E-notes Software Engineering

By Sagar Gharate

Syllabus

Chapter 3: Requirements Analysis Version 1.0

- 3.1 Requirement Elicitation,
- 3.2 Software requirement specification (SRS)
  - 3.2.1 Developing Use Cases (UML)
- 3.3 Building the Analysis Model
  - 3.3.1Elements of the Analysis Model
  - 3.3.2Analysis Patterns
  - 3.3.3 Agile Requirements Engineering
- 3.4 Negotiating Requirements
- 3.5 Validating Requirements

References:

1. Software Engineering : A Practitioner's Approach - Roger S. Pressman, McGraw hill(Eighth Edition) ISBN-13: 978-0-07-802212-8, ISBN-10: 0-07-802212-6
2. A Concise Introduction to Software Engineering - PankajJalote, Springer ISBN: 978-1-84800-301-9
3. The Unified Modeling Language Reference Manual - James Rambaugh, Ivar Jacobson, Grady Booch ISBN 0-201-30998-X

Chapter 3

Requirements Analysis

Requirement Engineering Process

- Requirement Engineering means that requirements for a product are defined, managed and tested systematically.
- Requirements engineering builds a bridge to design and construction.
- Requirements engineering provides the appropriate mechanism for understanding what the customer wants, analysing need, assessing feasibility, negotiating a reasonable solution, specifying the solution unambiguously, validating the specification, and managing the requirements as they are transformed into an operational system.

Requirement Engineering Tasks

- **Inception:**
    - Establish a basic understanding of the problem and the nature of the solution.
    - In project inception, you establish a basic understanding of the problem, the people who want a solution, the nature of the solution that is desired, and the effectiveness of preliminary communication and collaboration between the other stakeholders and the software team.
- **Elicitation**
    - Draw out the requirements from stakeholders.
    - Numbers of problems that are encountered as elicitation occurs are problems of scope, problems of understanding, problems of volatility.
- **Elaboration (Highly structured)**
    - Create an analysis model that represents information, functional, and behavioural aspects of the requirements.
    - Elaboration is driven by the creation and refinement of user scenarios that describe how the end user (and other actors) will interact with the system.
    - Each user scenario is parsed to extract analysis classes- business domain entities that are visible to the end user.
    - The attributes of each analysis class are defined, and the services that are required by each class are identified.
    - The relationships and collaboration between classes are identified, and a variety of supplementary diagrams are produced.
- **Negotiation**
    - Agree on a deliverable system that is realistic for developers and customers.
    - The best negotiations strive for a “win-win” result.
    - That is, stakeholders win by getting the system or product that satisfies the majority of their needs and you (as a member of the software team) win by working to realistic and achievable budgets and deadlines.
    - Activities in negotiation are:
    - Identification of the system or subsystem’s key stakeholders.
    - Determination of the stakeholder’s “wins conditions.”
    - Negotiation of the stakeholders’ win conditions to reconcile them into a set of win-win conditions for all concerned (including the software team).
- **Specification**
    - Describe the requirements formally or informally.

- A specification can be a written document, a set of graphical models, a formal mathematical model, and a collection of usage scenarios, a prototype, or any combination of these.

- **Validation**
  - Review the requirement specification for errors, ambiguities, omissions, and conflicts.
  - Requirements validation examines the specification to ensure that all software requirements have been stated unambiguously; that inconsistencies, omissions, and errors have been detected and corrected; and that the work products conform to the standards established for the process, the project, and the product.
  - The primary requirements validation mechanism is the technical review.
  - The review team that validates requirements includes software engineers, customers, users, and other stakeholders who examine the specification looking for errors in content or interpretation, areas where clarification may be required, missing information, inconsistencies (a major problem when large products or systems are engineered), conflicting requirements, or unrealistic (unachievable) requirements.
- **Requirements management**
  - Manage changing requirements.
  - Requirements management is a set of activities that help the project team identify, control, and track requirements and changes to requirements at any time as the project process.
  - Many of these activities are identical to the software configuration management.

- ## Requirement Elicitation

  - Requirements elicitation (also called requirements gathering) combines elements of problem solving, elaboration, negotiation, and specification.
  - In order to encourage a collaborative, team-oriented approach to requirements gathering, stakeholders work together to identify the problem, propose elements of the solution, negotiate different approaches and specify a preliminary set of solution requirements.

