**Use Case Specifications template**

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case Identification and History | | | |
| **Use Case ID:** | PROJ.UC.1.1.1 <Assign a unique name to each use case> | | |
| **Use Case Name:** | <Name – concise results oriented-name with an action verb and a noun. > | **Version No:** |  |
| **End Objective:** | < The directly observable purpose of this use case > | | |
| **Created by:** |  | **On (date):** |  |
| **Last Update by:** |  | **On (date):** |  |
| **Approved by:** |  | **On (date):** |  |
| **User/Actor:** | <Description of the person who uses the system to accomplish tasks> | | |
| **Business Owner Name:** |  | **Contact Details:** |  |
| **Trigger:** | <Who (system or user) triggers this use case> | | |
| **Frequency of Use:** | <How often does this series of activities occurs> | | |

|  |
| --- |
| Preconditions |
| <What is true of the system state before this flow of actions begins> |

|  |  |  |
| --- | --- | --- |
| **Basic Flow** <The optimal or normal ("good day") flow of events. The basic flow of events should describe the events that walk through a successful scenario. The basic flow should not include “and/if scenarios”> | | |
| **Step** | **User Actions** | **System Actions** |
| **1** | <Phrase saying what user does> | <Description of system response> |
| **2** | <Repeated as needed> | <Repeat…> |

|  |  |  |
| --- | --- | --- |
| Alternate Flow <may be more than one> | | |
| **Step** | **User Actions** | **System Actions** |
| **1** |  |  |
| **2** |  |  |

|  |  |  |
| --- | --- | --- |
| Exception Flow <identify system and data error conditions that could occur for each step in the normal and alternate flow> | | |
|  |  |  |
| 2 |  |  |

|  |
| --- |
| Post conditions |
| 1. <What is true of the system when the flow of activities finishes> |

|  |
| --- |
| Includes or Extension Points |
| 1. <Common functionality that appears in multiple use cases can be split out into separate use cases. Provide reference to such of the use cases that are called by the subject use case. > |

|  |
| --- |
| Special Requirements |
| 1. <Identify any special non-functional requirements such as legal, performance, etc. that need to be considered during design or implementation. These requirements should only be documented here if they are specific to this use case. If the requirements span across multiple use cases, document in the appropriate section of the Systems Requirements Specification> |

|  |
| --- |
| Business Rules |
| 1. <Identify any business rules or constraints particular to this specific use case. Example of a business rule would be: “When an Account of a subscription has a Credit Card on File, all subscriptions under that account rollover month-to-month.”> |

|  |
| --- |
| Other Notes (Assumptions, Issues,) |
| < Any special considerations that need to be kept in mind for this use case only; identify the type of item with a tag like   * **Assumptions:** * **Issues:** |

Use Case Specific Checks

Use Case Diagram

* The introduction section of the use-case diagram provides a clear, concise overview of the purpose and functionality of the system.
* The use case diagram clearly presents the behavior of the system; it is easy to understand what the system does by reviewing the diagram.
  + 1. No long chains of include and extend relationships, such as when an included use case is extended, or when an extended use case includes other use cases.  These can obscure comprehensibility.
    2. Minimal cross-dependencies where an included, extending, or specialized use case must know about the structure and content of other included, extending or specialized use cases.
* All use cases have been identified; the use cases collectively account for all required behavior.
* All functional requirements are mapped to at least one use case.
* All non-functional requirements that must be satisfied by specific use cases have been mapped to those use cases.
* The use-case diagram contains no extra system behavior; all use cases can be justified by tracing them back to a functional requirement.
* All relationships between use cases are required (i.e. there is justification for all include-, extend-, and generalization-relationships).

**Actors**

* 1. Have you found all the actors? That is, have you accounted for and diagramed all roles in the system's environment? Although you should check this, you cannot be sure until you have found and described all the use cases.
  2. Is each actor involved with at least one use case? Remove any actors not mentioned in the use-case descriptions, or any actors without communicates-associations with a use case. However, an actor mentioned in a use-case description is likely to have a communicates-association with that particular use case.
  3. Can you name at least two people who would be able to perform as a particular actor? If not, check if the role the actor diagrams is part of another one. If so, you should merge the actor with another actor.
  4. Do any actors play similar roles in relation to the system? If so, you should merge them into a single actor. The communicates-associations and use-case descriptions show how the actors and the system interrelate.
  5. Do two actors play the same role in relation to a use case? If so, you should use actor-generalizations to diagram their shared behavior.
  6. Will a particular actor use the system in several (completely different) ways or does he have several (completely different) purposes for using the use case? If so, you should probably have more than one actor.
  7. Do the actors have intuitive and descriptive names? Can both users and customers understand the names? It is important that actor names correspond to their roles. If not, change them.

**Use Case Specifications**

* Is each concrete use case involved with at least one actor? If not, something is wrong; a use case that does not interact with an actor is not required, and you should remove it.
* For an included use case: does it make assumptions about the use cases that include it? Such assumptions should be avoided, so that the included use case is not affected by changes to the including use cases.
* Do any use cases have very similar behaviors or flows of events? If so - and if you wish their behavior to be similar in the future - you should merge them into a single use case. This makes it easier to introduce future changes. Note: you must involve the users if you decide to merge use cases, because the users, who interact with the new, merged use case will probably be affected.
* Has part of the flow of events already been diagramed as another use case? If so, you can have the new use case use the old one.
* Is some part of the flow of events already part of another use case? If so, you should extract this subflow and have it be used by the use cases in question. Note: you must involve the users if you decide to "reuse" the subflow, because the users of the existing use case will probably be affected.
* Should the flow of events of one use case be inserted into the flow of events of another? If so, you diagram this with an extend-relationship to the other use case.
* Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage? If not, you change their names.
* Do customers and users alike understand the names and descriptions of the use cases? Each use-case name must describe the behavior the use case supports.
* Does the use case meet all the requirements that obviously govern its performance? You must include any (nonfunctional) requirements to be handled in the object diagrams in the use-case Special Requirements.
* Does the communication sequence between actor and use case conform to the user's expectations?
* Is it clear how and when the use case's flow of events starts and ends?
* Behavior might exist that is activated only when a certain condition is not met. Is there a description of what will happen if a given condition is not met?
* Are any use cases overly complex? If you want your use-case diagram to be easy to understand, you might have to split up complex use cases.
* Does a use case contain disparate flows of events? If so, it is best to divide it into two or more separate use cases. A use case that contains disparate flows of events will be very difficult to understand and to maintain.
* Is it clear who wishes to perform a use case? Is the purpose of the use case also clear?
* Are the actor interactions and exchanged information clear?
* Does the brief description give a true picture of the use case?

**References**

* Rational Unified Process Use Case Checkpoints
* Wiegers, K., “Software Requirements”, Microsoft, 1999