# Requirements Analysis and Modeling

## Requirements Determination

**Systems development** or **applications development** is a systematic process of defining, designing, testing, and implementing a new application. The four (4) fundamental phases common to all software development projects are **planning**, **analysis**, **design**, and **implementation**. In the analysis phase, all projects require systems analysts or requirements engineer. The role of a systems analyst is to gather requirements, analyze the gathered requirements, model the user needs, and create blueprints for how the system should be built. The first activity of a systems analyst is to determine the user requirements for a new system and refine them into a detailed requirements definition.

The purpose of **requirements determination** is to convert the high-level requirements defined from user requirements into a list of detailed requirements, called **requirements definition report**, that can be used as inputs for creating models such as functional model.

A **requirement** is simply a statement of what the system must perform or what characteristic it must have. User requirements are determined from discussions with the client and determine their actual needs. Then, these are refined in the design phase into *system requirements* that state the functional and non-functional requirements of the system. During analysis, requirements are written from the perspective of the needs of the user. In the design phase, the user requirements will be used to describe how the system will be implemented. Requirements can be functional or non-functional.

- **Functional requirements** – These are requirements are directly related to the process a system has to perform or data it needs to contain. For example, requirements stating that a system must have the ability to search for available products, report actual and budgeted expenses, or generate financial statements are all functional requirements. These requirements directly flow into the creation of functional, structural, and behavioral models that represent the functionality of the system. Functional, structural, and behavioral models are graphical representations that show how a system or software works.
- **Non-functional requirements** – These requirements pertain to behavioral properties that a system must have, including operational, performance, security, and cultural and political. These requirements define how a system or software is supposed to be or its system properties.
  - **Operational requirements** – These specify the operating environment(s) in which the system must perform, as well as how these might change over time. These usually refer to operating systems, system software, and information systems with which the system must interact. These might also include the physical environment, such as in what building area the system must be installed. For example, the system of a company must work over Web environment with Web browsers, or the system must be able to work with different operating system platforms such as Linux, Windows, and Mac OS.
  - **Performance requirements** – These deal with issues related to performance such as response time, capacity, and reliability of the system. For example, the system's database must be updated in real time, and the system will process and store data on approximately 5,000 customers for a total of about 2 Mb of data.
  - **Security requirements** – These address issues with security, such as who has access to the system's data and must have the ability to protect data from disruption or data loss. For example, only manager levels can change inventory items on the system within their own department; this way, the system's data will be encrypted to provide security and files will be scanned for viruses before saving them on the system to prevent the system from a potential threat.
  - **Cultural and political requirements** – These deal with issues related to the cultural and political factors and legal requirements that affect the system. For example, the system should not use any terms or icons that might offend anyone, the system should use the English language only, and personal information of customers are not transferrable outside the country. These also address the issues related to the rules of the company.

  Non-functional requirements are used primarily in design when decisions are made about the database, user interface, hardware and software, and the system's architecture. Non-functional requirements affect the decisions that will be made during the design of a system. These requirements are very important in understanding how the final system works. Requirements must define the clients' or users' needs as well as what is expected from the software or system.

### Requirements Definition Report

The **requirements definition report**, also referred to as **requirements specification document**, is a documentation that lists the functional and non-functional requirements of a system in an outline format and defines the scope of the system or software. This is used to describe what the systems can do, including its characteristics, and is used as a guide in developing the system or software. *Figure 1* shows a sample outline of the requirements definition report.

- **Revision History**

1. **Introduction**
   - **1.1. Document Purpose**
   - **1.2. Product Scope**
   - **1.3. Product Overview**
   - **1.4. Definitions**
2. **References**
3. **Specific Requirements**
   - **3.1. Functional Requirements**
   - **3.2. Non-functional Requirements**
     - **3.2.1. Operational Requirements**
     - **3.2.2. Performance Requirements**
     - **3.2.3. Security Requirements**
     - **3.2.4. Cultural and Political Requirements**
4. **Verification/Document Approvals**
5. **Appendixes**

*Figure 1.* Example outline for a requirements definition report
*Source*: Nailing Your Software Requirements Documentation, 2018

Many organizations provide industry standard templates similar to the outline in *Figure 1*. These templates may be based on the International Organization for Standardization (ISO) standards, Institute of Electrical and Electronics Engineers (IEEE) standards, or industry best practices. Software companies employ these templates or create their own outline based on the industry best practices. *Figure 2* shows a sample of specific requirements for an appointment system for a typical doctor's office.

**3. Specific Requirements**

**3.1. Functional Requirements**

**3.1.1. Manage Appointments**

REQ001. The patient shall be able to make a new appointment.
REQ002. The patient shall be able to change appointment.
REQ003. The patient shall be able to cancel appointment.

**3.1.2. Produce Schedule**

REQ004. The office manager shall be able to check daily schedule.
REQ005. The office manager shall be able to print daily schedule.

**3.1.3. Record Doctor Availability**

REQ006. The doctor shall be able to update his/her schedule.

**3.1.4. Authenticate User**

REQ007. The patient shall log in to the system by providing his/her username and password.
REQ008. The doctor and manager shall log in to the system by providing his/her employee ID and password.

**3.2. Non-functional Requirements**

**3.2.1. Operational Requirements**

REQ009. The system will operate in Windows Operating System platform.
REQ010. The system will connect to printers wirelessly.
REQ011. The system will automatically back up database into a Web server at the end of each day.

**3.2.2. Performance Requirements**

REQ012. The system shall store a new appointment in real time.
REQ013. The system shall retrieve the daily appointment schedule in less than two (2) seconds.

**3.2.3. Security Requirements**

REQ014. Only doctors shall be able to set their availability.
REQ015. Only managers shall be able to create a schedule.
REQ016. The system shall use Completely Automated Public Turing Test to Tell Computers and Humans Apart (CAPTCHA) recognition service to verify the user's login using "I'm not a robot" checkbox version of CAPTCHA.

**3.2.4. Cultural and Political Requirements**

REQ017. The system will use English language only.