Collaborative Requirements Gathering

- The goal is to identify the problem, propose elements of the solution, negotiate different approaches, and specify a preliminary set of solution requirements in an atmosphere that is conducive to the accomplishment of the goal.
- Meetings are conducted and attended by both software engineers and other stakeholders.
- Rules for preparation and participation are established.
- An agenda is suggested that is formal enough to cover all important points but informal enough to encourage the free flow of ideas.
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting.
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room, or virtual forum) is used.

Quality Function Deployment:

- Quality function deployment (QFD) is a quality management technique that translates the needs of the customer into technical requirements for software. QFD "concentrates on maximizing customer satisfaction from the software engineering process".
- QFD identifies three types of requirements normal requirements, expected requirements, exciting requirements.

Usage Scenarios

- As requirements are gathered, an overall vision of system functions and features begins to materialize.
- Users can create a set of scenarios that identify a thread of usage for the system to be constructed.
- The scenarios, often called use cases.

Elicitation Work Products

- The work products produced as a consequence of requirements elicitation will vary depending on the size of the system or product to be built. For most systems, the work products include
- A statement of need and feasibility.
- A bounded statement of scope for the system or product.

- A list of customers, users, and other stakeholders who participated in requirements elicitation.
- A description of the system's technical environment.
- A list of requirements (preferably organized by function) and the domain constraints that applies to each.
- A set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- Any prototypes developed to better define requirements.

System Requirement Specification (SRS)

- It contains a complete information description, a detailed functional description, a representation of system behaviour, an indication of performance requirements and design constraints, appropriate validation criteria, and other information pertinent to requirements.
- Software requirement specification (SRS) is a document that completely describes what the proposed software should do without describing how software will do it.
- The basic goal of the requirement phase is to produce the SRS, Which describes the complete behaviour of the proposed software.
- SRS is also helping the clients to understand their own needs.

Characteristics of an SRS:

Software requirements specification should be accurate, complete, efficient, and of high quality, so that it does not affect the entire project plan. An SRS is said to be of high quality when the developer and user easily understand the prepared document. Other characteristics of SRS are discussed below.

Correct:

- SRS is correct when all user requirements are stated in the requirements document.
- The stated requirements should be according to the desired system.
- This implies that each requirement is examined to ensure that it (SRS) represents user requirements.
- Note that there is no specified tool or procedure to assure the correctness of SRS. Correctness ensures that all specified requirements are performed correctly.

Unambiguous:

- SRS is unambiguous when every stated requirement has only one interpretation.
- This implies that each requirement is uniquely interpreted.
- In case there is a term used with multiple meanings, the requirements document should specify the meanings in the SRS so that it is clear and easy to understand.

Complete:

- SRS is complete when the requirements clearly define what the software is required to do.

- This includes all the requirements related to performance, design and functionality.

Ranked for importance/stability:

- All requirements are not equally important, hence each requirement is identified to make differences among other requirements.
- For this, it is essential to clearly identify each requirement. Stability implies the probability of changes in the requirement in future.

Modifiable:

- The requirements of the user can change, hence requirements document should be created in such a manner that those changes can be modified easily, consistently maintaining the structure and style of the SRS.

Traceable:

- SRS is traceable when the source of each requirement is clear and facilitates the reference of each requirement in future.
- For this, forward tracing and backward tracing are used.
- Forward tracing implies that each requirement should be traceable to design and code elements.
- Backward tracing implies defining each requirement explicitly referencing its source.

Verifiable:

- SRS is verifiable when the specified requirements can be verified with a cost-effective process to check whether the final software meets those requirements.
- The requirements are verified with the help of reviews. Note that unambiguity is essential for verifiability.

Consistent:

- SRS is consistent when the subsets of individual requirements defined do not conflict with each other.
- For example, there can be a case when different requirements can use different terms to refer to the same object.
- There can be logical or temporal conflicts between the specified requirements and some requirements whose logical or temporal characteristics are not satisfied.
- For instance, a requirement states that an event 'a' is to occur before another event 'b'. But then another set of requirements states (directly or indirectly by transitivity) that event 'b' should occur before event 'a'.

