소프트웨어공학 중간고사 정리

1단원

(1) Software Engineering

- Software : computer programs and associated documentation.
- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
- engineering discipline : using appropriate theories and methods to slove problems bearing in mind organizational and financial constraints.
- All aspects software production : Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.
- we need to be able to produce reliable and trustworthy systems economically and quickly.
- Software engineering vs computer science : computer sciences focuses on theroy and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
- Software Engineering vs Sytem engineering : System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering; Software engineering is part of this more general process.

(2) Software engineering activities : specification → development → validation → evolution

- Software Specification, where customers and engineers define the software that is to be produced and the constraints on its operation.

→ development constraint : 개발하는 동안의 비용, 시간, 품질 사용가능한 인력

→ operation constrain : 만들어진 시스템이 기존의 시스템과 충돌이 없는지

- Software Development, where the software is designed and programmed.
- Software Validation, where the software is checked to ensure that is what the customer requires by testing.

→ Verification : product를 만들기까지의 문서에 충족하는가 by review, code inspection

- Software Evolution, where the software is modified to reflect changing customer and market requirements.

(3) Software Costs

- Software costs often dominate computer system costs/quality/time to market.
- Software costs more to maintain than it does to develop especially long life system.
- 60% of software costs are development costs, 40% are testing costs. For custom software, eveolution costs often exceed devlopment costs.
- The majority of costs are the costs of changing the software after it has gone into use. So, to use software engineering methods and techniques for software systems is cheaper.

(4) Software Products

: implementation 과정에선 공통이지만 design, test 방식에서 차이 가짐

1) Generic Product

- Stand-alone systems that are marketed and sold to any customer who wishes to buy them
- The specification of what the software should do is owned by the sortware developer and decisions on software change are made by the developer.

ex) pc software such as graphics programs, project management tools(CAD),
software for specific markets(appointments system for dentists)

2) Customized Products

- Software tha is commissioned by a specific customer to meet their own needs.
- The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.

ex) embedded control systems, air traffic control software, traffic monitoring systems.

(5) Essential Attributes of Good Software

Maintainablility	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is ciritical attribute because software change is an inevitable requriement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system. • security and dependability considerations - Design options limited by procurement decisions - Human errors made during development may introduce faults into the system. - Inadequate testing may mean faults are not discovered before deployment. - Configuration errors during deployment may introduce vulnerabilities. - Assumptions made during procurement may be forgotten when system changes are made.
Efficienty	Software should not make wasteful use of system resources such as memeory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization.
Acceptability	Software must be acceptable to the type of users for which it is designed. this means that it must bu understandable, usable and compatible with other systems that they use.

(6) Software Engineering Diversity

- The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.
- There are many different types of software system and there is no universal set of software techniques that is applicable to all of these.

(7) Software Engineering Fundamentals

: Some fundatmental principles apply to all types of software system, irrespective of the development techniuqes used;

- systems should be developed using a managed and understood development process. Of course, different processes are used for different types os software.
- dependability and performance are important for all types of system.
- understanding and managing the software specification and requirements(what the software should do) are important.
- where appropriate, you should reuse software that has already been developed rather than write new software.

(8) Web Software Engineering

- The Web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems.
- Cloud computing is an approach to the provision of computer services where applications run remotely on the 'cloud'.
- Software reuse is the dominant approach for constructing web-based systems.

→ when building these systems, you think about how you can assemble them form pre-exisiting software ocmponents and systems

- Web-based systems should be developed and delivered incrementally.

(9) Software Engineering Ethics

- Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.

2단원 1. Sociotechnical Systems

(1) Systems
- System : purposeful collection of inter-related components working together to achieve some common objective.
- Software System is components of a broader systems (that have a human, social or organizational purpose engineering) process.

(2) Sociotechnical Systems Stack (7 layers in the STS stack)

software engineering

Society	Laws, regulation and culture that affect the operation of the system.
Organization	Higher level strategic business activities that affect the operation of the system such as business rules, norms that should be followed.
Business Processes	A set of processes involving people and computer systems that support the activities of the business.
Application System	Specific functionality to meet some organization requirements.
Communications and data management	Middleware that provides access to remote systems and databases.
Operating system	Provides a set of common facilities for higher levels in the system.
Equipment	- Hardware devices. - Most devices will included an embedded system of some kind.

system engineering

(3) Holistic system design
- There are insteractions and dependencies between the layers in a system and changes at one level ripple through the other levels
- For dependability, a systems perspective is essential.
- System is a purposeful collection of inter-related components working together to achieve some common objective.
- The properties and behaviour of system components are inextricably inter-mingled. This leads to complexity.

(4) System Categories
1) Technical computer-based systems
- Systems that include hardware and software but where the operators and operational processes are not normally considered to be part of the system.

2) Socio-technical systems
- Systems that include technical systems but also operational processes and people who use and interact with the technical system.
- Socio-technical systems are governed by organizational policies and rules.
- Socio-technical system characteristics : Emergent properties, Non-deterministic,
  Complex relationships with organizational objectives

(5) Organizational affects

- Human and organizational factors, such as the organizational structure, have a significant effect on the operation of socio-technical systems.
- Process Changes : Systems may require changes to business processes so training may be required. significant changes may be resisted by users.
- Job Changes : Systems may de-skill users or cause changes to the way they work. The status of individuals in an organization may be affected by the introduction of a new system.
- Organizational Changes : Systems may change the political power structure in an organization. If an organization depends on a system then those that control the system have more power.

(6) Emergent properties

: Consequence of the relationships between system components

- Therefore they can only be assessed and measured once the components have been integrated into a system.
- Non-functional emergent properties : These relate to the behaviour of the system in its operational environment. They are often critical computer-based systems as failure to achieve some minimal defined level in these properties may make the system unusable.

Volume	The volume of a system (total space occupied) baries depending on how the component assemblies are arranged and connected.
Reliability	- Because of component inter-dependencies, faults can be propagated through the system. - System failures occur because of unforeseen inter-relationships between components. - It is practically impossible to anticipate all possible component relationships. So software reliability measures may give a false picture of the overall system reliability. ① Hardware reliability (bathtub curve) ② Software reliability : 신뢰성을 설명하는 중요한 특성으로 시스템을 구성하는 component가 각각 reliable한다고 해도, 시스템이 reliable하다고 이야기 할 순 없다. 따라서 component끼리 영향을 미치는 relationship을 파악해야 하지만, software/hardware/operator끼리 영향을 끼치므로 어려움 ③ Operator reliability - Failures are not independent and they propagate from one level to another.
Security	- Ability to resist attack - complex property that cannot be easily measured - Attacks may be devised that were not anticipated by the system designers and so may defeat built-in safeguards.
Safety	- 시스템이 catastrophic failure 없이 동작하는 능력
Repairability	- How easy to fix a problem with the system once it has been discovered. - It depends on being able to diagnose the problem, access the components that are faulty, and modify or replace these components
Usability	- How easy to use the system - It depends on the technical system components, its operators, and its operating environment.

- Functional Properties : these appear when all the parts of a system work together to achieve some objective

(7) Non-determinism
- Deterministic system : one where a given sequence of inputs will always produce the same sequence of outputs.
- Software systems are deterministic.
- BUT, socio-technical systems that include humans and hardwares are non-deterministic. And, system behavior is unpredictable because of frequent changes to hardware, software and data.

(8) Success criteria
- Complex systems are developed to address 'wicked problems' - problems where there cannot be a complete specification.
- Different stakeholders see the problem in different ways and each has a partial understanding of the issues affecting the system.
- Different stakeholders have their own views about whether or not a system is 'successful'.

2단원 2. System engineering

(1) System engineering
- procuring, specifying, designing, implementing, validating, deploying and maintaining socio-technical systems.
- Concerned with the services provided by the system, constraints on its construction and operation and the ways in which it is used to fulfill its purpose or purposes.

(2) System engineering stages

Equipment and software updates

① Procurement(acquisition)
: The purpose of the system is established, high-level system requirements are defined, decisions are made on how functionality is distributed and the system components are purchased.
- System procurement covers all of the activites involved in deciding what system to buy and who should supply that system.

System evolution

② Development
: The system is developed – requirements are defined in detail, the system is implemented and tested and operational processes are defined and the training courses for system users are designed.
- System Development includes requirements specification, design, construction, integration and testing.

