engineering actions and tasks, and their associated states.
- The concurrent process model defines a series of events that will trigger transition from state to state for each of the software engineering activities, actions, or tasks.
- The concurrent process model is applicable to all types of software development and provides an accurate picture of the current state of project.

진화적 모델들 정리
- 프로토타입, 나선형, 동시성

- A final comment on evolutionary models
  - Time-to-market is the most important management requirement.
  - Problems;
    - A problem to project planning because of the uncertain number of cycles
    - If the evolutions occur too fast, process will fall into chaos. If the speed is too slow then productivity could be affected.
    - focused on flexibility and extensibility rather than on high quality.

컴포넌트 기반 모델

- The spiral + object-oriented
- Prepackaged software components
- Steps;
  - Available component-based products are researched and evaluated for the application domain in question.
  - Component integration issues are considered.
  - A software architecture is designed to accommodate the components.
  - Components are integrated into the architecture.
  - Comprehensive testing is conducted to ensure proper functionality
- Component-based development model leads to software reuse
- Reusability provides software engineers with a number of measurable benefits.
- Component-based development leads to a 70% reduction in development cycle time; a 84% reduction in project cost;
- A productivity index of 26.2, compared to an industry norm of 16.9

정형 방법론 모델

- Leads to formal mathematical specification of computer software
- Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.
- 'Clean-room software engineering'
- Defect-free software
- Problems;
  - Quite time-consuming and expensive
  - Extensive training
  - Difficult to use a communication mechanism
  - Nevertheless, Safety-critical software (e.g., aircraft avionics and medical devices)

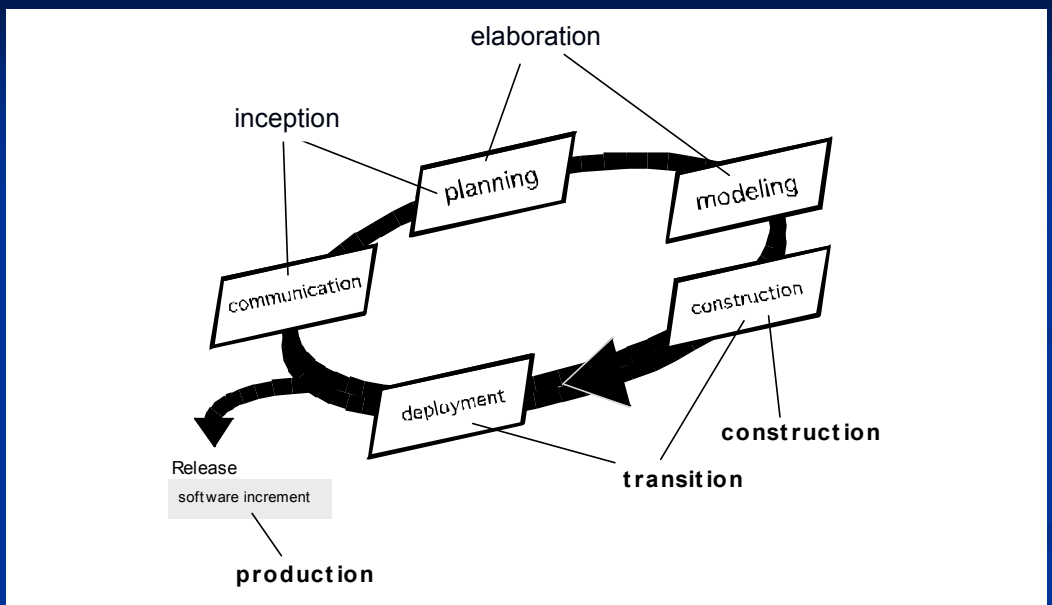
관점 지향 모델

- 최근의 복잡 정교화의 경향은 고객의 다양한 관심사(aspect)에 따라 시스템 수준, 특성, 기능 등에 영향을 받음.
- 진화 모델 (관점 파악에 유용) + 동시성 모델 (일부 모델과는 독립적)
- AOSD: Aspect-oriented SW Development
- AOP: Aspect-oriented Programming
- AOCE: Aspect-oriented Component Engineering

통합 프로세스 모델
(UP: Unified Process)

- "유스케이스 기반, 구조중심, 반복적이고, 점증적인"
- A brief History
  - James Rumbaugh, Grady Booch, and Ivar Jacobson
  - "Unified method"
    - That would combined the best features of each of their individual methods and adopt additional feature proposed by other experts in the OO field.
  - UML – a unified modeling language
  - Unified process
    - a frame work for OO software engineering using UML.