*Figure 2.* Example functional and non-functional requirements in specific requirements
*Source:* Systems Analysis & Design. An Object-Oriented Approach with UML (5th ed.), 2015. p. 90

*Figure 2* contains both functional and non-functional requirements. The functional requirements include requirements of how the users—doctor, patient, and manager—will use the system. The non-functional requirements include requirements that define how the system will operate and its characteristics. The requirements are numbered in an outline format so that each requirement is clearly identified. These requirements are grouped into functional and non-functional requirements, and these requirements include headings to further group the specific requirements. It is a best practice to include the words "shall" or "will" when writing requirements like the one in *Figure 2*.

A properly written requirements definition report describes exactly what the system needs to the analysts and benefits all successive phases of the software development process such as software testing. When discrepancies arise during software or system development, this document serves as a tool for clarification.

## Requirements-Gathering Techniques

**Requirements gathering** or **requirements elicitation** is the process of cooperating with clients or users to determine what requirements are needed. When determining requirements for the requirements definition report, the systems analysts need to define how a computer system or software should operate. They need to help clients or users to discover their needs. They use **requirements-gathering techniques** to collect information and list the business or user requirements that were defined from that information. The analysts then work with the entire project team and clients to verify, change, and complete the list of requirements that were identified before moving to design.

The challenge in requirements-gathering is choosing the way(s) on how to collect information. There are many techniques for gathering requirements. Each technique has its own strengths and weaknesses, many of which are complementary, so most projects use a combination of techniques. The five (5) most commonly used techniques are the following:

- **Interviews** – This is the most commonly used requirement-gathering technique. In general, interviews comprise interviewers and interviewees.

  There are five (5) basic steps to the interview process, which are as follows:

  *Step 1.* **Selecting Interviewees** – The first step in interviewing is to create an interview schedule that lists who will be interviewed, when, and for what purpose. The people who appear on the interview schedule are selected based on the analyst's information needs. The client, users of the system, and other members of the project team can help the analyst to determine who in the organization can best provide valuable information about the requirements. These people are listed on the interview schedule in order when they should be interviewed. *Figure 3* shows a sample of an interview schedule.

| Name | Position | Purpose of Interview | Meeting |
|---|---|---|---|
| Andrea Sison | Director, Accounting | Strategic vision for the new accounting system | Monday, March 1<br>8:00 – 10:00 AM |
| Linda Aquino | Director, Human Resources | Understand reports produced for Human Resources by the current system; determine information requirements for future system | Monday, March 1<br>2:00 – 3:30 PM |
| Jennifer Dela Cruz | Manager, Accounts Receivable | Current problems with accounts receivable process; future goals | Monday, March 1<br>4:00 – 530: AM |
| Mark Torres | Manager, Accounts Payable | Current problems with accounts payable process; future goals | Wednesday, March 3<br>10:30 – 11:30 AM |
| Anne Castro | Supervisor, Data Entry | Accounts receivable and payable processes | Wednesday, March 3<br>1:00 – 3:00 PM |
| Ferdinand Cruz | Data Entry Clerk | Accounts payable and payable processes | Wednesday, March 3<br>3:30 – 5:00 PM |

*Figure 3*. Sample interview schedule
*Source:* Systems Analysis & Design. An Object-Oriented Approach with UML (5th ed.), 2015. p. 97

People at different levels of the organization have varying perspectives on the system. Therefore, it is important to include managers who manage the processes and the employee who actually performs the processes in the interview schedule to gain high-level perspective on an issue.

*Step 2.* **Designing Interview Questions** – There are three (3) types of interview questions:

- *Close-ended questions* are those that require a specific answer. These types of interview questions are used when an analyst is looking for specific information. An analyst may ask, "How many requests does a company process per day?"
- *Open-ended questions* require the interviewee to answer in open text format where they can answer based on their complete understanding and knowledge. These types of interview questions are designed to gather rich information and give the interviewee more control over the information that is revealed during the interview. The information that the interviewee discusses uncovers information that is important as the answer. For example, the analyst could ask the interviewee, "What are some improvements you would like to see in the new system?"
- *Probing questions* follow up on what has just been discussed from close-ended or open-ended questions in order to learn more. These are often used when the interviewer is unclear about an interviewee's answer. These interview questions encourage the interviewee to expand on or confirm information from a previous response. For example, after the analyst asked the "What are some improvements you would like to see in the new system?", the analyst will ask a probing question "Why do you think those improvements are necessary for the new system?"

These types of interview questions are usually combined during an interview. Whatever type of interview is conducted, interview questions must be properly organized and listed into a logical sequence in a formal list so that the interview flows well.

*Step 3.* **Prepare for the Interview** – It is important for an interviewer to prepare for the interview. The interviewer should have a general interview plan listing the questions to be asked in the appropriate order and should anticipate possible answers and provide follow-up with them. The interviewer should confirm the areas in which the interviewee knows so as not to ask questions that the interviewee cannot answer. Review the topic areas, the questions, and the interview plan, and clearly decide which have the greatest priority in case time runs short.

*Step 4.* **Conduct the Interview** – The interviewer should start with an explanation of why s/he is conducting the interview and explain why the interviewee is there. The interviewer may proceed into the planned interview questions and should take notes to record the interview. Recording ensures that the interviewer does not miss important points an s/he can focus on the interview. But before doing so, ask the interviewee whether it is appropriate for him/her to record the interview. After conducting the interview, the interviewer should briefly explain what will happen for the system.

*Step 5.* **Post-Interview Follow-up** – After the interview, the analyst must prepare an interview report that describes the information from the interview. The report must contain interview notes, information that was collected throughout the interview, and a summary in a useful format. The interview report must be sent to the interviewee with a request to read it and inform the analyst for clarifications or updates. The following figure shows a sample interview report.

**INTERVIEW NOTES**
**Approved by: Linda Aquino**

**Person Interviewed:** Linda Aquino, Director, Human Resources
**Interviewer:** Katherine Castro
**Purpose of Interview:**

- Understand reports produced for Human Resources by the current system; and
- Determine information requirements for future system.