Functional and Non-Functional Requirements

Functional requirements:

- These describe the functionality of a system - how a system should react to a particular set of inputs and what should be the corresponding output.
- Given a problem statement, the functional requirements could be identified by focusing on the following points:
    - Identify the high level functional requirements simply from the conceptual understanding of the problem. For example, a Library Management System, apart from anything else, should be able to issue and return books.
    - Identify the cases where an end user gets some meaningful work done by using the system. For example, in a digital library a user might use the "Search Book" functionality to obtain information about the books of his interest.
    - If we consider the system as a black box, there would be some inputs to it, and some output in return. This black box defines the functionalities of the system. For example, to search for a book, user gives title of the book as input and get the book details and location as the output.
    - Any high level requirement identified could have different sub-requirements. For example, "Issue Book" module could behave differently for different class of users, or for a particular user who has issued the book thrice consecutively.

Non-Functional requirements:

- They are not directly related what functionalities are expected from the system.
- However, NFRs could typically define how the system should behave under certain situations.
- For example, a NFR could say that the system should work with 128MB RAM.

Product requirements:

- Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.

Organisational requirements:

- Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.

External requirements

- Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

- **Use case Diagram, Symbols and Examples**

As UML describes the real-time systems, it is very important to make a conceptual model and then proceed gradually.

- The building blocks of UML can be defined as −
  - Things
  - Relationships
  - Diagrams

- **Things**

Things are the most important building blocks of UML. Things can be

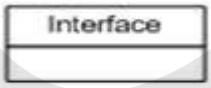
  - Structural
  - Behavioral
  - Grouping
  - Annotational

Structural Things

Structural things define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.

Class – Class represents a set of objects having similar responsibilities.

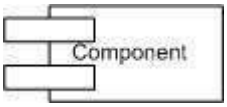
Interface – Interface defines a set of operations, which specify the responsibility of a class.



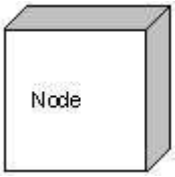
Collaboration –Collaboration defines an interaction between elements.

Use case –Use case represents a set of actions performed by a system for a specific goal.

Component –Component describes the physical part of a system.



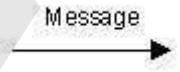
Node – A node can be defined as a physical element that exists at run time.



Behavioral Things

A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things -

Interaction – Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.



State machine – State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change

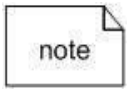
Grouping Things

Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available –

Package – Package is the only one grouping thing available for gathering structural and behavioral things.

Annotational Things

Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.



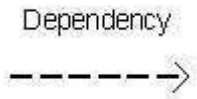
- **Relationship**

Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are 4 kinds of relationships available.

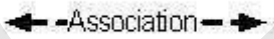
1. **Dependency**

Dependency is a relationship between two things in which change in one element also affects the other.



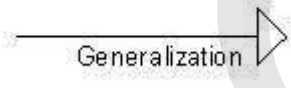
2. **Association**

Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.



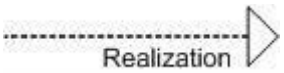
3. **Generalization**

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.



4. **Realization**

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.