통합 프로세스 모델



통합 프로세스 모델

- 도입 단계(Inception phase)
  - encompasses both customer communication and planning activities
  - By collaboration with the customer and end-user, business requirements for the software are identified, a rough architecture for the system is proposed, and a plan for the iterative, incremental nature of the ensuing project is developed.
  - Preliminary use-cases

통합 프로세스 모델

- 구체화 단계(Elaboration phase)
  - Encompasses the planning and modeling activities of the generic process model.
  - Elaboration refines and expands the preliminary use-case that was developed as part of the inception phase and expands the architecture representation to include five different views of the software
    - the use-case model
    - the analysis model
    - the design model
    - the implementation model
    - the deployment model
  - elaboration creates an “executable architectural baseline” that represents a “first cut” executable system.

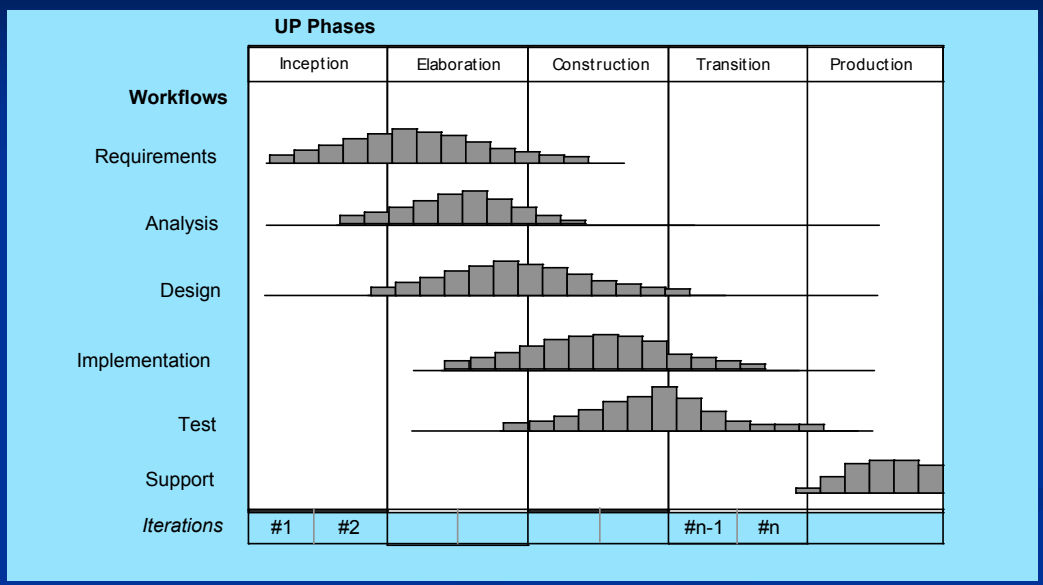
통합 프로세스 모델

- 구축 단계(Construction phase)
  - the construction activity
  - Using the architectural model as input, the construction phase develops or acquires the software components that will make each use-case operational for end-users.
  - To accomplish this, analysis and design models that were started during the elaboration phase are completed to reflect the final version of the software increment.

통합 프로세스 모델

- 변환 단계(Transition phase)
  - First art of the generic deployment activity
  - To end-users for beta testing
- 생산 단계(Production phase)
  - Deployment activity
  - The on-going use of the software is monitored, support for the operation environment (infrastructure) is provided, and defect reports and requests for changes are submitted and evaluated.

UP 단계



UP 작업 산출물

Inception phase

- Vision document
- Initial use-case model
- Initial project glossary
- Initial business case
- Initial risk assessment
- Project plan phases and iteration
- Business model one or more prototypes

Elaboration phase

- Use-case model
- Supplementary requirements including non-functional
- Analysis model
- Software architecture description
- Executable architectural prototype
- Preliminary design model
- Revised risk list
- Project plan including iteration plan adapted workflows milestones technical work products
- Preliminary user manual