**Summary of Interview:**

- Sample reports of all current HR reports are attached to this report. The information that is not used and missing information are noted on the reports.
- Two (2) biggest problems with the current system are as follows:
  1. The data are too old (the HR Department needs information within two (2) days of month end; currently, information is provided to them after a three-week delay); and
  2. The data are of poor quality (often reports must be reconciled with the departmental HR database).
- The most common data errors found in the current system include incorrect job level information and missing salary information.

**Open Items:**

- Get current employee roster report from May Santos (telephone extension 435).

- Verify calculations used to determine vacation time with May Santos.
- Schedule interview with Jimmy Lee (telephone extension 237) regarding the reasons for data quality problems.

**Detailed Notes:** See attached transcript.

*Figure 4.* Sample interview report
*Source:* Systems analysis & design. An object-oriented approach with UML (5th ed.), 2015. p. 101

- **Joint Application Development (JAD)** – JAD is an information-gathering technique that allows the project team, users, and management to work together to identify requirements for the system. This is the most useful method for collecting information from clients and is a structured process that involves a group of participants to meet together under the direction of a facilitator. The facilitator sets the meeting agenda and guides the discussion but does not join in the discussion as a participant. The facilitator must be an expert in both group-process techniques and systems and analysis design techniques. There must be someone called "scribes" to assist the facilitator by recording notes, making copies, and so on. Scribes can use computers and CASE tools to record information as the JAD session proceedings.

  The JAD session may take several hours, days, or weeks until all the issues have been discussed and needed information is collected. JAD sessions must take place in a special prepared meeting room away from the participant's offices to keep them from interruption and focus on the JAD session. There are five (5) basic steps to the JAD approach:

  *Step 1.* **Selecting Participants** – The process of selecting participants for a JAD session is the same as the process of selecting interviewee in the interview technique. Selecting participants must be based on the information they can contribute in order to provide a broad mixture of organization levels and build political support for the new system.

  *Step 2.* **Designing JAD Session** – The time cover of JAD session is depending on the size and scope of the project the session can. JAD sessions are used for collecting information and moving into analysis. For example, the participants and analysts can create analysis deliverables collectively, such as the functional models or requirements definition. These are also designed to collect specific information from the participants/users and require developing a set of questions before the meeting. The analysts can prepare a list of questions that can be a mixture of close-ended, open-ended, and probing questions. JAD sessions are structured so they must be carefully planned.

  *Step 3.* **Preparing for the JAD Session** – JAD sessions can go beyond the depth of a typical interview, so it is important to prepare the analysts and participants for a JAD session. It is important that the participants understand what is expected of them. For example, if the goad of the JAD session is to develop an understanding of the current system of the client, then the participants must bring procedure manuals and documents with them. Another example is if the goal of the session is to identify improvements for a system, then the participants must prepare their suggestions and ideas on how they would improve the system before the JAD session.

  *Step 4.* **Conducting the JAD Session** – JAD sessions follows a formal agenda and rules that define appropriate behavior. The common rules include following the schedule, respecting other's opinions, accepting disagreement, and ensuring that only one (1) person talks at a time. The JAD facilitator performs three (3) key functions:

  - Ensure that the group sticks to the agenda and follow the rules.
  - Help the group understand the technical terms and other terminologies in the system-development process and help the participants understand the specific analysis techniques used.
  - Record the group's input on a public display area (whiteboard or computer display, structure the information provided by the participants during the session, and help them to recognize the key issues and important solutions.

  During the session, the JAD facilitator's scribes must record notes, make copies, and so on.

  *Step 5.* **Post-JAD Follow-up** – JAD post-session report is prepared and circulated among the participants. The post-session report is essentially the same as the interview report in *Figure 3*. Because JAD sessions are longer and provide more information, it usually takes a week or more after the JAD session to create a report.

- **Questionnaires** – These are a set of written questions used to obtain information from individuals. These are often used when there is a large number of people from whom information and opinions are needed. Administering questionnaires are a common technique with systems or software intended for use outside the organization, such as customers or vendors. These can be distributed in paper form or electronic form, either through e-mail or on the Web.

  There are four (4) steps when using questionnaires as a gathering technique:

*Step 1.* **Selecting Participants** – As with interviews and JAD sessions, the first step is to identify the individuals to whom the questionnaires will be sent. When selecting participants, the standard approach is to select a sample (a subset of the population chosen to represent the population under study) of people who represents an entire group.

*Step 2.* **Designing a Questionnaire** – Questions on questionnaires must be clearly written and leave little room for misunderstanding, so closed-ended questions are mostly used. Questions must enable the analyst to separate facts from opinions. Before distributing the questionnaires, the analyst must be clear of how the collected information will be analyzed and used. Questions should be relatively consistent in style. Thus, it is generally a good practice to group related questions to make them simpler to answer.

*Step 3.* **Administering the Questionnaire** – When administering the questionnaires, the key issues are how to make sure that participants will complete the questionnaire and send it back. To improve the response rate, explain to the participants why the questionnaire is being conducted and why the respondent has been selected, stating a date by which the questionnaire is to be returned. Some analysts personally hand out the questionnaire and contact those who have not returned the questionnaire after the deadline.

*Step 4.* **Questionnaire Follow-up** – It is important to process the returned questionnaires and develop a questionnaire report soon after the questionnaire's deadline. This ensures that the analysis process proceeds in a timely fashion and that respondents who requested copies of the results receive them promptly.

- **Document Analysis** – This is often used by project teams to understand the current system (as-is system) of an organization. This is a form of qualitative research in which documents are analyzed and interpreted by the researchers to give voice and meaning around an assessment topic. The key issue in this technique is many projects or systems are not well documented because the project teams who developed the systems fail to document their projects along the way. However, many helpful documents exist in an organization that requirements analysts can review, such as paper reports, memorandums, policy manuals, user-training manuals, organization charts, forms such as registration forms, receipts, the user interface of the current system, and even their website. Document analysis helps in identifying what needs to be changed.

- **Observation** – This is the act of watching processes being performed. This is a powerful tool for gathering information about the current system (as-is system) because it enables the analyst to see the reality of a situation rather than listening to others describe it in interviews or JAD sessions. Observation is a good way to check the validity of information gathered from indirect sources such as interviews and questionnaires. In most cases, observation supports the information that users provide in interviews or JAD sessions. When it does not, it is an important sign that extra care must be taken in analyzing the business system or software.

