Negotiating and Validating Requirements

Negotiating requiremnts

- The inception, elicitation, and elaboration tasks in an ideal requirements engineering setting determine customer requirements in sufficient depth to proceed to later software engineering activities. You might have to negotiate with one or more stakeholders.
- Most of the time, stakeholders are expected to balance functionality, performance, and other product or system attributes against cost and time-to-market.

Contd.,

- The goal of this discussion is to create a project plan that meets the objectives of stakeholders while also reflecting the real-world restrictions (e.g., time, personnel, and budget) imposed on the software team.
- The successful negotiations aim for a “win-win” outcome. That is, stakeholders, benefit from a system or product that meets the majority of their needs, while you benefit from working within realistic and reasonable budgets and schedules.

Contd.,

- At the start of each software process iteration, *Boehm* defines a series of negotiating actions. Rather than defining a single customer communication activity, the following are defined:
- Identifying the major stakeholders in the system or subsystem.
- Establishing the stakeholders' "win conditions."
- Negotiation of the win conditions of the stakeholders in order to reconcile them into a set of win-win conditions for all people involved.

Validating Requirements

- Each aspect of the requirements model is checked for consistency, omissions, and ambiguity as it is developed.
- The model's requirements are prioritized by stakeholders and bundled into requirements packages that will be implemented as software increments.

The following questions are addressed by an examination of the requirements model:

- Is each requirement aligned with the overall system/product objectives?
- Were all requirements expressed at the appropriate level of abstraction? Do some criteria, in other words, give a level of technical information that is inappropriate at this stage?
- Is the requirement truly necessary, or is it an optional feature that may or may not be critical to the system's goal?
- Is each requirement well defined and unambiguous?
- Is each requirement attributed? Is there a source noted for each requirement?
- Are there any requirements that conflict with others?

- Is each requirement attainable in the technical environment in which the system or product will be housed?
- Is each requirement, once implemented, testable?
- Does the requirements model accurately represent the information, functionality, and behaviour of the system to be built?
- Has the requirements model been “partitioned” in such a way that progressively more detailed information about the system is exposed?
- Have requirements patterns been used to reduce the complexity of the requirements model?
- Have all patterns been validated properly? Are all patterns in accordance with the requirements of the customers?

- **Requirements Modeling in Software Engineering: Classes, Functions & Behaviors**

Requirements Modeling

- Requirements modeling in software engineering is essentially the planning stage of a software application or system. Generally, the process will begin when a business or an entity (for example, an educational institution) approaches a software development team to create an application or system from scratch or update an existing one.
- Requirements modeling comprises several stages, or '**patterns**': scenario-based modeling, data modeling, flow-oriented modeling, class-based modeling and behavioral modeling. Each of these stages/patterns examines the same problem from a different perspective.

Identifying Requirements

- Requirements in this context are the conditions that a proposed solution or application must meet in order to solve the business problem.
- Identifying requirements is not an exclusively technical process, and initially involves all the stakeholders, like the representatives of the entity that has commissioned the software project, who may not necessarily be from a technical background, as well as the software developers, who are not necessarily the technical team.
- Together, they discuss and brainstorm about the problem, and decide what functions the proposed application or system must perform in order to solve it.

Functional vs. Non-Functional Requirements

- A **functional requirement** specifies something that the application or system should do. Often, this is defined as a behavior of the system that takes input and provides output.
- For example, a traveler fills out a form in an airline's mobile application with his/her name and passport details (input), submits the form, and the application generates a boarding pass with the traveler's details (output).

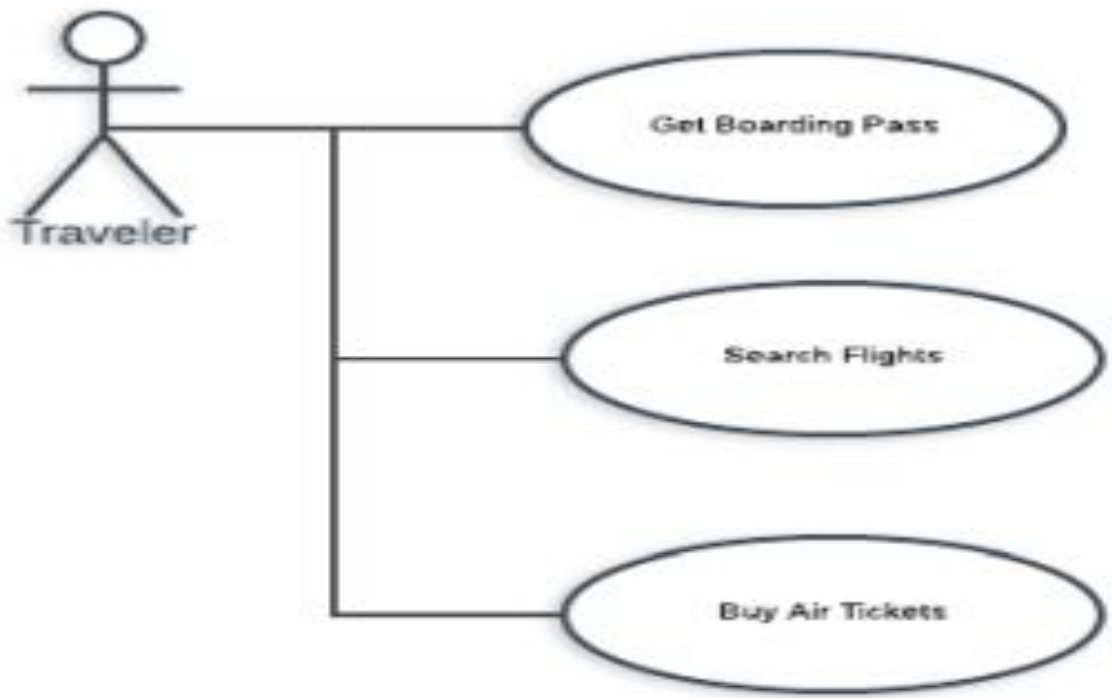
Contd.,

- **Non-functional requirements**, sometimes also called quality requirements, describe how the system should be, as opposed to what it should do.
- Non-functional requirements of a system include performance (e.g., response time), maintainability and scalability, among many others. In the airline application example, the requirement that the application must display the boarding pass after a maximum of five seconds from the time the traveler presses the 'submit' button would be a non-functional requirement.

Working with Requirements Modeling

Scenario-based Modeling

- **Scenario-based modeling's** primary objective is to look at a system from the user's perspective and produce a use case, an example instance of the user interacting with the system, like the traveler using the airline application to generate their boarding pass. It makes sense to start with this step as the other requirements modeling stages/patterns will make reference to this use case.
- Below is a use case diagram depicting three possible use cases of the traveler using the airline application. The first is the one in the example, getting a boarding pass. The following two are searching for flights and buying air tickets.


Traveler
Get Boarding Pass
Search Flights
Buy Air Tickets

Class-based Modeling

- **Class-based modeling identifies** classes, attributes and relationships that the system will use. In the airline application example, the traveler/user and the boarding pass represent classes. The traveler's first and last name and travel document type represent attributes, characteristics that describe the traveler class.
- The relationship between traveler and boarding pass classes is that the traveler must enter these details into the application in order to get the boarding pass and that the boarding pass contains this information along with other details, like the flight departure gate, seat number, etc.

Data Modeling

- **Data modeling** essentially works with the same elements as class-based modeling (object, attributes, relationships), but uses the information to produce a detailed model, termed a physical model, of what the database structure will be that will hold all the data; for example, the name, passport number and other details of the travelers that use the airline application.