③ Operation
: The system is deployed and put into use. changes are made as new requirements emerge. Eventually, the system is decommissioned.
- When a system is put into use, the operational processes and the system itself have to change to reflect changing business requriements.

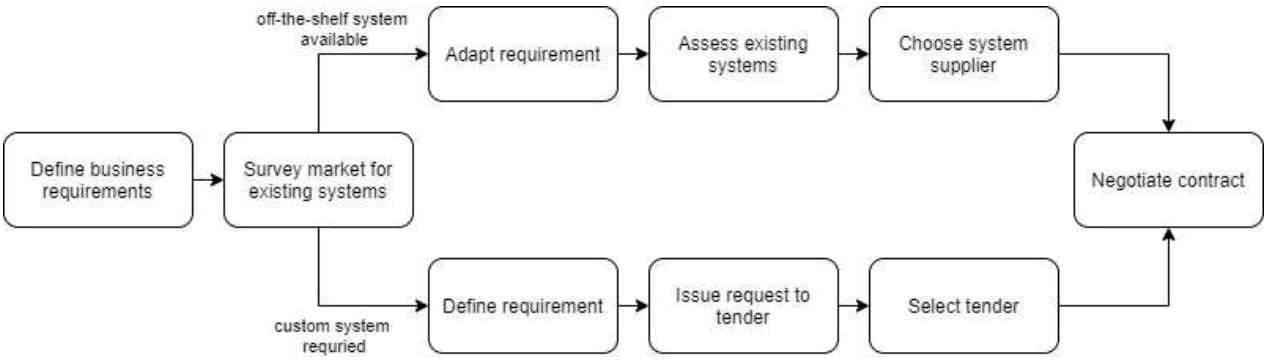
(3) Inter-disciplinary working
: communication difficulties, differing assumptions, professional boundaries

(4) System Procurement
: Acquiring a system (or systems) to meet some identified organizational need.
- Before procurement, decisions are made on
① Scope of the system, ② System budgets and timescales, ③ High-level system requirements
- Based on this information, decisions are made on whether to procure a system, the type of system and the potential system suppliers.
- Decision drivers : The state of other organizational systems, The need to comply with external regulations, External competition, Business re-organization, Available budget
- Procurement issues
① Requirements may have to be modified to match the capabilities of off-the-shelf components.
② The requirements specification may be part of the contract for the development of the system.
③ There is contract negotiation period to agree changes after the contractor to build a system has been selected.

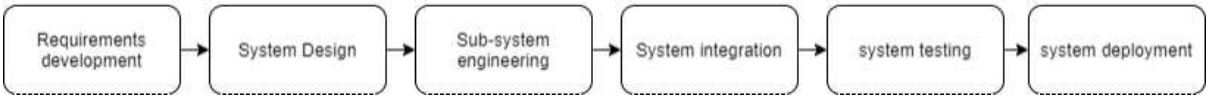
(5) Procurement and development
- Some system specification and architectural design is usually necessary before procurement.
- You need a specification to let a contract for system development.
- The specification may allow you to buy a commercial off-the shelf (COTS) system. Almost always cheaper than developing a system from scratch.
- Large complex systems usually consist of a mix of off the shelf and specially designed components. The procurement processes for these different types of component are usually different.

1) Procurement and dependability
- Procurement decisions have profound effects on system dependability as these decisions limit the scope of dependability requirements.
- For an off-the-shelf system, the procurer has very limited influence on the security and dependability requirements of the system.
- For a custom system, considerable effort has to be expended in defining security and dependability requirements.

2) system procurement processes



(6) System development



- Usually follows a plan-driven approach because of the need for parallel development of different parts of the system

1) System requirements definition

- Three types of requirement defined at this stage

① Abstract functional requirements : System functions are defined in an abstract way.

(상세한 것은 sub-system engineering 단계에서 함 + 중간에 삽입 가능)

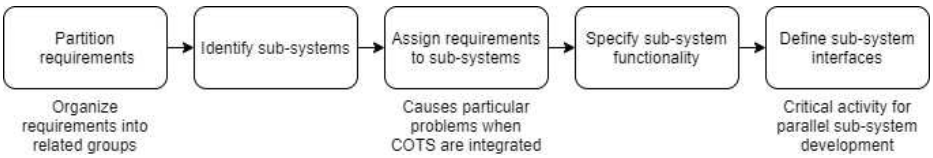
② System properties : Non-functional requirements for the system in general are defined.

(BUT. 얘는 architecture 단에서 초기에 검토 필요)

③ Undesirable characteristics : Unacceptable system behaviour is specified.

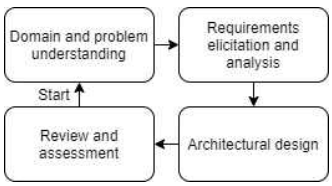
- Should also define overall organizational objectives for the system.

2) System design process



3) Requirements and design spiral

- Requirements engineering and system design are inextricably linked.
- Constraints posed by the system's environment and other systems limit design choices so the actual design to be used may be a requirement.
- Initial design may be necessary to structure the requirements.
- As you do design, you learn more about the requirements.



4) Sub-system Development

- Typically parallel projects developing the hardware, software and communications.
- May involve some COTS (Commercial Off-the-Shelf) systems procurement.
- Lack of communication across implementation teams can cause problems.
- There may be a bureaucratic and slow mechanism for proposing system changes, which means that the development schedule may be extended because of the need for rework.

5) System integration

: The process of putting hardware, software and people together to make a system.

- Should ideally be tackled incrementally so that sub-systems are integrated one at a time.
  한번에 하는 걸 big-bang 방식이라고 함
- The system is tested as it is integrated.
- Interface problems between sub-systems are usually found at this stage.
- May be problems with uncoordinated deliveries of system components.

6) System delivery and deployment

- After completion, the system has to be installed in the customer's environment
- Decisions are made on dependability and security requirements and trade-offs made between costs, schedule, performance and dependability.

① system dependability와 testing의 관계 : 비용부족으로 testing 못하면 dependability 줄음
② system dependability와 requriement의 관계
: 문서화가 잘 되어 있지 않다면 목표가 명확해지지가 않아 요구사항을 못 맞춰서 dependability 줄음
③ system dependability와 evolution의 관계
: requirement가 명확하지 않으면 candidate가 왜 그런 선택을 했는지 이유가 충분히 설명되어 있지 않아서 유지보수가 어려움.
- Human errors may lead to the introduction of faults into the system.
- Testing and validation processes may be limited because of limited budgets.
- Problems in deployment mean there is a mismatch between system & operational environment.

(7) System Operation
: the processes involved in using the system for its defined purpose.
- For new systems, these processes may have to be designed and tested and operators trained in the use of the system.
- Operational processes should be flexible to allow operators to cope with problems and periods of fluctuating workload.

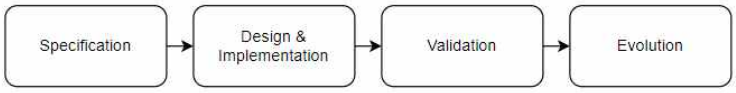
(8) Human Error
: Human errors occur in operational processes that influence the overall dependability of the system.
1) Viewing human errors:
– The person approach makes errors the responsibility of the individual and places the blame for error on the operator concerned. Actions to reduce error include threats of punishment, better training, more stringent procedures, etc.
– The systems approach is designed to detect these mistakes before they lead to system failure. When a failure occurs, the aim is not to blame an individual but to understand why the system defenses did not trap the error.

2) System Defenses
- To improve security and dependability, designers should think about the checks for human error that should be included in a system.
- There should be multiple (redundant) barriers which should be different (diverse)
- No single barrier can be perfect. There will be latent conditions in the system that may lead to failure. However, with multiple barriers, all have to fail for a system failure to occur.

(9) System Evolution (legacy systems)
1) Evolution is inherently costly
– Changes must be analysed from a technical and business perspective;
– Sub-systems interact so unanticipated problems can arise;
– There is rarely a rationale for original design decisions;
– System structure is corrupted as changes are made to it.

2) Evolution and Dependability
- Changes to a system are often a source of problems and vulnerabilities.
- Changes may be made without knowledge of previous design decisions made for security and dependability reasons. As a result, built-in safeguards may stop working.
- New faults may be introduced or latent faults exposed by changes. These may not be discovered because complete system retesting is too expensive.