**Selecting the Appropriate Techniques**

The five (5) requirements-gathering techniques have strengths and weaknesses, which are summarized in *Table 1*. In practice, most projects use a combination of techniques. These are the following characteristics that describe the strengths and weaknesses of each technique:

- **Type of information** – Some techniques are more suited for use at different stages of the analysis process, namely: understanding the three (3) stages: current system (as-is system), identifying improvements, or developing a new system (to-be system).
  - Interviews and JAD sessions are commonly used for the three (3) stages above.
  - The document analysis and observation are usually most helpful for understanding the as-is system – although occasionally, they provide information about current problems that need to be improved.
  - Questionnaires are often used to gather information about the as-is system as well as general information about improvements.

- **Depth of Information** – This refers to how rich and detailed the information is that the technique usually produces and the extent to which the technique is useful for obtaining not only facts and opinions but also an understanding of why those facts and opinions exist.
  - Interviews and JAD sessions are very useful for providing a good depth of rich and detailed information and helping the analyst to understand the reasons behind them.
  - Documents analysis and observation are useful for obtaining facts but little beyond that.

- Questionnaires can provide a medium depth of information, soliciting both facts and opinions with little understanding of why they exist.

- **Breadth of Information** – This refers to the range of information and information sources that can be easily collected using the chosen technique.
  - Questionnaires and document analysis are both easily capable of soliciting a wide range of information from a large number of information sources.
  - Interviews and observation require analysts to visit information sources individually and therefore takes more time.
  - JAD sessions are in the middle because many information sources are brought together at the same time.
- **Integration of Information** – One of the most challenging aspects of requirements-gathering is integrating the information from different sources or people. Combining the gathered information and attempting to resolve differences in opinions or facts are usually very time consuming because it means contacting each information source in turn, explaining the discrepancy, and attempting to refine the information when in fact it is another user in the organization who is doing so.
  - All techniques suffer integration problems to some degree, but JAD sessions are designed to improve integration because all information is integrated when it is collected during the session. The immediate integration of information is one of the most important benefits of JAD that distinguishes it from other techniques, and this is why most organizations use JAD for important projects.
- **User Involvement** – This refers to the amount of time and energy the intended users of the new system must devote to the analysis process. Involving the users in the analysis process increases the chance of success in developing the system. However, user involvement can have a significant cost, and not all users are willing to contribute valuable time and energy.
  - Questionnaires, document analysis, and observation place the least burden on users.
  - Interviews and JAD sessions require effort of the users.
- **Cost** – This defines the cost to spend when using each technique and does not imply if a technique is more or less effective than the other techniques.
  - Questionnaires, document analysis, and observation are low-cost techniques, although observation can be quite time-consuming.
  - Interviews and JAD sessions generally have moderate costs. In general, JAD sessions are much more expensive initially because they require many users and often involve highly paid consultants.

| Consideration | Interviews | Joint Application Development | Questionnaires | Document Analysis | Observation |
|---|---|---|---|---|---|
| *Type of information* | As-is, improvements, to-be | As-is, improvements, to-be | As-is, improvements | As-is | As-is |
| *Depth of information* | High | High | Medium | Low | Low |
| *Breadth of information* | Low | Medium | High | High | Low |
| *Integration of information* | Low | High | Low | Low | Low |
| *User involvement* | Medium | High | Low | Low | Low |
| *Cost* | Medium | Low to Medium | Low | Low | Low to Medium |

*Table 1.* Table of characteristics of requirements-gathering techniques
*Source:* Systems analysis & design. An object-oriented approach with UML (5th ed.), 2015. p. 108

**Combining Techniques**

In practice, requirements gathering combines a series of different techniques. Most analysts start by using interviews with senior managers to gain an understanding of the project and the key issues. From these interviews, it becomes clear whether large or small changes are anticipated. These interviews are often followed with analysis of documents and policies to gain some understanding of the as-is system. Usually, interviews are next conducted to gather the rest of information needed for the project development.

The JAD sessions are commonly used when identifying improvements because these enable the users and key stakeholders to work together through an analysis technique and come to a shared understanding of the possibilities for the new system. The JAD sessions are usually followed by questionnaires sent to a much wider set of users or potential users to see whether the opinions of those who participated in the JAD sessions are widely shared.

When developing the concept of the new system, the interviews with senior managers is conducted, followed by JAD sessions with users of all levels to make sure that the key needs of the new system are well understood.

## Requirements Analysis

After gathering data, the **requirements analysis** is conducted to determine if the specified requirements are actually required. This is the process which gathered requirements are analyzed to ensure that those requirements are actually required to solve conflicts that will affect the designing and development of the system or software.

### *How to Analyze Requirements*

Requirements should be analyzed for the characteristics of good requirements and project teams, or analysts employ a checklist as a tool to determine if each specified requirement is actually required. Requirements analysis must be performed by a team including project manager, systems analysts, developers, quality assurance analysts or software testers, and other relevant team members. Functional and non-functional requirements should organize elicited requirements before analyzing them.

### *Characteristics of a Good Requirement*

Requirements are the foundation of the system, and project planning and development is based on the elicited requirements. Therefore, it is essential that each elicited requirement has the characteristics of a good requirement. A good requirement should possess the following characteristics:

- **Necessary** – The requirement is really needed or specifies an essential factor on the system. This characteristic defines that if the requirement is removed, a deficiency will exist that cannot be achieved by other features of the system.
- **Unambiguous** – The requirement is stated that it should only be interpreted in only one (1) way. This characteristic defines that requirement is specific and easy to understand.
- **Consistent** – The requirement should not have conflicts with other requirements. Any conflicts must be determined, and overlap between requirements must be resolved.
- **Complete** – The requirement must completely describe the necessary functionality that will result to meet the user's need.
- **Singular** – The requirement statement includes only one (1) requirement with no use of conjunctions. Requirement statements with conjunctions such as "and," "or," and "but" must be reviewed carefully to see if they can be broken into singular requirements. The requirement must not contain multiple needs to avoid anomalies and for an analyst to able to analyze it independently.
- **Feasible** – The requirement is achievable within system constraints such as time, cost, legal, and available resources. This characteristic defines that the requirement is possible to be implemented.
- **Verifiable** – The requirement is verifiable if the implemented system or software can be tested to prove that the specified requirement has been met. Different testing tools are used to prove that the system or software satisfies the specified requirement. Verifiability is enhanced if the requirement is measurable by analysis, inspection, and test cases.
- **Traceable** – A requirement is traceable if it satisfies all the other characteristics of a good requirement, then the requirement must have a unique identifier to trace all changes to it throughout the development life cycle. An example of a unique identifier for a requirement can be "REQ001."

