**Requirements Engineering — Elicitation & Analysis**

# Requirements Elicitation & Analysis

It’s a process of interacting with customers and end-users to find out about the domain requirements, what services the system should provide, and the other constraints.

*Domain requirements reflect the environment in which the system operates so, when we talk about an application domain we mean environments such as train operation, medical records, e-commerce etc.*

It may also involve different kinds of stockholders; end-users, managers, system engineers, test engineers, maintenance engineers, etc.

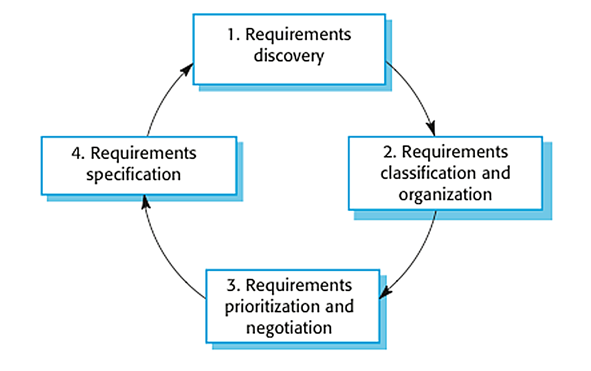
*A stakeholder is anyone who has direct or indirect influence on the requirements.*

## The requirements elicitation and analysis has 4 main process

We typically start by gathering the requirements, this could be done through a general discussion or interviews with your stakeholders, also it may involve some graphical notation.

Then you organize the related requirements into sub-components and prioritize them, and finally, you refine them by removing any ambiguous requirements that may arise from some conflicts.

Here are the 4 main processes of requirements elicitation and analysis.



The process of requirements elicitation and analysis — From [Software Engineering, 9th edition, Chapter 4](https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Presentations/index.html), by [Ian Sommerville](https://sites.google.com/site/iansommerville/home)

It shows that it’s an iterative process with feedback from one activity to another. The process cycle starts with requirements discovery and ends with the requirements document. The cycle ends when the requirements document is complete.

# 1. Requirements Discovery

It’s the process of interacting with, and gathering the requirements from, the stakeholders about the required system and the existing system (if exists).

It can be done using some techniques, like interviews, scenarios, prototypes, etc, which help the stockholders to understand what the system will be like.

## Gathering and understanding the requirements is a difficult process

That’s because stakeholders may not know what exactly they want the software to do, or they may give unrealistic requirements.

They may give different requirements, which will result in conflict between the requirements, so we have to discover and resolve these conflicts.

Also, there might be some factors that influence the stakeholder’s decision, like, for example, managers at a company or professors at the university want to take full control over the management system.

## Interviews

In Interviews, requirements engineering teams put the questions to the stakeholder about the system that’s currently used, and the system to be developed, and hence they can gather the requirements from the answers.

The questions fall under two categories:

1. Closed-Ended Questions: A pre-defined set of questions.
2. Open-Ended questions: There is no pre-defined expected answer, they are more generic questions. It’s used to explore issues that are not clear in a less structured way.

In practice, interviews usually use a mixture of both. Usually, start with the open-ended questions, and then use the closed-ended questions to be more specific about some requirements that aren’t clear yet.

Interviews are good to get an overall understanding of what stakeholders need, how they might interact with the new system and the difficulties they face with the current system.

However, interviews aren’t so helpful in understanding the domain requirements. This is for two reasons:

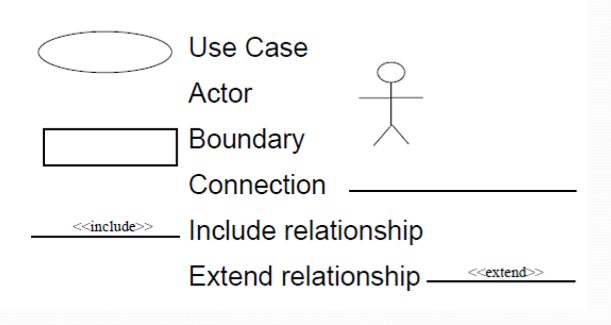
1. Domain requirements may be expressed using special domain terminologies, and software engineers often find it difficult to understand and it’s easy for them to misunderstand.
2. Sometimes stakeholders won’t tell you some requirements because they assume it’s so fundamental and it doesn’t worth mentioning, or they find it difficult to explain, which won’t be taken into consideration in the requirements.