3단원 – Software Process activities

(1) Software Process 

: process activities

: A structured set of activities required to develop a software system.

① specification : defining what the system should do

② design & implementation : defining the organization of the system and implementing the system

③ validation : checking that it does what the customer wants

④ evolution : changing the system in response to changing customer needs.

1) Software process model

: Abstract representation of a process

- It presents a description of a process from some particular perspective.
- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc. and the ordering of these activities.
- Process descriptions may also include:

① Products, which are the outcomes of a process activity;

② Roles, which reflect the responsibilities of the people involved in the process;

③ Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

(2) Plan-driven and Agile process

1) Plan-driven processes

: processes where all of the process activities are planned in advance and progress is measured against this plan.

- Waterfall model : separate and distinct phases of specification and development.
  (Specification(15%) → Design(25%) → Development(20%) → Intergration & Testing(40%))

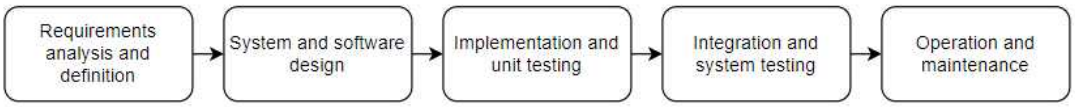
2) Agile processes

: Planning is incremental and it is easier to change the process to reflect changing customer requirements.

3) Plan-driven + Agile

- Incremental development : specification, development and validation are interleaved.
  (Specification(10%) → Iterative Development(60%) → System Testing(30%))
- Reuse-oriented software engineering : the system is assembled from existing components.
  (Component-based Model : Specification(20%) → Development(30%) → Integration&Testing(50%))

(3) Waterfall model



: separate identified phase

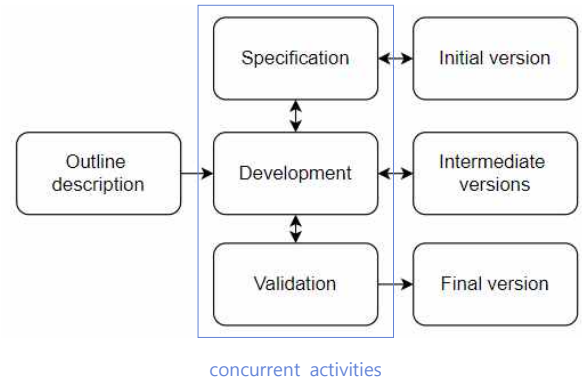
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway.
- In principle, a phase has to be complete before moving onto the next phase.

- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
- The plan-driven nature of the waterfall model helps coordinate the work. (Parallel development)
- Easy of management

1) Waterfall model problem
- Inflexible partitioning of the project into distinct stage makes it difficult to respond to changing customer requriements
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.

(4) Incremental development

1) incremental development benefits
- The cost of accommodating changing customer requirements is reduced
  : The amount of analysis and documentation is less.
- easy to get customer feedback on the development
  : Customers can comment on demonstrations of the software and see how much has been implemented.
- rapid delivery and deployment to the customer
  : Customers are able to use and gain value from the software earlier



2) incremental development problems
- The process is not visible.

: Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

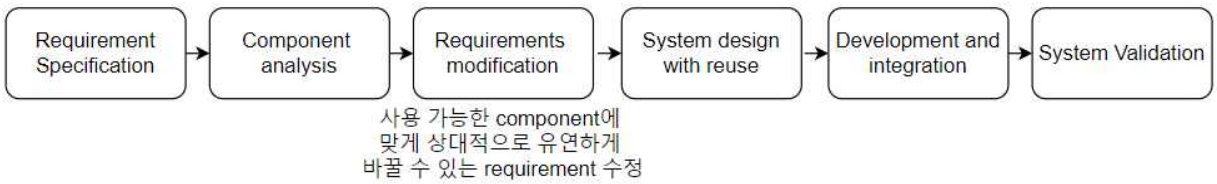
→ 문서가 없으므로 관리 어려움, evolution 때에도

- System structure tends to degrade as new increments are added.

: Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly.

→ 시스템이 어떤 시스템에 영향을 끼칠지 모르므로

(5) Reuse-oriented software engineering



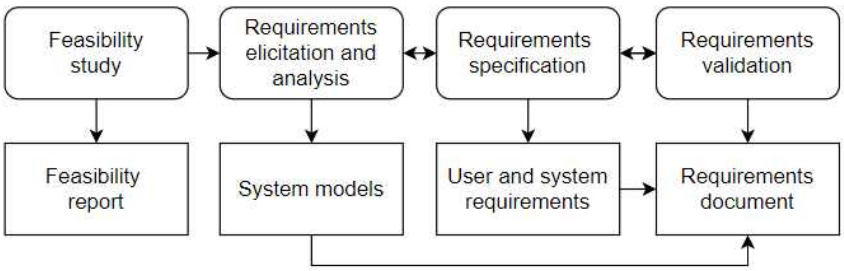
- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.

1) Types of software component
- web services : developed according to service standards and which are available for remote invocation.

- collections of objects : developed as a package to be integrated with a component framework such as .NET or J2EE.
- stand-alone software systems(COTS) : configured for use in a particular environment.

(6) Software specification

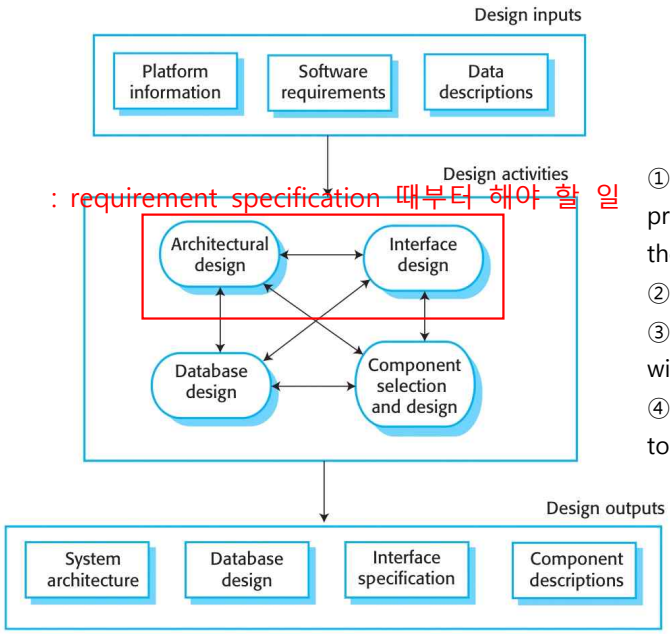
: The process of estabilishing what services are required and the constraints on the system's operation and development.



: 이때 다시 돌아가는 것은 비용이 덜 들지만, development까지 하고 다시 돌아가는 것은 비용이 많이 듦

(7) Software design and implementation

- The process of converting the system specification into an executable software system.
- Software design : Design a software structure that realizes the specification.
- Implementation : Translate this structure into an executable program.
- The activities of design and implementation are closely related and may be inter-leaved.



: requirement specification 때부터 해야 할 일

software engineer

: 이런 전반적인 과정을 할 수 있는 사람

① Architectural design : identify the overal structure of the system, the principal components(=sub-systems, modules), their relationships and how they are distributed.

② Interface design : define the interfaces between system components

③ component design : take each system component and design how it will operate.