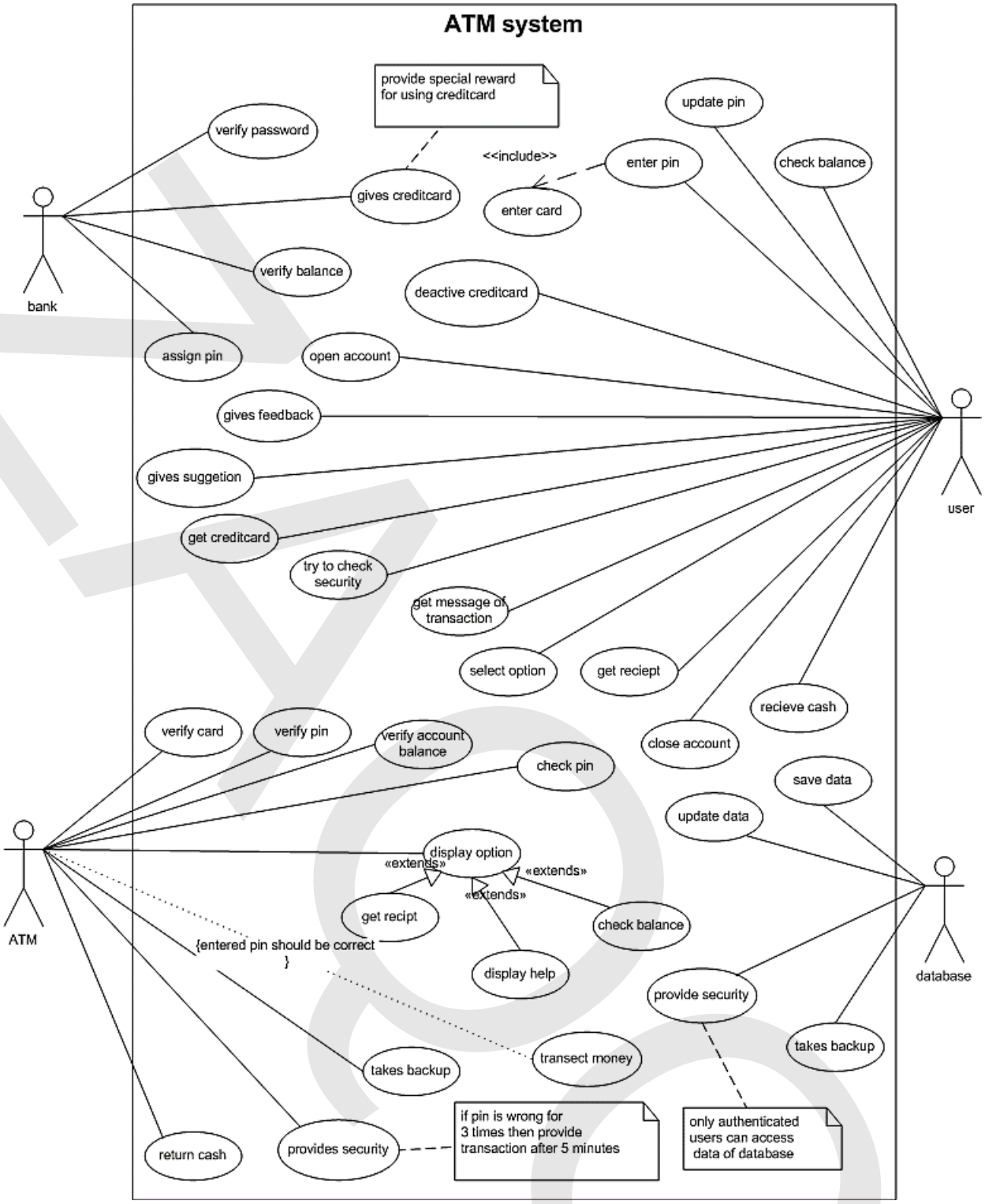
- **UML Diagrams**

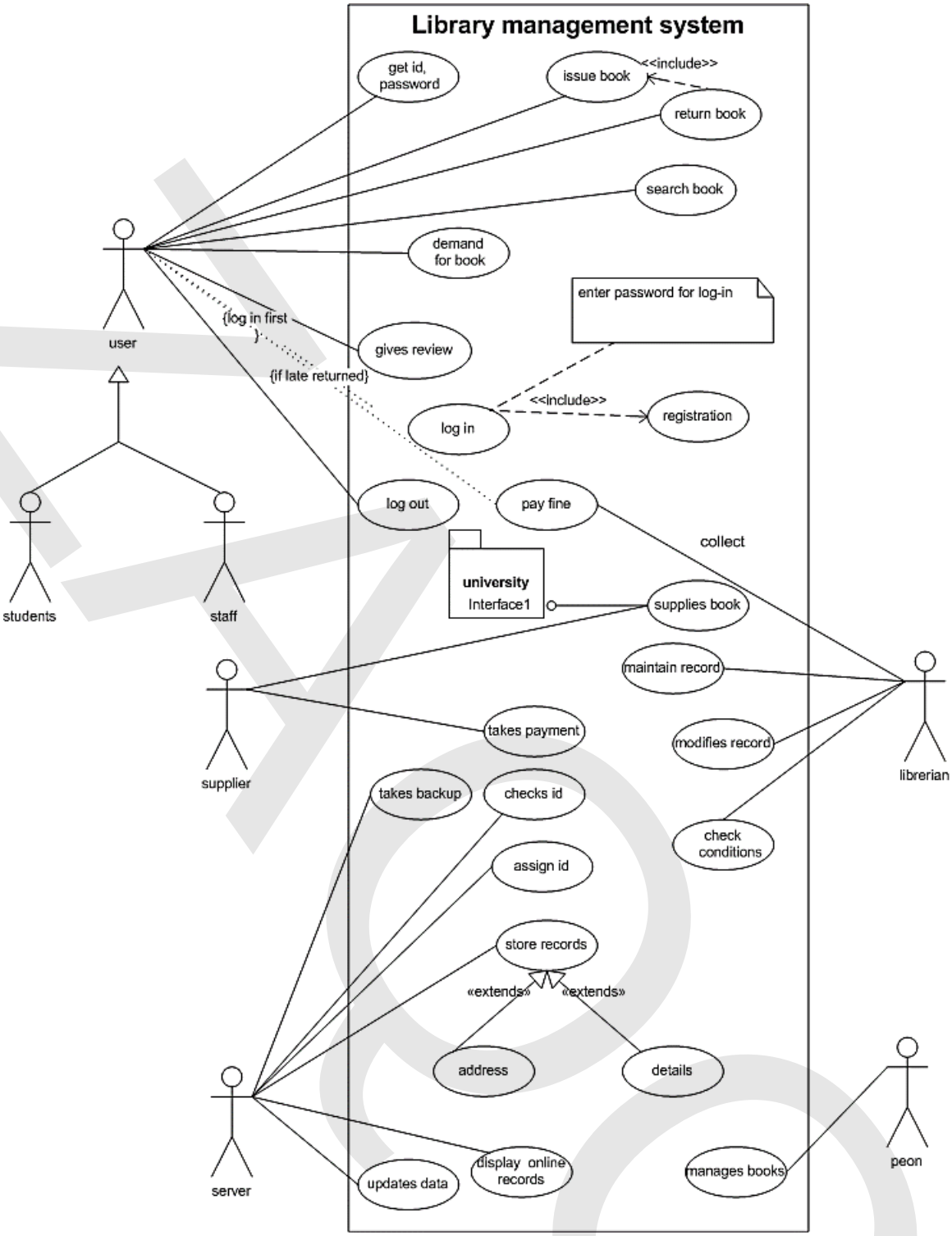
UML diagrams are the ultimate output of the entire discussion. All the elements, relationships are used to make a complete UML diagram and the diagram represents a system.

The visual effect of the UML diagram is the most important part of the entire process. All the other elements are used to make it complete.

UML includes the following nine diagrams, the details of which are described in the subsequent chapters.

- Class diagram
- Object diagram
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Activity diagram
- Statechart diagram
- Deployment diagram
- Component diagram


ATM system
provide special reward for using creditcard
verify password
update pin
<<include>>
enter pin
check balance
gives creditcard
enter card
bank
verify balance
deactive creditcard
assign pin
open account
gives feedback
gives suggetion
user
get creditcard
try to check security
get message of transaction
select option
get reciept
recieve cash
verify card
verify pin
verify account balance
close account
check pin
save data
update data
display option
«extends»
«extends»
«extends»
get recipt
check balance
ATM
{entered pin should be correct }
display help
database
provide security
takes backup
transect money
takes backup
if pin is wrong for 3 times then provide transaction after 5 minutes
only authenticated users can access data of database
return cash
provides security


Library management system
get id, password
issue book
<<include>>
return book
search book
demand for book
enter password for log-in
{log in first }
gives review
{if late returned}
log in
<<include>>
registration
log out
pay fine
collect
university
Interface1
supplies book
maintain record
takes payment
modifies record
takes backup
checks id
check conditions
assign id
store records
«extends»
«extends»
address
details
display online records
updates data
manages books
user
students
staff
supplier
librerian
server
peon

Analysis Modelling in Software Engineering

Analysis Model is a technical representation of the system. It acts as a link between system description and design model. In Analysis Modelling, information, behavior and functions of the system is defined and translated into the architecture, component and interface level design in the design modeling.