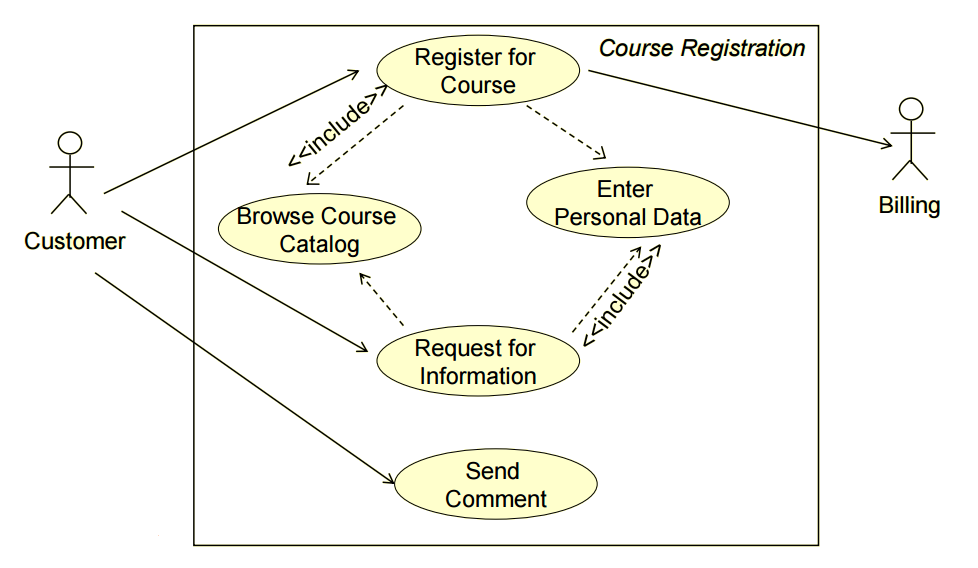
## Use Cases & Scenarios

The use cases and scenarios are two different techniques, but, usually, they are used together.

Use cases identify interactions between the system and its users or even other external systems (using graphical notations), while a scenario is a textual description of one or more of these interactions.

Use case involves some symbols to describe the system:





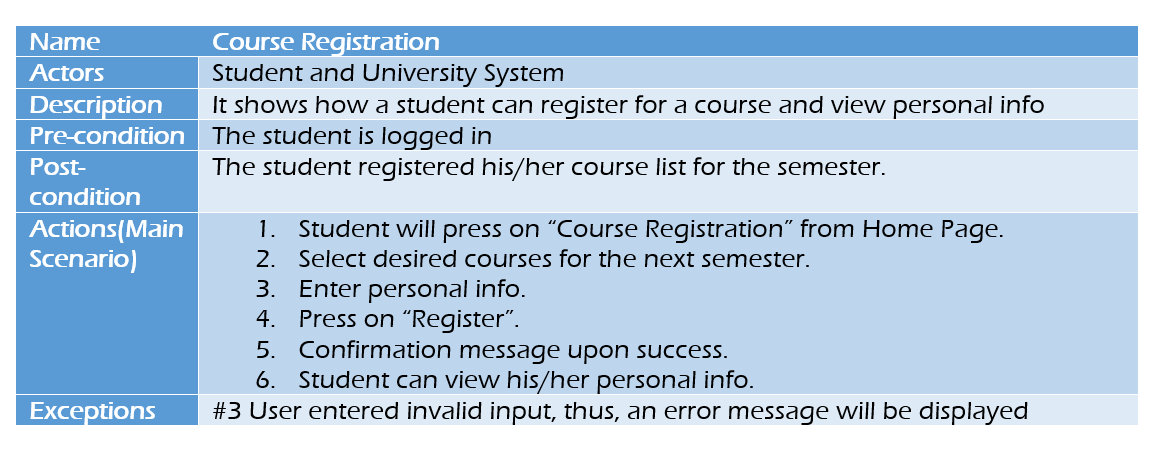
Use case diagram symbols and an example

1. **Actors**: Are those who interact with the system; human or other systems
2. **Interaction (Use Case)**: It denotes the name of the interaction (verb). It’s represented as a named ellipse.
3. **Connection**: Lines that link between the actors and the interactions.
4. **Include** **Relationship**: It denotes a connection between two interactions when an interaction is invoked by another. As an example, splitting a large interaction into several interactions.
5. **Exclude** **Relationship**: It denotes a connection between two interactions when you want to extend an interaction by adding an optional behavior, but you can use the main interaction on its own without the extending interaction.

Now, we are going to use scenarios to describe the interactions in each use case textually. They should have a format and include the following:

1. A description of the initial situation.
2. A description of the flow of the events or interactions with the system
3. A description of the exceptions, or in other words, what can go wrong, and how it can be handled.
4. Any concurrent activities should be mentioned
5. A description of the final state.

Here is the example for a scenario for the use case example above.



Scenario

Use cases and scenarios are effective techniques for eliciting the requirements. But, because they focus on the interactions with the system, they aren’t effective for eliciting high-level business, non-functional, or domain requirements.

The next two phases are about analyzing requirements: determining whether the stated requirements are clear, complete, consistent, and unambiguous, group-related requirements and organizing them into related components, and resolving any apparent conflicts.

# 2. Requirements Classification & Organization

It’s very important to organize the overall structure of the system.

Putting related requirements together, and decomposing the system into sub-components of related requirements. Then, we define the relationship between these components.

What we do here will help us in the decision of identifying the most suitable architectural design patterns.

# 3. Requirements Prioritization & Negotiation

We previously explained why eliciting and understanding the requirements is not an easy process.

One of the reasons is the conflicts that may arise as a result of having different stakeholders involved. Why? because it’s hard to satisfy all parties if it’s not impossible.

This activity is concerned with prioritizing requirements and finding and resolving requirements conflicts through negotiations until you reach a situation where some of the stakeholders can compromise.

We shouldn’t reach a situation where a stakeholder is not satisfied because his requirements are not taken into consideration.

Prioritizing your requirements will help you later to focus on the essentials and core features of the system, so you can meet the user expectations. It can be achieved by giving every piece of function a priority level. So, functions with higher priorities need higher attention and focus.

# 4. Requirements Specification

The requirements are then documented.