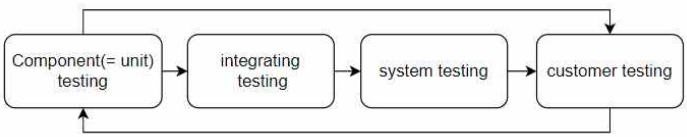
④ database design : design the system data structures and how these are to be represented in a database.

coder, programmer

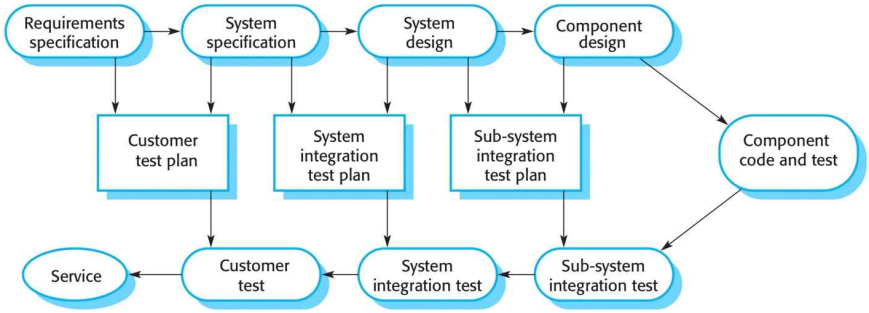
: 이걸 바탕으로 코딩하는 사람

(8) Software validation

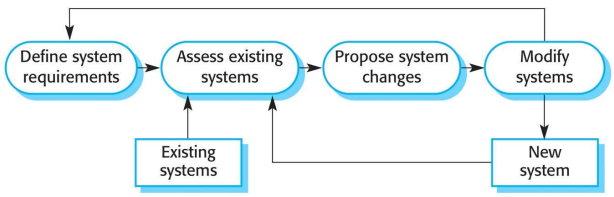
- Verification and validation (V & V) is intended to show that a system conforms to its specification (by checking, inspection, review) and meets the requirements of the system customer (by testing).
- System testing : testing system of emergent properties with test cases that are derived from the specification of the real data to be processed by the system.
- Acceptance testing : testing with customer data to check that the system meets the customer's needs



1) Testing phases in a plan-driven software process : v-shape model



(9) Software Evolution



- Software is inherently flexible and can change.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

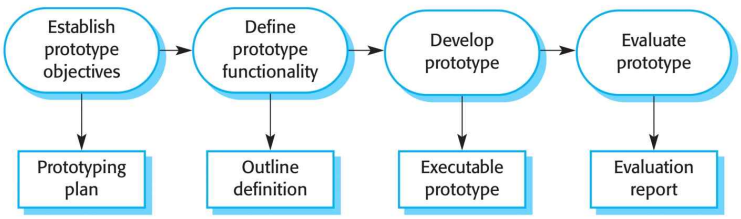
3단원 - rework

(1) Rework

- Change leads to rework so the costs of change include both rework(eg. re-analyzing requirement) as well as the costs of implementing new functionality
- Change avoidance, Change tolerance 두가지 측면

(2) Change avoidance : the software process includes activities that can anticipate possible changes before significant rework is required.

→ a prototype system may be developed to show some key features of the system to customers.



1) prototyping

- prototype : initial version of a system used to demonstrate concepts and try out design options.
- prototytpe can be used in

① requriement engineering process : to help with requriements elicitation and validation

② design process : to explore options and develop a UI design

③ testing process : to run back-to-back test

(동일한 input을 주었을 때 원하는 output을 내놓는가, input/output만 봄)

2) benefits of prototyping

- improved system usability
- a closer match to user's real needs
- improved design quality
- imporved maintainability
- reduced development effort

3) Prototype development

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality

① Prototype should focus on areas of the product that are not well-understood
② Error checking and recovery may not be included in the prototype
③ Focus on functional rather than non-functional requirements such as reliability and security

4) Throw-away prototypes

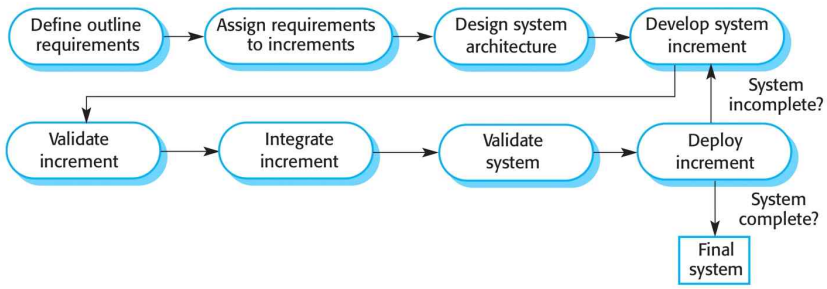
- Prototypes should be discarded after development
- they are not a good basis for a production system:

① It may be impossible to tune the system to meet non-functional requirements
② Prototypes are normally undocumented
③ The prototype structure is usually degraded through rapid change
④ The prototype probably will not meet normal organizational quality standards.
⑤ Not exploratory prototypes : prototype을 점점 개발해감

(3) Change tolerance : changes can be accommodated at relatively low cost.

→ This normally involves some form of incremental development.

→ Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.



1) Incremental delivery

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritized and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.
- Deploy an increment for use by end-users so it can get more realistic evaluation about practical use of software
- Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

2) Incremental development
- Develop the system in increments and evaluate each increment before proceeding to the development of the next increment
- Normal approach used in agile methods
- Evaluation done by user/customer proxy.

3) Incremental delivery advantages
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

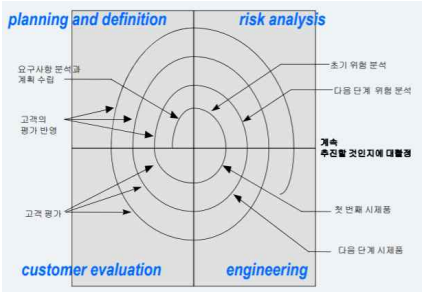
4) Incremental delivery problems
- Most systems require a set of basic facilities that are used by different parts of the system.

→ As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.

- The essence of iterative processes is that the specification is developed in conjunction with the software.

→ However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

(4) Boehm's risk-based spiral model
- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- No fixed phases such as specification or design
  : loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.



1) Spiral model sectors
- Objective setting : Specific objectives for the phase are identified.
- Risk assessment and reduction : Risks are assessed and activities put in place to reduce the key risks.
- Development and validation : A development model for the system is chosen which can be any of the generic models.
- Planning : The project is reviewed and the next phase of the spiral is planned.

2) Template for spiral round
: objectives, constraints, alternatives, risks, risk resolution, results, plans, commitments

3) Spiral model(= meta model) usage
- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.
- In practice, however, the model is rarely used as published for practical software development.

(5) The Rational Unified Process (RUP)

: modern generic process model that is organized into phases (inception, elaboration, construction and transition) but separates activities (requirements, analysis and design, etc.) from these phases.

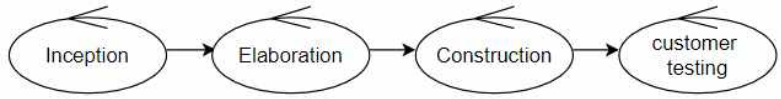
- A modern generic process derived from the work on the UML and associated process.
- Brings together aspects of the three generic process models discussed previously.
- Normally described from three perspectives

① A dynamic perspective : shows phases over time

② A static perspective : shows process activities (workflows in RUP)

③ A practice perspective : suggests good practice to be used during the process.

1) RUP phases : dynamic perspective



• In-phase iteration : Each phase is iterative with results developed incrementally.

• Cross-phase iteration : As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.

① Inception (개념화 단계)

- Establish the business case for the system.
- Identify all external entities and interactions (context diagram)
- Assess the contribution that the system makes to the business

② Elaboration

- Develop an understanding of the problem domain and the system architecture.
- Develop project plan and identify key project risks
- Req. model, architectural description, development plan

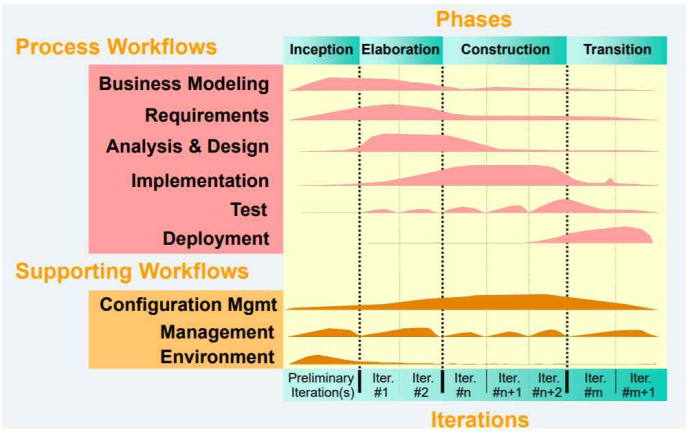
③ Construction

- System design, programming and testing (parallel dev. & integration)
- Working software, associated documents

④ Transition

- Deploy the system in its operating environment.
- Documented working SW system in its operational environment

2) Static workflows in the RUP : static perspective



Business modeling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.