Construction phase

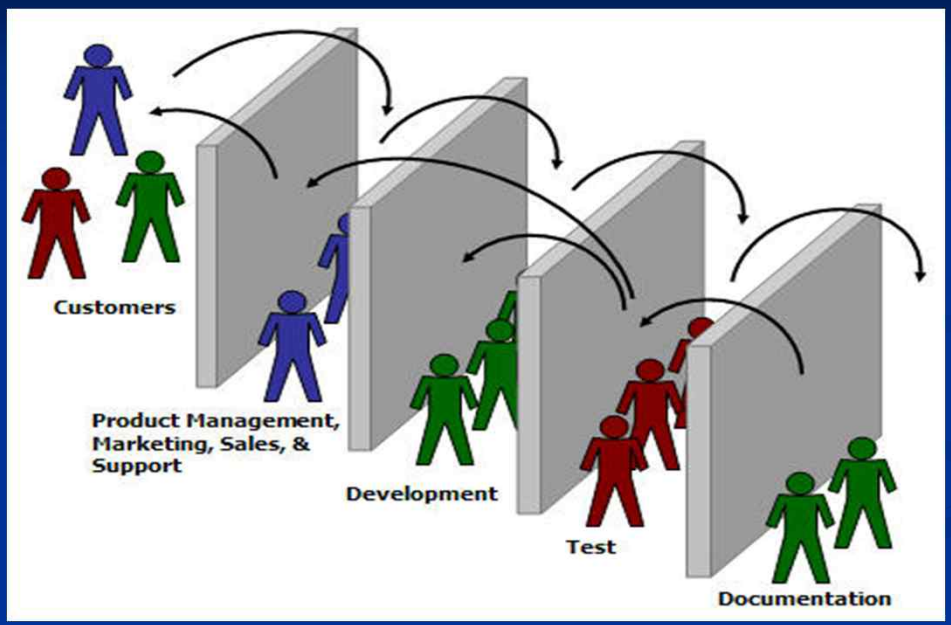
- Design model
- Software components
- Integrated software increment
- Test plan and procedure
- Test cases
- Support documentation user manuals installation manuals description of current increment

Transition phase

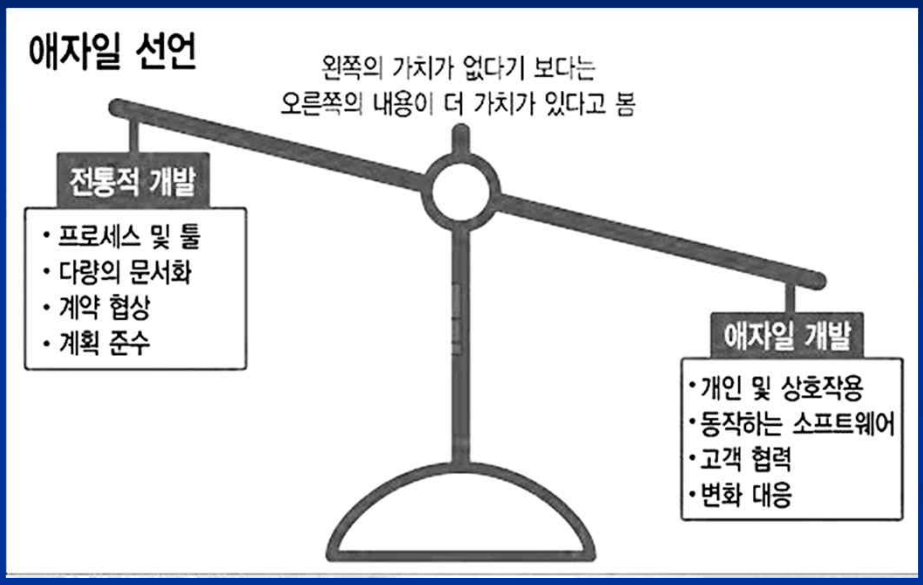
- Delivered software increment
- Beta test report
- General user feedback