*Table 2* list the characteristics of good requirements with examples of bad and good requirements.

| Characteristics of Good Requirement | Example Description of Bad Requirements | Example Description of Good Requirements |
|---|---|---|
| ***Necessary*** | Assuming the system is online student portal mobile application:<br>*REQ001 – The student shall log in to the system by providing his/her student number and password and by scanning his/her fingerprint.*<br>This requirement may not be necessary since | *REQ001 – The student shall log in to the system by providing his/her student number and password.*<br>*REQ002 – If the student's device is equipped with fingerprint sensor, s/he shall be able to log in to the system by scanning his/her* |

| Characteristics of Good Requirement | Example Description of Bad Requirements | Example Description of Good Requirements |
|---|---|---|
| | some smartphones and computers has no built-in fingerprint sensor or the client verified that it is not necessary for their process. This requirement must be verified and resolved before designing the system or software. | *fingerprint.* |
| ***Unambiguous*** | *REQ001 - The system shall automatically refresh the databases quickly.*<br>This requirement is ambiguous since the word “quickly” is ambiguous. The specific time must specify to avoid ambiguity in the statements. | *REQ001 - The system shall automatically refresh the databases within 0.5 seconds.* |
| ***Consistent*** | *REQ001 – The system shall accept PayPal payment.*<br>*REQ002 – The system shall accept only credit card payments.*<br>These requirements show conflicts since REQ001 overlaps REQ002. Overlaps between these requirements must be resolved. | *REQ001 – The system shall accept PayPal payment.*<br>*REQ002 – The system shall accept credit card payments.* |
| ***Complete*** | *REQ001 – The employee shall log in to the system by providing his/her relevant information.*<br>In this statement, the other relevant information is not clear. This must be clearly defined to make the requirement complete. | *REQ001 - The employee shall log in to the system by providing his/her username, password, and department code.* |
| ***Singular*** | *REQ001 – The registrar shall be able to enroll a student to an undergraduate course and drop that enrolled undergraduate course.*<br>This statement contains two (2) requirements that must be broken down into singular requirements. | *REQ001 – The registrar shall be able to enroll a student to an undergraduate course.*<br>*REQ002 – The registrar shall be able to drop a student from his/her enrolled undergraduate course.* |
| ***Feasible*** | *REQ001 – The employees shall log in to the system by examining his/her deoxyribonucleic acid (DNA).*<br>This requirement may be not feasible within available resources and budget of the company or clients. | *REQ001 – The employees shall log in to the system by scanning his/her fingerprint.* |
| ***Verifiable*** | *REQ001 – The user interface of the system shall be user-friendly.*<br>This requirement statement is not testable or measurable since it’s ambiguous and not complete. This statement needs to be resolved and must be verified to the clients. This can be fixed by the adding details of how it will be tested. | *REQ001 – The system shall display the current total enrolled students on the dashboard page every two (2) seconds.*<br>*REQ002 – The user interface of the system shall use the Arial font.* |

*Table 2.* List of characteristics of good requirements with examples of bad and good requirements
*Source.* Project Workflow Management. A Business Process Approach, 2014. pp. 49–50

These are the standard characteristics that a requirement must have, but project teams can add other characteristics depending on how they will satisfy the actual requirements needed by client. For example, a project team can add a characteristic that the requirement must not conflict with the policy of the company or does the requirement align with the business goals of the company.

*Using Requirements Checklist*
A checklist is a list of questions which the analyst will use to assess each requirement. *Figure 5* shows an example of a checklist

used for analyzing single requirement. There will also be as many completed checklists as there are many requirements to analyze. The checklist must be filled in for each requirement. All questions must be answered "yes" to ensure that the requirement satisfies the specified characteristics or questions. All the requirements must be analyzed with the project team, and the identified deficiencies must be corrected.

| **Requirement Checklist** | | | |
|---|---|---|---|
| Project Name: ______________________________ | | | Client Name: ____________________ |
| Reviewed by: ______________________________ | | | Date: ____________________ |
| Requirement ID: ____________________ | | | |
| Requirement Description: ________________________________________ | | | |
| **Question** | | **Yes** | **No. Explanations (Report the requirement in the verification or restate the requirement if possible).** |
| 1 | Is the requirement detailed and adequate to develop an estimate and clear enough so the business knows what will be delivered? | ☐ | |
| 2 | Is the requirement technically achievable? | ☐ | |
| 3 | Is the requirement complete with no further clarification required? | ☐ | |
| 4 | Has it been verified that the stated requirement does not have a negative impact on other requirements and systems? | ☐ | |
| 5 | Is the requirement unique that it does not duplicate other requirements? | ☐ | |
| 6 | Is the same terminology used for similar elements in all other requirements? | ☐ | |
| 7 | Does the requirement have only one (1) interpretation? | ☐ | |
| 8 | Is the language clear enough to leave no doubt about the intended descriptive or numeric value? | ☐ | |
| 9 | Does the stated requirement use common non-technical terminology and avoid technical terminology so the client can easily understand it? | ☐ | |
| 10 | Does the requirement use standard words or phrases such as "shall" and "will" and avoid ambiguous words like "user-friendly," "flexible," "fault tolerant," "simple," "efficient," and "easy to use?" | ☐ | |
| 11 | Can the requirement be verified by inspection, analysis, demonstration, or test? | ☐ | |
| 12 | Has the requirement been assigned a unique identifier in order to follow its changes through all phases of the project? | ☐ | |
| 13 | Is the requirement clear and easy to understand, and does it state only what must be done? | ☐ | |
| 14 | Does the requirement document WHAT is required, not HOW the requirement should be met? | ☐ | |
| 15 | Does the requirement align with the business goals of the company? | ☐ | |
| 16 | Is the requirement traceable? | ☐ | |

*Figure 5.* Example requirement analysis checklist
*Source.* Project Workflow Management. A Business Process Approach, 2014. p. 60

*Figure 5* shows a simple checklist, but any project teams can create or modify their own standard checklist that benefits their system or software development. Before analyzing the requirements, it is important that the team must have their checklist to help them in the analysis process.