Implementation	The components in the system are implemented and structured into implemetation sub-systems. Automatic code generation from design models helps accelerate this process.
Testing	Testing is an iterative process that is carried out in conjunction with implementation. system testing follows the completion of the implementation.
Deployment	A project release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow manages changes to the system.
Project management	This supporting workflow manages the system development
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

3) RUP good practice : practice perspective

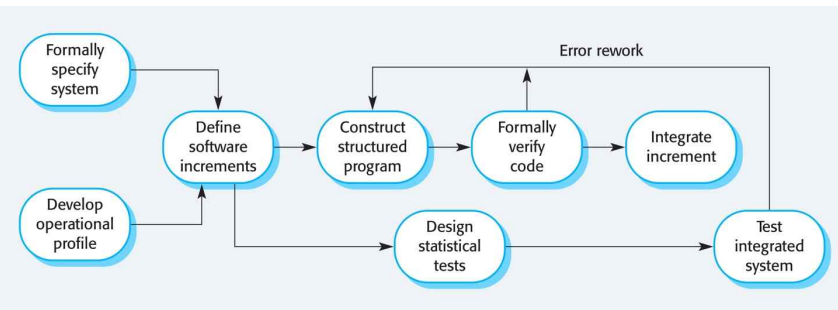
- Develop software iteratively : Plan increments based on customer priorities and deliver highest priority increments first.
- Manage requirements : Explicitly document customer requirements and keep track of changes to these requirements.
- Use component-based architectures : Organize the system architecture as a set of reusable components.
- Visually model software : Use graphical UML models to present static and dynamic views of the software.
- Verify software quality : Ensure that the software meets organizational quality standards.
- Control changes to software : Manage software changes using a change management system and configuration management tools.

(6) Cleanroom SW development model

- The philosophy is defect avoidance rather than defect removal : 처음부터 결점 없이 만들자
- 조직의 능력이 뛰어나야 사용 가능
- This software development process is based on :

① Incremental development
② Formal specification : 명세서를 자연어로 쓰지 않고, formal language로 작성함
③ Structured programming : programming을 할 때 sequence, iteration, condition만 씀
structured란 : modulization, top-down적인 approach로 step-wise, refinment한 기술
④ Static verification using rigorous SW inspections : software를 excusion하지 않고 scanning만 함
⑤ Statistical testing to determine program reliability

: 사용자가 자주 사용하는 operation의 결점을 없애야 사용자가 느끼는 reliability가 높아짐.
따라서, operational profile<input class, 사용빈도>기반으로 testing을 진행함



4단원

(1) Rapid software development
- Specification, design and implementation are inter-leaved
- System is developed as a series of versions with stakeholders involved in version evaluation
- User interfaces are often developed using an IDE and graphical toolset.

(2) Agile methods
– Focus on the code rather than the design
– based on an iterative approach to software development
– intended to deliver working software quickly and evolve quickly to meet changing requirements.
- The aim of agile methods
: to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Individual & Interactions	>	Processes & Tools
Working software		Comprehensive documentation
Customer Collaboration		Contract negotiation
Responding to change		Following a plan

1) Principles of agile methods

Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not processes	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

2) Agile method applicability
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process
- not a lot of external rules and regulations that affect the software.
- Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems. → small or medium-sized product for sale

3) Problems with agile methods
- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.

- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work. (refactoring)
- Contracts may be a problem as with other approaches to iterative development.

(3) Agile methods and software maintenance

- Most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
- Two key issues:

① Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?

문서 x
→ maintainence 감소

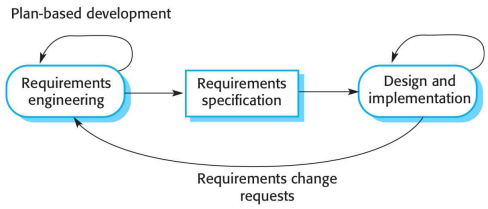
② Can agile methods be used effectively for evolving a system in response to customer change requests?

- Problems may arise if original development team cannot be maintained.

(4) Plan-driven and agile development

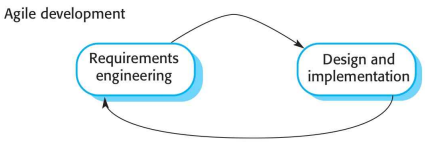
1) Plan-driven approach

- Based around separate development stages with the outputs to be produced at each of these stages planned in advance.
- Not necessarily waterfall model
- plan-driven, incremental development is possible
- Iteration occurs within activities.



2) Agile development

- Specification, design, implementation, testing are inter-leaved
- Outputs from the development process are decided through a process of negotiation during software development process.



3) Techincal, human, organizational issues

– Is it important to have a very detailed specification and design before moving to implementation? If so, you probably need to use a plan-driven approach.

– Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so, consider using agile methods.

– How large is the system that is being developed? Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally. This may not be possible for large systems that require larger development teams so a plan-driven approach may have to be used.

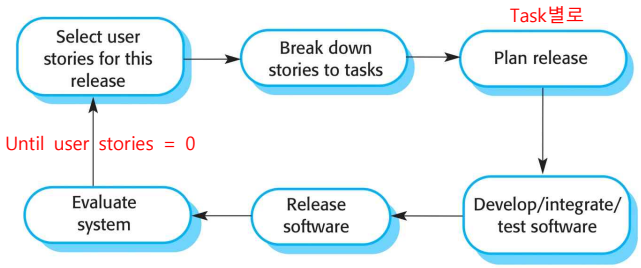
– What type of system is being developed? Plan-driven approaches may be required for systems that require a lot of analysis before implementation (e.g. real-time system with complex timing requirements).

– What is the expected system lifetime? Long-lifetime systems may require more design documentation to communicate the original intentions of the system developers to the support team.

– What technologies are available to support system development? Agile methods rely on good tools to keep track of an evolving design

– How is the development team organized? If the development team is distributed or if part of the development is being outsourced, then you may need to develop design documents to communicate across the development teams.
– Are there cultural or organizational issues that may affect the system development? Traditional engineering organizations have a culture of plan-based development, as this is the norm in engineering.
– How good are the designers and programmers in the development team? It is sometimes argued that agile methods require higher skill levels than plan-based approaches in which programmers simply translate a detailed design into code
– Is the system subject to external regulation? If a system has to be approved by an external regulator (e.g. the FAA approve software that is critical to the operation of an aircraft) then you will probably be required to produce detailed documentation as part of the system safety case.

(5) Extreme programming(XP)
- New versions may be built several times per day
- Increments are delivered to customers every 2 weeks
- All tests must be run for every build and the build is only accepted if tests run successfully.



- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

1) XP practices

Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test drived development (TDD)	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented. - Test code를 작성해서 code/application을 평가함 - unit test를 지원하므로 다양한 level의 test는 불가능
Refactoring	Expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job. (driver & observer) - This helps develop common ownership of code and spreads knowledge across the team. (Pairs are created dynamically) - Where pair programming and collective ownership are used, others benefit immediately from the refactoring so they are likely to support the process. - Individuals are not held responsible for problems with the code. Instead, the team has collective responsibility for resolving these problems. - It serves as an informal review process as each line of code is looked at by more than 1 person. - The sharing of knowledge reduces the overall risks to a project when team members leave.
Colelctive ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

2) Requriements scenarios
- User requirements are expressed as scenarios or user stories.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

3) XP and change
- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- However, XP maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

4) Refactoring
- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is wellstructured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

5) Examples of refactoring
- Re-organization of a class hierarchy to remove duplicate code.
- Tidying up and renaming attributes and methods to make them easier to understand.
- The replacement of code with calls to methods that have been included in a program library.

6) Testing in XP
① Test-first development (TDD)
- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
- Usually relies on a testing framework such as JUnit.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

= Regression test(회귀 시험)의 자동화 : 새로 고친 코드가 다른 곳에 영향을 끼치진 않았는지 확인

② Incremental test : development from scenarios

③ User involvement(= customer involvement)
- The role of the customer in the validation/testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

④ Automated test harnesses
- Test automation : tests are written as executable components before the task is implemented
- run all component tests each time that a new release is built
- development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- As testing is automated, there is always a set of tests that can be quickly and easily executed
- Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

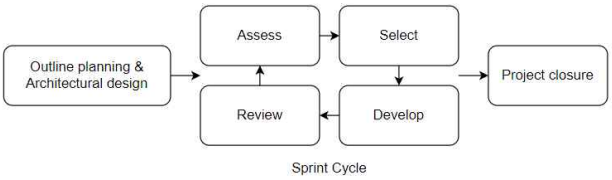
7) XP testing difficulties
- Programmers prefer programming to testing and sometimes they take short cuts when writing tests.
- Some tests can be very difficult to write incrementally. For example, in a complex user interface, it is often difficult to write unit tests for the code that implements the 'display logic' and workflow between screens.
- It is difficult to judge the completeness of a set of tests. Although you may have a lot of system tests, your test set may not provide complete coverage.

