Preparation with Product Owner

# Structure

* RE process followed
* IEEE830
* Appendix

# Requirements Elicitation Log/Process

* Process
* Techniques (prioritization, interviews, ...)
* Schedule (meetings, ....)
* Validation
* Negotiation
* Management

# IEEE830

Revision history

Document approval

1. Introduction (Purpose, Scope, Definitions, References, Overview)

2. General description (Product perspective, Product functions, User characteristics, General constraints, Assumptions and dependencies)

3. Specific requirements

3.1 External interface requirements (User Interfaces, Hardware Interfaces, Software Interfaces, Communications Interfaces)

3.2 Functional requirements (Functional Requirement or Feature #1, Functional Requirement or Feature #2, ...)

3.3 Use cases (Use Case #1, Use Case #2, ...)

3.4 Classes / objects (Class / Object #1, Class / Object #2, ...)

3.5 Non-functional requirements (Performance, Reliability, Availability, Security, Maintainability, Portability)

3.6 Inverse requirements

3.7 Design constraints

3.8 Logical database requirements

3.9 Other requirements

4. Analysis models

5. Change management process

a. Appendices (Sequence diagrams, Data flow diagrams (dfd), State-transition diagrams (std), ...)

# Appendix

* **Goals (AND/OR diagram)**
* **Stakeholders & Other sources or requirements**
* **Use cases & Scenarios**
* Activity diagrams
* Class diagram
* State diagrams
* Requirements attributes
* Tools used
* Mocks of the UI

# Goals

## Template

*This subsection should:*

*(1) Identify the software product(s) to be produced by name; for example, Host DBMS, Report Generator, etc*

*(2) Explain what the software product(s) will, and, if necessary, will not do*

*(3) Describe the application of the software being specified. As a portion of this, it should:*

*(a) Describe all relevant benefits, objectives, and goals as precisely as possible. For example, to say that one goal is to provide effective reporting capabilities is not as good as saying parameter-driven, user-definable reports with a 2 h turnaround and on-line entry of user parameters.*

*(b) Be consistent with similar statements in higher-level specifications (for example, the System Requirement Specification) , if they exist. What is the scope of this software product.*

## Example

This software system will be a Web Publishing System for a local editor of a regional historical society. This system will be designed to maximize the editor’s productivity by providing tools to assist in automating the article review and publishing process, which would otherwise have to be performed manually. By maximizing the editor’s work efficiency and production the system will meet the editor’s needs while remaining easy to understand and use.

More specifically, this system is designed to allow an editor to manage and communicate with a group of reviewers and authors to publish articles to a public website. The software will facilitate communication between authors, reviewers, and the editor via E-Mail. Preformatted reply forms are used in every stage of the articles’ progress through the system to provide a uniform review process; the location of these forms is configurable via the application’s maintenance options. The system also contains a relational database containing a list of Authors, Reviewers, and Articles.

# Use case

## Template

*This subsection should:*

*(1) Identify the software product(s) to be produced by name; for example, Host DBMS, Report Generator, etc*

*(2) Explain what the software product(s) will, and, if necessary, will not do*

*(3) Describe the application of the software being specified. As a portion of this, it should:*

*(a) Describe all relevant benefits, objectives, and goals as precisely as possible. For example, to say that one goal is to provide effective reporting capabilities is not as good as saying parameter-driven, user-definable reports with a 2 h turnaround and on-line entry of user parameters.*

*(b) Be consistent with similar statements in higher-level specifications (for example, the System Requirement Specification) , if they exist.What is the scope of this software product.*

## Example

2.2.3 Reviewer Use Case

**Use case: Submit Review**

**Diagram:**

Reviewer

Submit Review

**Brief Description**

The reviewer submits a review of an article.

**Initial Step-By-Step Description**

Before this use case can be initiated, the Reviewer has already connected to the Online Journal Website.

1. The Reviewer chooses the *Email Editor* button.
2. The System uses the *sendto* HTML tag to bring up the user’s email system.
3. The Reviewer fills in the Subject line and attaches the file as directed and emails it.
4. The System generates and sends an email acknowledgement.

**Xref:** Section 3.2.2, Communicate

|  |  |
| --- | --- |
| **Use Case Name** | Communicate |
| **XRef** | Section 2.2.2, Submit Article; Section 2.2.3, Submit Review  SDD, Section 7.2 |
| **Trigger** | The user selects a *mailto* link. |
| **Precondition** | The user is on the *Communicate* page linked from the Online Journal Main Page. |
| **Basic Path** | This use case uses the *mailto* HTML tag. This invokes the client email facility. |
| **Alternative Paths** | If the user prefers to use his or her own email directly, sufficient information will be contained on the Web page to do so. |
| **Postcondition** | The message is sent. |
| **Exception Paths** | The attempt may be abandoned at any time. |
| **Other** | None |

**3.3 Identifying the Actors**

Often, people find it easiest to start the requirements elicitation process by identifying the actors. The following questions can help you identify the actors of your system (Schneider and Winters, 1998):

• Who uses the system?

• Who installs the system?

• Who starts up the system?

• Who maintains the system?

• Who shuts down the system?

• What other systems use this system?

• Who gets information from this system?

• Who provides information to the system?

• Does anything happen automatically at a present time?

**3.4 Identifying the Use Cases**

Then, the scenario-based requirements elicitation process continues by asking what outwardly visible, measurable result of value that each actor desires. The following questions can be asked to identify use cases, once your actors have been identified (Schneider and Winters, 1998):

• What functions will the actor want from the system?