After analyzing the requirements, the analyzed requirements must be systematically organized to create the requirements definition report and must be reviewed by the client for approval. Creating requirements definition is an iterative process where the analysts or project team elicit information, analyze the information, then create or revise the requirements definition report. This process continues, and the requirements definition report contains revisions until the client approves the document.

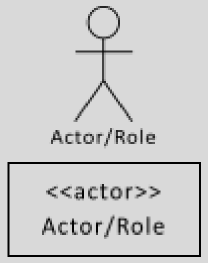
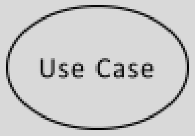
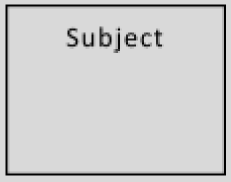
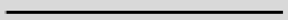
## Modeling with Unified Modeling Language (UML)

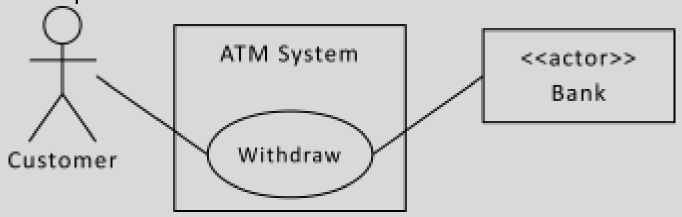
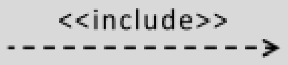
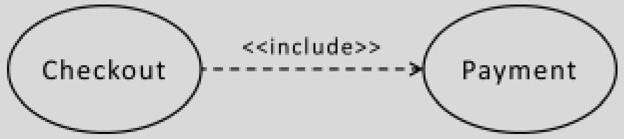
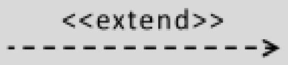
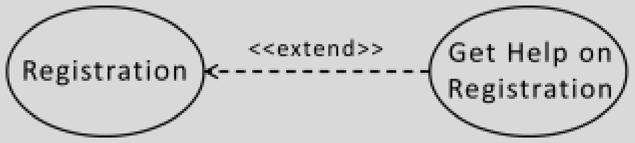
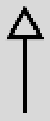
The **unified modeling language (UML)** is a visual modeling language that provides a visual means of developing and documenting object-oriented software and systems. UML is a popular approach to modeling software and systems. There are several UML diagrams providing different viewpoints and blueprints of the system. The **use-case diagram** is one of the UML diagrams.

Functional models describe business process and the interaction of an information system with its environment. Use-case diagrams can be included in requirement definition report when defining functional requirements. The purpose of a use-case diagram in UML is to summarize and demonstrate the details of functional requirements and how they are used by different users from the point of view of the user of the system.

*Components of Use-Case Diagrams*

The building components of a use-case diagram include actors, use-cases, subject boundaries, and a set of relationships among actors, actors and use-cases, and use-cases. *Table 3* describes the components of a use-case diagram where it shows a set of use-cases and each use-case representing a functional requirement.

| NOTATION | DESCRIPTION |
|---|---|
|  | ***Actor***<br>• An actor is a person or a system that represents the role of someone interacting with the system or software and is the user of the system. For example, in a library management system, one of the actors is Student who interacts with the system.<br>• The stick figure is used when the actor refers to a person type and labeled with role of the actor.<br>• When an actor represents a role played by a nonhuman, such as object, another system, or subsystem, a rectangle with <<actor>> keyword and labeled with the role is used to illustrate this type of actor.<br>• Actors can be primary or secondary actors. Primary actors initiate interaction with the system or software. Secondary actors are called upon by the system for support when the primary actor initiates an interaction with a use-case of the system.<br>• A use-case diagram can have one (1) primary actor and can also have several actors of primary and secondary actors.<br>• Actors must be placed outside the subject boundary box. They produce and access data. |
|  | ***Use-case***<br>• A use-case represents the functionality of a system.<br>• Every use-case must be labeled with a descriptive verb-noun phrase that defines a functional requirement.<br>• The use-case functionality is initiated by an actor. Therefore, every use-case is required to be associated with one (1) or more actors.<br>• Use-cases are placed inside the subject boundary box. |
|  | ***Subject boundary box***<br>• A subject boundary box represents the system's scope of which a set of use-cases are applied.<br>• The subject could be a system, software, a module/subsystem such as login system of a library management system, or an individual business process.<br>• The subject name must be placed at the top position inside the box.<br>• All use-cases outside the box are considered outside the scope of that system. |
|  | ***Association relationship***<br>• An association relationship is a line between actor and use-case. This specifies that the actor interacts with the system and uses a certain functionality. |