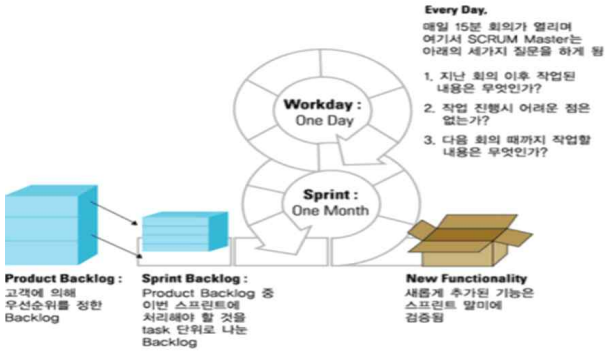
(6) Agile project management

- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the particular strengths of agile methods.

(7) Scrum

- Scrum focus is on managing iterative development rather than specific agile practices.





① Outline planning phase establish general objectives for project and design software architecture.
② A series of sprint cycles : each cycle develops an increment of the system.
③ Project closure : wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

1) Sprint Cycle

- Sprints are fixed length(2–4 weeks).
- They correspond to the development of a release of the system in XP.

① Assess : The starting point for planning is the product backlog, which is the list of work to be done on the project.
(→ sprint backlog로 나눠서 이 sprint cycle동안에 수행해야할 작업을 계획함)
② Selection : involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.
③ Develop : During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called 'Scrum master'.
④ Review : the work done is reviewed and presented to stakeholders. The next sprint cycle begins.

2) Teamwork in Scrum

- The 'Scrum master' is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog, communicates with customers and management outside of the team and to protect the development team from external distractions.
- The whole team attends short daily meetings where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day. → 문서화를 안하지만 이를 보강해줌
- This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

3) Scrum benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

(7) Large systems development

- Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones.
- Large systems are 'brownfield systems', that is they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development.
- Where several systems are integrated to create a system, a significant fraction of the development is concerned with system configuration rather than original code development. Large systems and their development processes are often constrained by external rules and regulations limiting the way that they can be developed.
- Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.
- Large systems usually have a diverse set of stakeholders. It is practically impossible to involve all of these different stakeholders in the development process.

5단원 - Requirements specification

(1) Requirement

: Descriptions of the system services and constraints that are generated during the requirements engineering process.

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
- May be the basis for a bid for a contract - therefore must be open to interpretation.
- May be the basis for the contract itself - therefore must be defined in detail.

1) Types of requirement

① User requirements

- Statements in natural language plus diagrams, table of the services the system provides and its operational constraints.
- Written for customers. + Client managers, Constractor Manageers, System End-users, Client engineering, System Architecture

② System requirements

- A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.
- Written for Software Developers, System End-users, Client engineering, System Architecture

2) Functional and non-functional requirements

① Domain requirements

- constraints on the system from the domain of operation.
- The system's operational domain imposes requirements on the system.
- Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
- If domain requirements are not satisfied, the system may be unworkable.

② Functional requirements

- Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations and descriptions of how some computations must be carried out.
- May state what the system should not do.
- Describe functionality or system services.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail

③ Non-functional requirements

- System properties and constraints while developing and development process being used.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.
- Non-functional requirements may affect the overall architecture of a system rather than the individual components(features, services). = Emergent properties

3) Requirement imprecision
- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- In principle, requirements should be both complete and consistent.
- Complete : They should include descriptions of all facilities required.
- Consistent : There should be no conflicts or contradictions in the descriptions of the system facilities.

4) Domain requirements problems
① Understandability
- Requirements are expressed in the language of the application domain;
- This is often not understood by software engineers developing the system.
② Implicitness
- Domain specialists understand the area so well that they do not think of making the domain requirements explicit and how it interacts with other requirements.

(2) Goals and requirements
- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.
- Goal : A general intention of the user such as ease of use.
- Verifiable non-functional requirement : A statement using some measure that can be objectively tested.
- Goals are helpful to developers as they convey the intentions of the system users.

(3) Software requirements document(=specification)
: Official statement of what is required of the system developers.
- Should include both definition of user requirements and specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.
- The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.

1) Agile methods and requirements
- Methods such as XP use incremental requirements engineering and express requirements as 'user stories'.
- This is practical for business systems but problematic for systems that require a lot of pre-delivery analysis (e.g. critical systems) or systems developed by several teams or embedded system.

2) Requirements document variability
- Information in requirements document depends on type of system and the approach to development used.
- Systems developed incrementally will, typically, have less detail in the requirements document.
- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

(4) Requirements specification
- The process of writing the user and system requirements in a requirements document.
- The requirements may be part of a contract for the system development
- user requirement / system requirement 존재

1) Ways of writing a system requriements specification

Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML(Unified Model Language) use case and sequence diagrams are commonly used.
Mathematical specifications FDT(Formal Description Techinque) 기반	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract.

2) Requirements and design
- Requirements should state what the system should do
- Design should describe how it does this.
- Requirements and design are inseparable : architecture, abstract, interface design 겹침

① A system architecture may be designed to structure the requirements
② The system may inter-operate with other systems that generate design requirements
③ The use of a specific architecture to satisfy non-functional requirements may be a domain requirement. This may be the consequence of a regulatory requirement

(5) Natural language specification
- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

1) Problems
① Lack of clarity : Precision is difficult without making the document difficult to read.
② Requirements confusion : Functional and non-functional requirements tend to be mixed-up.
③ Requirements amalgamation : Several different requirements may be expressed together.

2) guidelines for wirting requriement

① Invent a standard format and use it for all requirements.
② Use language in a consistent way.
③ Use shall for mandatory requirements, should for desirable requirements.
④ Use text highlighting to identify key parts of the requirement.
⑤ Avoid the use of computer jargon.
⑥ Include an explanation (rationale) of why a requirement is necessary

(6) Structed specification

- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.
- Form-based specification

① Definition of the function or entity.
② Description of inputs and where they come from.
③ Description of outputs and where they go to.
④ Information about the information needed for the computation and other entities used.
⑤ Description of the action to be taken.
⑥ Pre and post conditions (if appropriate).
⑦ The side effects (if any) of the function.

(7) Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.

5단원 – Requirements engineering processes

(1) Requirements engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organization developing the requirements.

- The requirements engineering process is an iterative process including requirements elicitation, specification and validation.

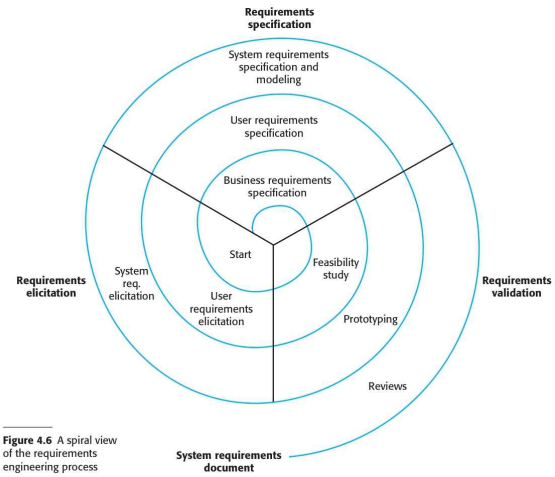


Figure 4.6 A spiral view of the requirements engineering process

(2) Requirements Elicitation(=discovery) and Analysis

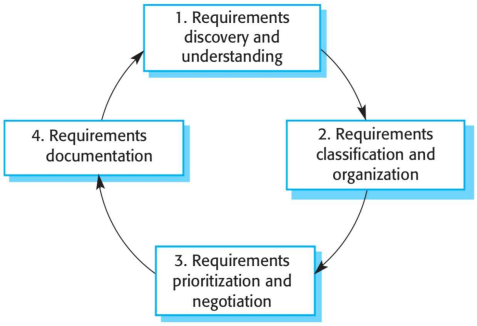
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.
- An iterative process that can be represented as a spiral of activities

1) Process activities

① Requirements discovery

: Interacting with stakeholders to discover their requirements.