• Does the system store information? What actors will create, read, update or delete this information?

• Does the system need to notify an actor about chances in the internal state?

• Are there any external events the system must know about? What actor informs the system of those events?

**3.5 Identifying the Boundary**

It is important to clearly define the boundary of your system. Things inside the boundary of the system are things you need to worry about creating. In a UML use case diagram, the system boundary is denoted by a rectangle, as in Figure 3.

**3.6 Use Case Diagram**

A use case diagram is a visual representation of the relationships between actors and use cases together that documents the system’s intended behavior. A simple use case diagram is shown below in Figure 3.

Arrows and lines are draw between actors and use cases and between use cases to show their relationships. We will discuss these relationships more later on in the chapter. The default relationship between an actor and a use case is the «communication» relationship, denoted by a line with a small circle. For example, the actor in Figure 3 is communicating with the use case.

**Figure 3: A UML use case diagram**

Use case diagrams are often developed incrementally. When you feel that you are done with your use case diagram, any remaining actors that do not communicate with any use cases should be removed from your system.

**4 Use Case Flow-of-Events**

The use case diagram is important for visualizing a system. However, a textual description of the sequence of transactions of a use case is also needed for us to understand what really happens in a use case. In this section, we will use the use case flow-of-events, a description of what the system should do. The flow-of-events is written in terms of what the system should do, not how the system does it.

**4.1 Templates for a Use Case Flow of Events**

Many different templates are available for writing a use case flow of events. The exact structure of these templates can vary slightly from author to author. In this book, we use the format that was described by Quantrani (Quatrani, 1998). This template is shown in Figure 4 followed by an example of a completed flow of events for the Simulate a Configuration use case.

**X Flow-of-Events for the <name> Use Case**

*X.1 Preconditions. What needs to happen (in another use case) before this use case can start? What state must the system be in before the use case? X.2 Main Flow. The main flow is a series of declarative steps. X.3 Sub-flows. Sub-flows break down the main flow and other sub-flows to improve document readability. X.4 Alternative Flows. The alternative flows define exceptional behavior that can interrupt the normal flow. Often alternative flows indicate what is to be done under error conditions. To determine alternative flows, ask yourself, “What could possibly go wrong?” for each of the actions in the main flow and the sub-flows.*

*Note: X is a unique identifier for each use case.*

**Figure 4: Use Case Flow-of-Events Template**

* 1. **An Example Flow of Events**

Below is an example flow-of-events for the Simulate a Configuration use case. The example uses the template of Figure 4 to structure the flow of events.

UC8 Flow of Events for the Buy House Use Case 8.1 Preconditions: 1. It is the player’s turn. 2. The player has not rolled the dice. 3. The player has monopoly on one or more color groups. 8.2 Main Flow: When a player has all the tradable cells in a color group, this player is said to have monopoly on the color group. A player may build house(s) in the property cells in the color groups the player has monopoly on by pressing the Buy House button before he or she rolls the dice [S1] [E1 – E2]. The price of the house is determined by the cell. After buying the house(s), the status of the player is updated and displayed on the game board [UC13]. 8.3 Subflows: [S1] When the Buy House button is clicked, the Buy House dialog shows up. The player

selects the monopoly color group and the number of houses from that dialog. After clicking on OK in the dialog box, the player pays the fee, and the houses are created. All the property cells in the selected color group have the same number of houses. 8.4 Alternative Flows: [E1] Nothing happens if the player does not have enough money. [E2] The player can build at most five houses in a cell.

Let us now dissect this flow of events.

• The use case precondition indicates that before the use case can begin, it must be the player (who wants to buy a house)’s turn. The player has not rolled the dice, and the player must have a monopoly by owning all properties in a color group.

• The main flow lists the sequence of events.

o When a main flow or sub-flow has an event marked such as [Sx], this

indicates that a sub-flow of this use case must be “run.” When that sub- flow completes, “control” is passed back. For example, the buy house dialog shows up [S1]. Once the dialog box is clicked, control is passed back to the main use case and the house is purchased. o When a main flow or sub-flow has an event marked such as [Ex], this

indicates that an exceptional condition might occur. If it does occur, the appropriate alternative flow explains how the situation should be handled. For example, if the player does not have enough money or has more than five houses [E1-E2], the buy house dialog will not show up. o When a main flow or sub-flow has an event marked such as [UCx], this

indicates that another use case must be “run.” When that use case completes, “control” is passed back to this use case. For example, once the house purchase is complete, the status of the player is updated and displayed. [UC13]

• The sub-flows list individual sequences of the main flow. Sub-flows can also handle the “calling” of other use cases, other sub-flows, and alternative flows similarly to the main flow.

• Alternative flows list individual sequences of how exceptional situations should be handled.

• All sub-flows and all alternative flows must be “called” from the main flow or from sub-flows(s) by an indication such as [Sx] or [Ex]. If they are not called, they have no purpose because they can never be executed.

4.3 A Scenario as One Flow Through a Flow of Events As we said, multiple scenarios are handled by one use case. Consider the following two scenarios of this use case.

*The player has all the tradable cells in a color group and wants to buy a house for the color group. The player has enough money to buy the house and is shown the number of houses own in that group [S1], and purchases the house. The player’s status is displayed [UC13]. The player has all the tradable cells in a color group and wants to buy a house for the color group. The player does not have enough money to buy the house. The player’s status is displayed [UC13]. Both of these scenarios and a multitude of others are represented with this use case. A scenario is just one flow through the use case flow-of events.*