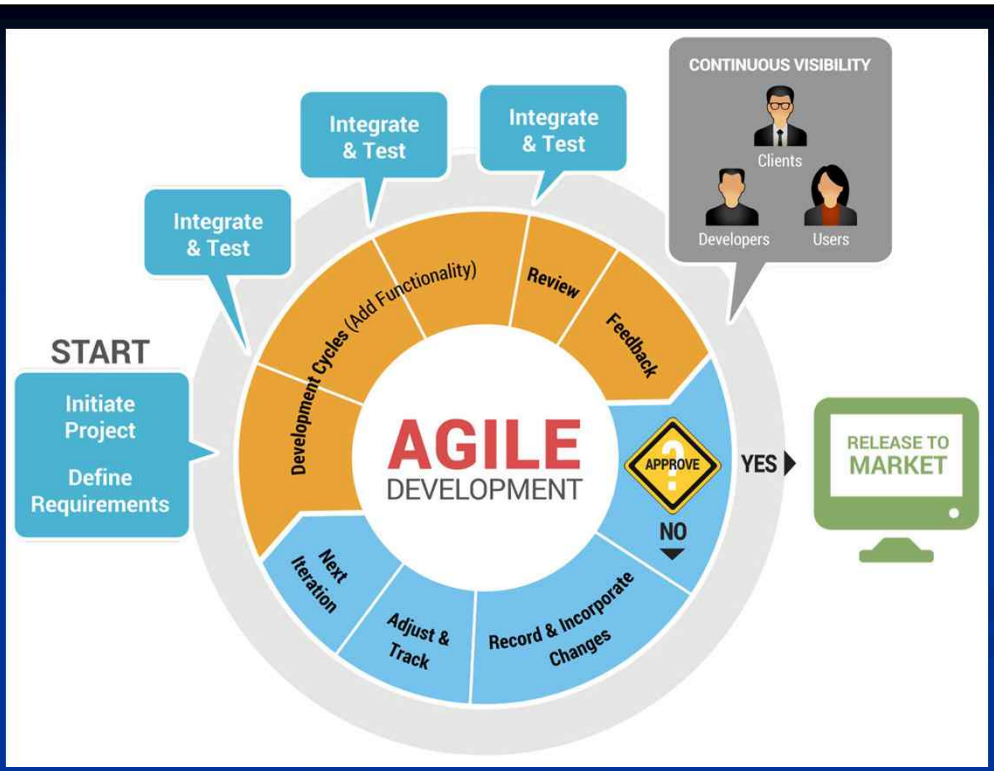
애자일
개발 프로세스

전형적인 개발 프로세스

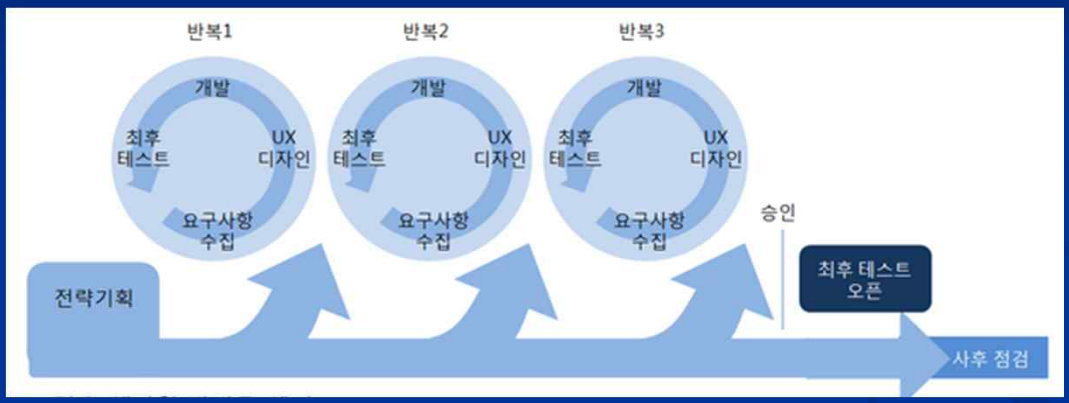


전형적 방법 vs. 애자일 방법



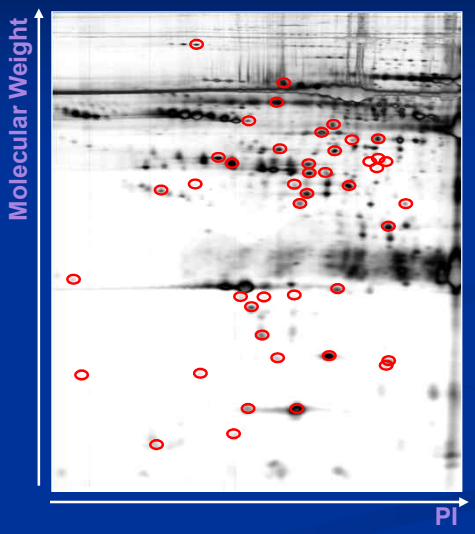


애자일 방법론



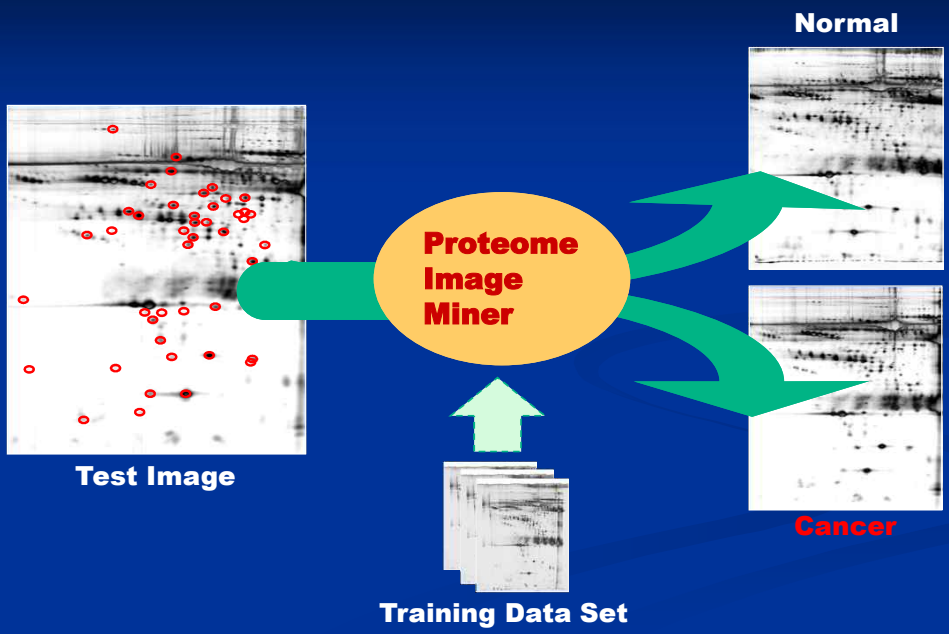
***UX (User** Experience)**:** 사용자가 무언가에 대해 경험하고 느끼는 감정의 총체

혈청 프로테옴 이미지



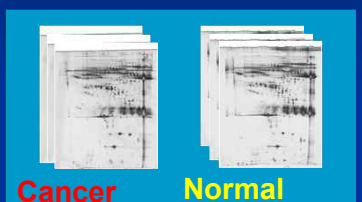
< 2D Gel Proteome Image >

프로테옴 이미지 마이닝

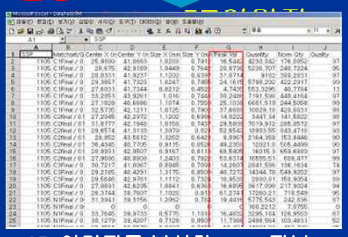