- Domain requirements are also discovered at this stage.
- The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
- Interaction is with system stakeholders from managers to external regulators.
- by interviewing



② Requirements classification and organization

: Groups related requirements and organized them into coherent clusters.

③ Prioritization and negotation : Prioritizing requirements and resolving requirements conflicts.

④ Requirements specification

: Requirements are documented and input into the next round of the spiral

2) Problems of requriements elicitation

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organizational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment change.

(3) Interviewing

- Formal or informal interviews with stakeholders are part of most RE processes.

1) Types of interview

① Closed interviews : based on pre-determined list of questions

② Open interviews : various issues are explored with stakeholders.

2) Effective interviewing

- Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
- Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

3) Interviews in practice

- Normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviews are not good for understanding domain requirements

→ Requirements engineers cannot understand specific domain terminology

→ Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

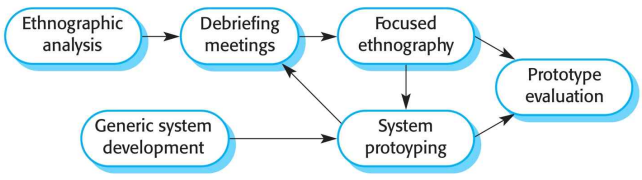
(4) Scenarios
: real-life examples of how a system can be used.

1) They should include
– A description of the starting situation
– A description of the normal flow of events
– A description of what can go wrong
– Information about other concurrent activities
– A description of the state when the scenario finishes

2) Use cases
- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself(use-case).
- A set of use cases should describe all possible interactions with the system.
- High-level graphical model supplemented by more detailed tabular description.
- Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.
- stackholder와 사용자의 관점에서 use case마다 어떤 actor가 영향이 있는지 알 수 있음
- 하지만 충분한 정보를 포함하진 못하므로 tabular description과 sequence diagram을 같이 써서 보충

(5) Ethnography 참여적 관찰법
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.
- Requirements that are derived from the way that people actually work rather than the way which process definitions suggest that they ought to work.
- Requirements that are derived from cooperation and awareness of other people's activities.
- Awareness of what other people are doing leads to changes in the ways in which we do things.
- Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

1) Focused enthnography



- Developed in a project studying the air traffic control process
- Combines ethnography with prototyping
- Prototype development results in unanswered questions which focus the ethnographic analysis.
- The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant
- ethnographic analysis를 함으로써 prototype의 overhead를 줄일 수 있으며, evaluation할 때도 유용함

Range of techniques for requirements elicitation : interviews, scenarios, use cases and ethnography.

(6) Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
- Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

1) Requirement checking

- Validity : Does the system provide the functions which best support the customer's needs?
- Consistency : Are there any requirements conflicts?
- Completeness : Are all functions required by the customer included?
- Realism : Can the requirements be implemented given available budget and technology
- Verifiability : Can the requirements be checked? goal vs requirement
- validity vs completeness : 고객 need를 충족하도록 가지고 있는가 vs 고객이 원하는 need가 다 있는가

2) Requirements validation techniques

① Requirements reviews

: Systematic manual analysis of the requirements.

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.
- formal review : document가 존재하며, 관계자들이 정해진 시간, 장소에 모여서 진행
- informal review : 미완성된 document를 중간에 하는 개별적 검토
- review checking
  - Verifiability : Is the requirement realistically testable?
  - Comprehensibility : Is the requirement properly understood?
  - Traceability : Is the origin of the requirement clearly stated?
  - Adaptability : Can the requirement be changed without a large impact on other requirements?

② Prototyping

: Using an executable model of the system to check requirements.

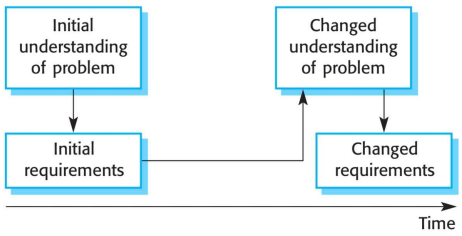
③ Test-case generation

: Developing tests for requirements to check testability.

(7) Requirement Management

- the process of managing and controlling changing requirements during the requirements engineering process and system development. → maintanence와 관련
- Business, organizational and technical changes inevitably lead to changes to the requirements for a software system.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.
- Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.

1) Requirement Evolution



2) Requirments management planning

- Establishes the level of requirements management detail that is required.
- Requirements management decisions

① Requirements identification

: Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.

② A change management process 조직마다 requirement request를 받는 방법, 평가방법, 실행 방법 정립

: This is the set of activities that assess the impact and cost of changes.

③ Traceability policies

: These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.

+ between design and code : 어떤 code를 변경 → 어떤 design에 영향 → 어떤 requirement에 영향

④ Tool support

: Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

3) Requirements change management

- Deciding if a requirements change should be accepted

① Problem analysis and change specification

- The problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor who may respond with a more specific requirements change proposal, or decide to withdraw the request. (a validity of change request)

② Change analysis and costing 기술, 비용 고려

- The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change. (a validity of change execution)

③ Change implementation

- The requirements document and, where necessary, the system design and implementation, are modified. Ideally, the document should be organized so that changes can be easily implemented.

The fundamental stages of system engineering contain conceptual design, procurement, development and operation. The conceptual design sets out the purpose of the system, why it is needed and the high-level features that users expect to see in the system. This design process involves the following activities: concept formulation, problem understanding, system proposal development, feasibility study, system structure development and system vision document. Of these, what are related with the next procuring process?
system vision document.

User requirements are described in natural language plus diagrams of the services the system provides and its operational constraints. It is written for contractor.
아니요. User requirement is written for the customer

The requirements engineering processes vary widely depending on the application domain, the people involved and the organization developing the requirements. There are a number of generic activities common to all processes: requirements elicitation, requirements analysis, requirements validation, and requirements management. In practice, RE is a sequential activity that looks like the waterfall model.
아니요. sequential activity는 이론, 실제로는 iterative process in which the activities are interleaved.

The software process is a structured set of activities required to develop a software system. And it commonly involves specification, design, implementation, validation, and evolution. The validation is to check whether the system works as specified.
아니오. validation -> verfication

The sociotechnical systems can be represented as a seven-layer stack structure in the following order: equipment, operating system, application system, communications and data management, organization, business processes and society.
아니요. Seven-layer stack structure는 equipment, operating system, communications and data management, application system, business processes, organization, society 순서로 이루어져 있다.

System properties as a whole rather than properties that can be derived from the properties of components of a system. They can therefore only be assessed and measured once the components have been integrated into a system. What is this?
Emergent property.

The waterfall model involves the following steps in sequence: Requirements definition-System and software design-Implementation and acceptance testing-Integration and system testing-Operation and maintenance
아니요. implementation and acceptance testing -> implementation and unit testing

The waterfall model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process. Few business systems have stable requirements and rather they have to reflect rapidly changing requirements, so engineers must adopt the waterfall model.
아니요. Reflect rapidly changing requirements를 위해서 waterfall model을 사용하는 것은 적절하지 않다. Waterfall model은 변화가 많지 않을때 사용해야 된다.

The incremental development has some problems as follows: Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system. It means that this approach has inherently poor visibility. Also, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly easier and cheaper, so additional effort for refactoring is required.
아니요. Incorporating further software changes becomes harder and expensive.

Reuse-based software development is pursuing the systematic reuse of existing components or COTS systems. The most popular reusable component in recent years is open source software component. Of particular importance when using open source is the identification of the type of license. The most stringent license is the MIT license.
예

Requirement engineering is the process of establishing what services are required and the constraints on the system's operation and development. It contains feasibility study, requirements elicitation and analysis, requirements specification, requirements validation, and testing.
아니요. testing 없음

Software validation and verification is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system respectively.
아니요. Validation과 verification 각각에 대한 설명이 반대로 서술되어 있다. Verification이 checking that the system conforms to its specification에 해당하고 validation이 checking that it meets the real needs of the users에 해당한다.

Change is inevitable in all large software projects. Change leads to rework so the costs of change include rework such as re-analysis and re-design as well as the costs of implementing new functionality. To reduce the costs of rework, change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required. As another approach, change tolerance, where the process is designed so that changes can be accommodated at relatively low cost. Prototyping may be a good solution for change tolerance.
아니요. Prototyping은 change avoidance를 위한 좋은 솔루션이다.