| NOTATION | DESCRIPTION |
|---|---|
| | • An actor can be associated to one (1) or more use-cases.<br>• It is a best practice to place the primary actors on the left side of the subject boundary box and the secondary actors must be placed on the left side of the box. This visually defines that the primary actors initiate interaction with the system and then the secondary actors provides support or service to the system.<br>• For example, in a simple **ATM system**, the primary actor **Customer** and secondary actor **Bank** are both associated on the **Withdraw** use-case. This will be easy to understand that the **Customer** actor initiates the functionality of the **Withdraw** use-case. Then, the transaction will be sent to the **Bank** actor for processing the withdraw request of the customer.<br> <br>• In a use-case diagram, it is important to know which actors are associated with which use-cases. |
|  | ***Include relationship***<br>• An include relationship is a directed relationship between two (2) use-cases when required.<br>• This represents the required use of functionality of one (1) use-case by another in order to perform its task.<br>• It is used when there are common parts of the behavior of two (2) or more use-cases.<br>• It is used to simplify large use-cases by splitting it into several use-cases.<br>• For example, a **Checkout** use-case (base case) is used and the **Payment** use-case (included) is called to make a payment. The payment feature is needed in the checkout feature. Hence, the include relationship is used.<br> <br>• This is a dashed arrow with an open arrowhead from the base use-case to the included use-case. This arrow is labeled with the keyword **<<include>>**. |
|  | ***Extend relationship***<br>• An extended relationship indicates optional functionality under a certain use-case.<br>• This defines a base use-case with extending use-case that contains optional behavior.<br>• For example, a **Registration** use-case (base) is complete and could be extended with optional **Get Help on Registration** use-case (extending), since getting help on registration is an optional feature while an actor is registering; hence, the extend relationship is used.<br> <br>• This is a dashed arrow with an open arrowhead from the extending use-case to the extended (base) use-case. This is labeled with keyword **<<extend>>**. |
|  | ***Generalization relationship***<br>• A generalization relationship represents a specialized use-case to a more generalized one. It is similar to generalization between classes. The child use-case inherits properties and behavior of the parent use-case and may override the behavior of the parent.<br>• This is used when there are common behaviors between two (2) use-cases and specialized behavior specific to each use-case. |

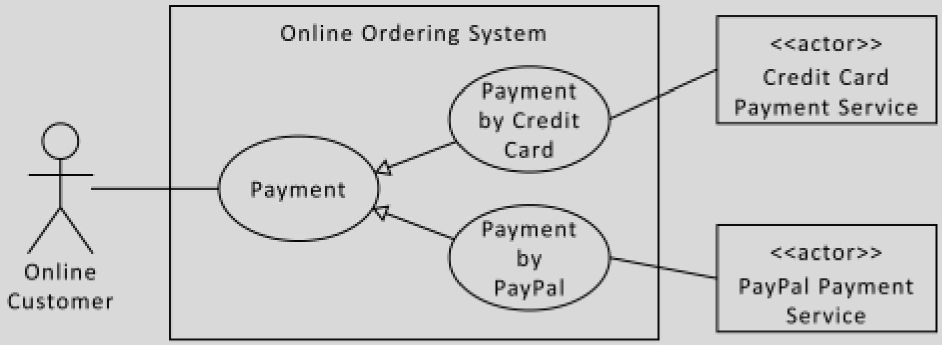
| NOTATION | DESCRIPTION |
|---|---|
| | • For example, a **Payment** use-case (parent use-case) can be generalized to **Pay by Credit Card** and **Pay by PayPal** use-cases (child use-cases). Then **Payment by Credit Card** and **Payment by PayPal** use-cases inherits the functionality of **Payment** use-case. The **Customer** actor initiates payment and can select either credit card or PayPal. Both payment methods are associated with their particular payment service. These payment services are in form of actors that provide service such as verification of customer's credit card or PayPal.   Online Ordering System<br>Payment<br>Payment by Credit Card<br>Payment by PayPal<br>Online Customer<br><<actor>> Credit Card Payment Service<br><<actor>> PayPal Payment Service • This is depicted as an arrow drawn from the child use-case to the parent use-case. |

*Table 3.* Use-case diagram components
*Source:* Systems analysis & design. An object-oriented approach with UML (5th ed.), 2015. p. 122

*Figure 6* shows an example use-case diagram for an appointment system. This use-case diagram illustrates the functional requirements defined on *Figure 2* and summarizes the basic processes that the system needs to support, which can be easily understood on the user's point of view.