- Software analysis and design includes all activities, which help the transformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.
- Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

Requirements Analysis

Requirement Analysis results in the specification of software's operational characteristics; indicates software's interface with other system element and establishes constraints that software must need.

Throughout analysis modeling the software engineer's primary focus is on *what* not *how*.

Requirement Analysis allows the software engineer (called an *analyst* or *modeler* in this role) to:

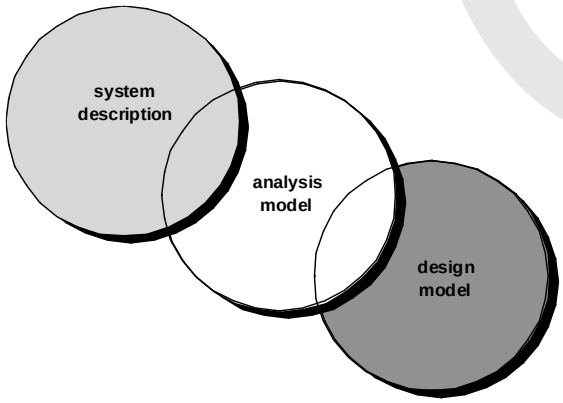
- Elaborate on basic requirements established during earlier requirement engineering tasks
- Build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.

Overall Objectives and Philosophy

The analysis model must achieve three primary objectives:

1. To describe what the customer requires
2. To establish a basis for the creation of a software design, and
3. To define a set of requirements that can be validated once the software is built.

The analysis model bridges the gap between a system-level description that describes overall functionality as it is achieved by applying software, hardware, data, human, and other system elements and a software design that describes the software application architecture.



Analysis Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and other non-functional models until design.
    - For example, a database may be required, but the classes necessary to implement it, the functions required to access it, and the behavior that will be exhibited as it is used should be considered only after problem domain analysis has been completed.
- Minimize coupling throughout the system.
    - The level of interconnectedness between classes and functions should be reduced to a minimum.
- Be certain that the analysis model provides value to all stakeholders.
    - Each constituency has its own use for the model.
- Keep the model as simple as it can be.
    - Ex: Don't add additional diagrams when they provide no new information.
    - Only modeling elements that have values should be implemented.

Domain Analysis

Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain. [Object-oriented domain analysis is] the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks.

- Define the domain to be investigated.
- Collect a representative sample of applications in the domain.
- Analyse each application in the sample.
- Develop an analysis model for the objects

Analysis Modeling Concepts and Approaches

- One view of analysis modeling, called structural analysis, considers data and the processes that transform the data as separate entities.
- Data objects are modeled in a way that defines attributes and relationships.
- Processes that manipulate data objects are in a manner that shows how they transform data as data objects flow through the system.
- A second approach to analysis modeling called ***object-oriented analysis*** focuses on the definition of classes and the manner in which they collaborate with one another to affect customer requirements.

Data Modeling Concepts

Data Objects

A data object is a representation of almost any composite information that must be processed by software. By composite, we mean something that has a number of different properties and attributes.

- "Width" (a single value) would not be a valid data object, but dimensions (incorporating height, width and depth) could be defined as object.

A data object encapsulates data only – there is no reference within a data object to operations that act on the data. Therefore, the data can be represented as a table below.

object: automobile

attribute:

- ✓ make
- ✓ model
- ✓ body type
- ✓ price
- ✓ option code

Data Attributes

Data attributesdefine the properties of a data object and take one of three different characteristics. They can be used to:

1. Name an instance of the data object.
2. Describe the instance, or
3. Make reference to another instance in another table.

In addition, one or more of the attributes, must be defined as an identifier, i.e., the identifier attribute becomes a "key" when we want to find an instance of the data object. Values for the identifier(s) are unique, although this is not a requirement.

Referring to the data object car, a reasonable identifier might be the ID number.

Relationships

Indicates "connectedness"; a "fact" that must be "remembered" by the system and cannot or is not computed or derived mechanically

- several instances of a relationship can exist
- objects can be related in many different ways

We can define a set of object/relationship pairs that define the relevant relationships. For example:

- A person *owns* a car.
- A person is *insured to drive* a car.

The relationship *owns* and *insured to drive* define the relevant connections between person and car.

Object-Oriented Analysis

The intent of Object Oriented Analysis (OOA) is to define all classes and the relationships and behavior associated with them that are relevant to the problem to be solved.