A prototype is an initial version of a system used to demonstrate concepts and try out design options. A prototype can be used in the requirements engineering process to help with requirements elicitation and validation. And also used in design processes to explore options and develop a UI design. The prototype is also used as a baseline to be extended to a large system.
아니요. Prototype은 large system으로 확장되기 위한 baseline으로 사용되지 않고, prototype을 사용한 후에는 해당 prototype을 버려야 한다.

The benefits of incremental development and delivery are: customer value can be delivered with each increment so system functionality is available earlier, early increments act as a prototype to help elicit requirements for later increments, lower risk of overall project failure, and the highest priority system services tend to receive the least testing.
아니요. Highest priority system services가 testing을 가장 많이 받게 된다.

The aim of agile methods is to reduce overheads in the software process and to be able to respond quickly to changing requirements without excessive rework. And there was value in the

items on the right previously, we value the items on the left more now: Individuals and interactions over processes and tools, comprehensive documentation over working software, customer collaboration over contract negotiation, responding to change over following a plan.
아니요. Agile method에서는 working software를 comprehensive documentation보다 중요시한다.

The agile method is applicable under the circumstance: product development where a software company is developing a small or medium-sized product for sale, custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process, and where there are a lot of external rules and regulations that affect the software.
아니요. When there are a lot of external rules and regulations that affect the software의 경우 agile method를 사용하기 적절하지 않다. External rules and regulations가 많지 않아야 한다.

The agile methodology tries to maintain the simplicity of the code to reduce the burden of documentation. This improves the understandability of the software and so reduces the need for documentation. What is the technical activity for this?
Refactoring

Testing is central to XP and XP has developed an approach where the program is tested after every change has been made. Typical XP testing features are: test-first development, incremental test development from scenarios, user involvement in test development and validation, automatic GUI test, and automated test harnesses.
아니요. Typical XP testing features에 automatic GUI test는 포함되지 않는다.

Agile project management requires a different approach, which is adapted to incremental development and the particular strengths of agile methods. The Scrum approach is a general agile method but its focus is on managing sequential development rather than specific agile practices.
예

Many agile methods argue that producing a requirements document is a waste of time as requirements change so slowly.
아니요. Requirements change so fast.

Agile methods such as XP use incremental requirements engineering and express requirements as '(user stories)'.

Large systems are usually not suitable applying incremental development with the following reasons. They are brownfield systems, that is, they include and interact with a number of existing systems. Many of the system requirements are concerned with this interaction and so don't really lend themselves to flexibility and incremental development. But large systems and their development processes are often not constrained by external rules and regulations limiting the way that they can be developed.
아니요. Large systems are constrained by external rules and regulations.

Typical design activities involve abstract design, architectural design, interface design, component design, and database design. Of these, architectural design and component design overlap with the requirements engineering course.

아니요. Component design does not overlap with the requirements engineering course.

Test cases are a set of input data and expected output data. The software tester runs a series of test cases to check if the execution passed or not to check for any defects in the software. If it fails, it is very likely that the software is defective. However, the premise is that the test case is well made. The test case uses numeric data, not characters.
아니요. Test case uses characters.

A system is a purposeful collection of inter-related components working together to achieve some common objective. Software engineering is not an isolated activity but is part of a broader systems engineering process. Nonetheless, software systems are isolated systems and are not essential components of broader systems that have a human, social or organizational purpose.
아니요. Software systems are essential components of broader systems.

The requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process. The requirement engineering is the process of establishing the requirements. The requirements may serve a dual function: the basis for a bid for a contract - therefore must be defined in detail. The basis for the contract itself - therefore must be open to interpretation.
아니요. Basis for a bid for a contract - therefore must be open to interpretation, basis for the contract itself - therefore must be defined in detail이 맞는 설명이다.

Interview is the most popular way of gathering user requirements. It is usually conducted with a mix of closed and open-ended types. It is good for getting an overall understanding of what stakeholders do and how they might interact with the system. It is also good for understanding domain requirements.
아니요. Interview is not good for understanding domain requirements.

Ethnography is a good way to acquire hidden user requirements by observing and analyzing how people actually work. Because human's work is usually richer and more complex than suggested by simple system models. Ethnography is effective for understanding existing processes and can identify new features that should be added to a system.
아니요. Ethnography is not effective for identifying new features that should be added to a system.

Non-functional requirements are often apply to the system as a whole rather than individual features or services. System properties like safety, performance, and etc. Non-functional requirements may not be more critical than functional requirements. Domain requirements are constraints on the system from the domain of operation.
아니요. Non-functional requirements are more critical than functional requirements.

Agile method의 특징을 고르시오
process oriented, risk driven development, customer involvement, incremental delivery, refactoring, formal method, test-driven development, tightly-integrated team
: risk driven development, customer involvement, incremental delivery, refactoring, TDD, tightly-integrated team

high dependability를 요구하는 상황에서 agile의 applicapability를 설명해라

1. Testing의 2가지 목표
- validation testing : developer와 cutomer의 입장에서 software가 고객의 requirements와 일치하는지 보여주는 것. 이 경우에 성공적인 테스트는 시스템이 의도한 대로 동작하는가를 보여줌
- defect testing : software 내에 존재하는 defect를 발견하는 것이 목적으로 이 경우엔 성공적인 테스트는 시스템이 비정상적으로 동작하게끔 만드는 것. 시스템 내에 있는 defect가 노출되는 것
  ex) unit testing, component testing : whitebox

+ Regression testing : 코드가 끊임없이 수정되고 변경된 코드를 실행할 때 이전의 코드들에 나쁜 영향이 미치지 않았는가를 확인하는 테스트로, TDD처럼 자동화된 테스팅 환경은 regression testing의 부하를 줄여줌
+ Release testing : 만들어진 시스템을 release 할지 말지 결정하는 테스팅. 시장으로부터의 요구상이 충족되었는지를 확인 : blackbox 내부는 잘 모르고 입/출력만 확인하는 것

2. Risk management 의 4단계
- risk identification : identify project, project and business risks
- risk analysis : assess the likelihood and consequence of these risks
- risk planning : draw up plans to avoid or minimise the effects of the risk
- risk monitoring : monitor the risks throughout the project

3. Inspection에서 확인할 수 있는 것은?
Maintainability O, Performance, Availability, Usability X
- Inspection이란 verification 기술 : specification과 일치하는지 확인하는 것이지 실제 customer의 real requirement를 충족하는지 보는 것은 아님 따라서 testing과 같이 수행하는 것이 필요함. testing이 찾아내지 못하는 특성(품질 특성, quality attribute)을 찾아내지만, 반대로 performance, usability는 testing에서만 확인 가능
- Inspection 강점 : 실제로 시스템을 실행시킬 필요가 없기 때문에 implementation을 하지 않아도 시스템 검증이 가능함. 모든 종류의 representation에 대해서 적용 가능함. 일반적인 테스트는 실행하다가 에러가 발견되면 고치고 다시 실행시키고 다음 에러를 발견하고,, 이렇게 sequentially하게 작업하지만, 이런 과정은 결국 하나의 에러가 다른 에러를 hide하므로 반복적으로 수행하는 문제가 있음. 그러나 Inspection은 한번에 쭉 스캔하면서 한 사람이 에러를 동시다발적으로 확인 가능함. 실제로 돌아가지 않는 incomplete version에서도 추가적인 비용 없이도 수행 가능. 이때 추가적인 비용이란 모델에 대해 dummy module을 만들면서 계속 테스트하는 비용임

4. verification vs validation
- verification : 문서를 충족 by inspection → quality 판단 가능 : standard, portability, maintainability
- validation : 요구사항을 충족 by testing → performance, usability

5. TDD
- 보다 튼튼한 객체지향적인 코드 생산 가능, 재설계 시간의 단축, agile과의 시너지 효과, 추가 구현의 용이함, 테스트 원칙 때문에 쉽게 넘어가지 못하는 단점(융통성 발휘 문제)

6. Back – to –Back test
: 2개 또는 그 이상의 다양한 컴포넌트나 시스템을 동일한 입력 값으로 실행하는 테스팅, 실행 결과를 비교하여 불일치한 경우 이를 분석함