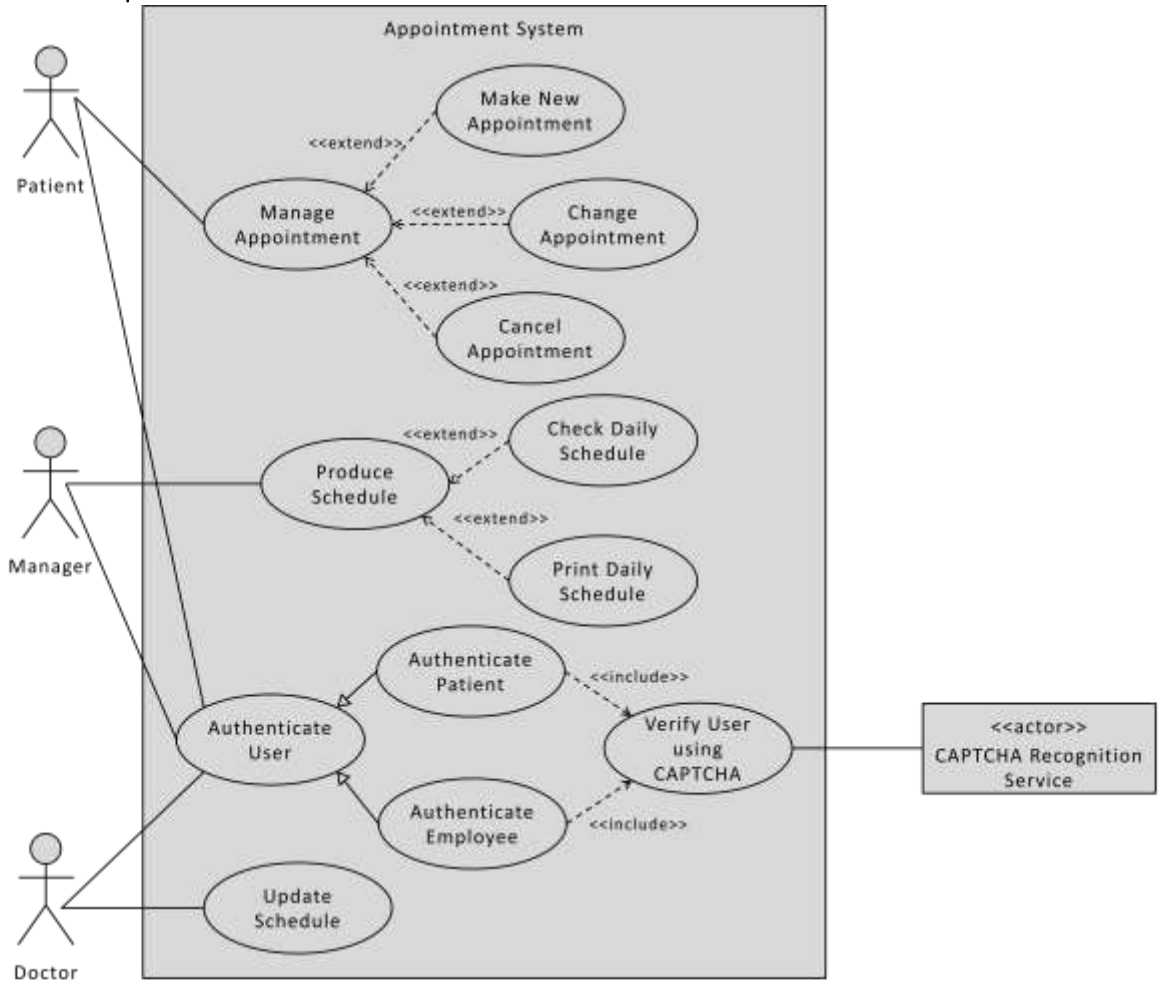


*Figure 6.* Example use-case diagram for an appointment system

*Identifying the Major Use-Cases*

It is important to identify the use-cases before creating the use-case diagram. It will be easier to create a use-case diagram once the actors, subject boundaries, and uses cases are identified and reviewed. These are the following steps in identifying major use-cases:

*Step 1.* **Review requirements definition.** This helps analysts to get a complete overview of the underlying business process being modeled.

*Step 2.* **Identify subject's boundaries.** The analysts must identify the subject's boundaries. This helps the analysts in identifying the scope of the system. However, there are changes that may occur during development process, which will then most likely change the boundary of the subject.

*Step 3.* **Identify primary and secondary actors and goals.** The analysts must identify the primary and secondary actors that will be involved on the subject and their goals. The primary actors involved with the system come from the list of clients and users. The goals represent the functionality that the subject must provide to the actor. Identifying the tasks that each actor must perform can facilitate in identifying goals. As actors are identified and their goals are uncovered, the boundary of the subject will change. Hence, Step 2 is repeated.

*Step 4.* **Identify business process and major use-cases.** The analysts must create use-cases for every identified goal of actors. Identifying the major use-cases prevents the users and analysts from forgetting the key business process and helps users explain the overall set of business processes for which they are responsible. It is important to understand and define acronyms and terminologies so that the project team and others (from outside the user group) can clearly understand the use-cases. For example, the Manage Appointments requirement from *Figure 2* is a major use-case. This requirement included the use-cases for both new patient appointment and existing patient appointment, as well as for canceling the patient appointment. Identifying use-cases is an iterative process, with users often changing their minds about what a use-case is and what it includes.

*Step 5.* **Review current set of use-cases.** It is important to review the identified set of use-cases carefully. It may be necessary to split some of them into multiple use-cases or combine some of them into a single use-case. When reviewing the current set of use-cases, a new use-case may be identified.

It is important to create a list of the identified major actors, boundaries, and use-cases. This will be the selection list when creating the use-case diagram.

*Creating a Use-Case Diagram*

The actual use-case diagram encourages the use of polymorphism or information hiding. The only major parts drawn on the use-case diagram are the system boundary, the use-cases, the actors, and the various relationships between these components. However, when use-case changes during the development process, it could affect the use-case diagram.

There are four (4) major steps in drawing a use-case diagram:

*Step 1.* **Place and draw use-cases.** Placing and drawing the use-cases on the diagram is the first thing to do. The uses cases are taken directly from identified use-cases. The generalization, extend, and include relationships can be added to connect the use-cases with relationships. There is no formal order to the use-cases, so they can be placed in whatever style is needed to make the diagram easy to read and to minimize the number of relationship lines that cross. It is necessary to redraw the diagram several times with use-cases in different positions to make the diagram easy to read.

*Step 2.* **Place and draw actors.** The second step is to place the actors in the diagram. To minimize the number of relationship lines that cross on the diagram, place the actors near the use-cases with which they are associated.

*Step 3.* **Draw subject boundary.** Drawing the subject boundary box will form the border of the subject, separating use-cases from actors.

*Step 4.* **Add associations.** The last step is to add associations by drawing lines to connect the actors to the use-cases with which they interact. It is necessary to rearrange the use-cases to help in minimizing the number of association lines that cross.

Various available tools can be used in creating use-case diagrams. Some of these tools can be installed on computers, and the others are accessed through the Internet and browser. Examples of UML tools are Microsoft Visio and Visual Paradigm.

**REFERENCES:**

Dennis, A., Wixom, B.H., & Tegarden, D. (2015). *Systems analysis & design. An object-oriented approach with UML* (5th ed.). Hoboken: John Wiley & Sons, Inc.

Epstein, D. & Maltzman, R. (2014). *Project workflow management. A business process approach*. Retrieved from https://books.google.com.ph/books?id=_ePJCgAAQBAJ&printsec=frontcover#v=onepage&q&f=false

Nailing Your Software Requirements Documentation (2018). In *Lucidchart.* Retrieved from https://www.lucidchart.com/blog/software-requirements-documentation

O'Regan, G. (2017). *Concise guide to software engineering: From fundamentals to application methods*. Cham, Switzerland: Springer.

Seidl, M., Scholz, M., Huemer, C. & Kappel, G. (2015). *UML @ classroom. An introduction to object-oriented modeling.* Cham, Switzerland: Springer International Publishing.

Use-case Diagrams Reference (n.d.). In *UML Diagrams*. Retrieved from https://www.uml-diagrams.org/use-case-reference.html

Use-case Diagram Symbols (n.d.). In *Requirements Quest*. Retrieved from https://requirementsquest.com/lesson/use-case-diagram-symbols/