To accomplish this, a number of tasks must occur:

1. Basic user requirements must be communicated between the customer and the software engineer.
2. Classes must be defined.
3. A class hierarchy is defined
4. Object-to-object relationships should be represented.
5. Object behavior must be modeled.
6. 1 – 5 are repeated iteratively until the model is complete.

OOA builds a class-oriented model that relies on an understanding of OO concepts.

- Classes and objects
- Attributes and operations
- Encapsulation and instantiation
- Inheritance

There are also four other Analysis Modeling technics in software engineering, they are as follow:

- Class-Based Modeling
- Scenario-Based Modeling
- Flow-Oriented Modeling
- BehavioralModeling

Analysis Patterns in Software Engineering

It has been observed that the software engineer 'reuses' certain functions, classes, and/ or behavior across all the projects, which mayor may not form a part of the specific application domain. For example, features and functions described by a user interface are almost common, regardless of the application domain chosen. Analysis patterns refer to classes, functions, and other features that can be reused when developing applications within a specific application domain. The objectives of analysis patterns are listed below.

1. To quicken the requirements analysis phase by providing reusable analysis models with the description of both advantages and limitations.
2. To suggest several design patterns and feasible solutions to common problems in order to help the software designer translate an analysis model into a design model.

Analysis patterns have a unique pattern name, which allows the development team to refer to them with their pattern names. These patterns are stored in a repository so that the software engineer can refer to these patterns and reuse them while developing new software.

Information is stored in the analysis pattern, which can be viewed in the form of a template. The analysis pattern template, which is comprises the following sections.

1. **Pattern name:** Consists of important information (in the form of descriptor) about the analysis pattern. This descriptor is used within the analysis model when reference is made to a pattern.
2. **Intent:** Describes the problem that is addressed in an analysis pattern, which facilitates the software engineer to use analysis patterns in the specified application domain.
3. **Motivation:** Represents how an analysis pattern can be used to address a particular problem in the application domain with the help of a scenario.
4. **External issues (Forces) and Contexts:** Lists the external issues that affect the manner in which an analysis pattern works. The external issues include business related subjects, external technical constraints, and so on. In addition, external issues and contexts specify the external issues that are to be resolved by an analysis pattern.
5. **Solution:** Describes the procedure by which an analysis pattern is implemented in the analysis model.
6. **Consequences:** Describes the implementation effects of an analysis pattern into the application domain. It also provides a description of limitations that arises during its implementation.
7. **Design:** Describes the procedure of achieving analysis patterns by using the known design patterns.
8. **Known uses:** Provides information about the uses of an analysis pattern within the system.
9. **Related patterns:** Lists the analysis patterns that are related to the 'chosen' pattern.

Requirements Engineering From TheAgile Development Point Of View

- RE is concerned with discovering, analysing, specifying, and documenting the requirements of the system. RE activities deserve the greatest care because the problems inserted in the system during RE phase are the most expensive to remove.
- Most of the time 35% of the problems occurred in the development of challenging systems are related to the requirements phases.
- The main difference between traditional and agile development is not whether to do RE but when to do it.
- The RE processes in traditional systems focuses on gathering all the requirements and preparing the requirements specification document before going to the design phase, while the agile RE welcomes changing requirements even late in the development lifecycle.
- The processes used for agile RE vary widely depending on the application domain, the people involved and the organization developing the requirements.
- The secret of the success of agile RE is customer collaboration, good agile developers, and experienced project managers.
- RE process in Agile provides some recommendations to solve the requirements documentation problem in agile projects, to make agile methodology suitable for handling projects with critical (functional and non-functional) requirements, to allow agile teams involved in large software projects to work in parallel with frequent communications between them.

Requirements Validation and Negotiation

Purpose of Requirements Negotiation

- Unresolved conflicts reduce system acceptance
  - E.g., when only one of the contradictory requirements is implemented, the other party is demotivated
- Overall goal of negotiation
  - Gain a common and agreed-upon understanding of the requirements among all stakeholders
  - Resolving contradictory requirements of different stakeholders
    - "Shut down in case of failure" vs. "Restart in case of failure"
- Negotiation needs to be done throughout the entire requirements engineering process (not as a single step at the end!)