유방암혈청을 이용한 선행연구 결과

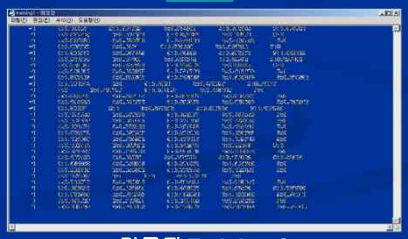
- SVM/GA를 이용한 유방암 진단 – Training 단계



기계학습 / 진화 알고리즘 적용



2D 이미지를 분석한 spot 정보

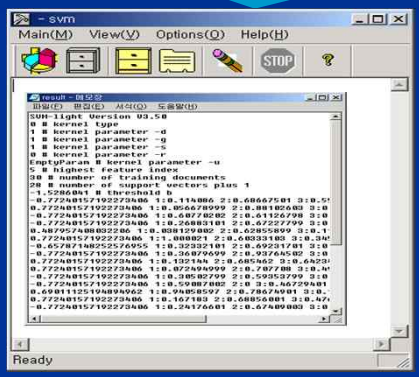


가공된 Training Set

유방암혈청을 이용한 연구 결과

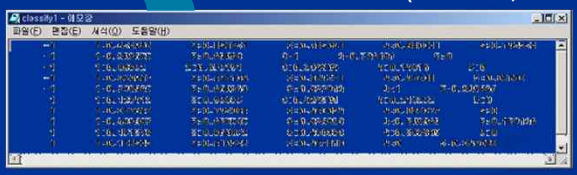
- SVM/GA를 이용한 유방암 진단

Test



Decision Model

검진 대상자 Test Set (cancer 33, normal 35)



판정결과

- 민감도 → 100%
- 특이도 → 88.57%
- 분류 정확도 → 94.11%