Performing Requirements Negotiation

- Negotiation requires systematic conflict management, which includes:
  - conflict identification by
    - directly analyzing elicited requirements
    - analyzing the requirements while documenting them
    - reviewing the requirements explicitly during validation
  - conflict analysis
  - conflict resolution
    - It can motivate or demotivate stakeholders to cooperate further
    - It is essential to involve all relevant stakeholders in conflict resolution
  - documentation of the resolution
    - Conflicts and their resolution must be documented

Purpose of Requirements Validation

- Reviewing requirements in order to discover errors or quality problems.
- Early assurance that (documented) requirements:
  - represent the actual needs and expectations of the stakeholders
  - have the necessary level of quality
  - can be approved for further development activities
    - Design, implementation, test, …
    - Especially important in client–contractor relationships

Remember: the later an error is found, the more expensive it is to correct

Relevant Quality Aspects for Validation

- Content
  - Have all relevant requirements been elicited and documented at the appropriate level of detail?
- Documentation
  - Do all documented requirements respect the predetermined guidelines for documentation and specification?
- Agreement

- Do all stakeholders concur with the documented requirements and have all known conflicts been resolved?

- Approval for further development activities should only be given if all three aspects are being fulfilled

Quality Aspect of Content:

- Content errors are present when the quality criteria for requirements are violated
- Fulfilled if no shortcoming have been detected with regard to:
  - completeness (set of all requirements)
  - completeness (individual requirements)
  - traceability
  - correctness / adequacy
  - consistency
  - no premature design decisions
  - verifiability
  - necessity

Quality Aspect of Documentation:

- Documentation errors are present when specification guidelines have been violated
- Risks of documentation errors
  - Impairment of development activities
    - Format not usable for processing
  - Misunderstanding
    - Requirements consumers do not understand the notation properly
  - Incompleteness
    - Used format does not allow representing all information
  - Overlooking requirements
    - Requirements consumers do not find information where expected
- Fulfilled if no shortcoming have been detected with regard to:
  - conformity to documentation format (e.g., template, notation)
  - conformity to documentations structure (e.g., correct section)
  - understandability (e.g., defined terminology)
  - unambiguity (e.g., only one possible interpretation)
  - conformity to documentation rules (e.g., correct use of notation syntax)

Quality Aspect of Agreement

- Agreement errors are present when the set of requirements does not or no longer represent the stakeholders' actual needs and expectations
- Fulfilled if no shortcoming have been detected with regard to:
  - agreement (all requirements agreed upon by the stakeholders)
  - agreement after changes
  - conflict resolution

Essential Validation Principles

1. Involvement of correct stakeholders
2. Separation of the identification and correction of errors

3. Validation from different views
4. Adequate change of document type
5. Construction of development artefacts
6. Repeated validation

These principles are independent of how the validation is performed.

Final!

Que still remaining:

Chapter 2 Questions:

A. MCQ's:

1. Which of the following is not a key human factor considered during agile software development?

a. Competence b. Common Focus

c. Consistency d. Self-organization

2. Which of the following is not activity of XP?

a. Planning b. Design

c. Coding d. Learning

3. Which of the following is not the phase in ASD?

a. Speculation b. Training

c. Collaboration d. Learning

Ans:

1.c 2.d 3.b

B. State True or False

1. Agility can be applied to any software process.
2. According to agile principles, the highest priority is to satisfy the customer through early and continuous delivery of valuable Software.
3. XP does not encourage the use of Class Responsibility Collaborator (CRC).
4. ASD is more concentrated on human collaboration and team self-organization.
5. DSDM suggests an iterative software process.

Ans:

1. True 2.True 3.False 4.True 5.True

C. 1 mark questions

1. State any two human factors for agile process.
2. State any two Extreme Programming Values**.**
3. What is Agility
4. What is Scrum

D. Brief Questions

1. Explain Agility Principles.
2. What is agile Process and gives examples of agile Process?
3. What is Extreme Programming?
4. Explain extreme programming process.
5. What are the extreme programming values?
6. Write short note on Industrial extreme programming (XP)?
7. What is Adaptive Software Development (ASD)?
8. What is Dynamic Systems Development Methods (DSDM)?
9. Write short note on Scrum
10. Explain agile unified process?