USE CASES & REQUIREMENT
Lecture 2
CS-463
Umair Javed

Use Case Analysis

• Use case (Stories of using the system) is a technique to understand requirements
• Mainly deal with the functional requirements of the system

Use Case Analysis

• What is a Scenario?
  Specific sequence of actions and interactions between actor(s) and system (= one story; success or failure)
• Use Case is a collection of related success and failure scenarios describing the actors attempts to support a specific goal
• The UP defines the use cases with in the requirements discipline.

Use Cases Describe Function not Form

• Use Cases describe WHAT the system will do, but never describe HOW it will be done.
• Use Cases are Analysis Products, not Design Products.

Example: Library System

• The Library System is used by the ABC College students and Faculty. The Library contains Books and Journals.
• Books can be issued to both the Students and Faculty. Journals can only be issued to the Faculty.
• Books and Journals can only be issued by the Librarian. The deputy-Librarian is in-charge of receiving the Returned Books and Journals.
• The Accountant is responsible for Receiving the fine for over-due books. Fine is charged only to students, and not to the Faculty.
Example....

• We need to automate these tasks

• To automate these tasks we need to identify the requirements using use cases

• Example of an issue book use case

How Do We Develop the Use Case Model?

• A Simple Use Case Recipe

  – Step 1. Identify the who is going to be using the system directly - e.g. hitting keys on the keyboard or automatically interfaces with the system. These are the Actors.

  – Step 2. Pick one of these Actors.

  – Step 3. Define what that Actor wants to do with the system. Each of these things that the actor wants to do with the system become a Use Case

Finding out Actors

• Step 5. Describe that basic course in the description for the use case.
  – Describe it as “Actor does something, system does something. Actor does something, system does something.” but keep it at a high level. Do not mention any GUI specifics for example. Also you only describe things that the system does that the actor would be aware of and conversely you only describe what the actor does that the system would be aware of.

• Step 6. Repeat the above steps for each actor

Use Case Diagram for the System

Use Cases Types & Formats

• Black Box
  – Treats the system as a black box
  – No information about the internal working of the system
  – Focus on what and not how
  – E.g.
    • The system records lending of the book (best)
    • The system saves the data of the book lender into a file or database (bad)
    • the system issues the SQL insert statement. (worst)

Use Case Formality Style

• Brief
  – One paragraph description, usually of a happy day scenario

• Casual
  – Multiple paragraphs describing success as well as alternate course of action

• Fully dressed
  – Two column or a single column format with everything explained
Stories

- Using a system to meet a goal; e.g. (brief format) --

**Process Sale**: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a Receipt from the system and then leaves with the items.

**Example (casual format)**

**Handle Returns**

**Main Success Scenario**: A customer arrives at a checkout with items to return. The cashier uses the POS system to record each returned item...

**Alternate Scenarios**:
- If they paid by credit, and the reimbursement transaction to their credit account is rejected, inform the customer and pay them with cash.
- If the item identifier is not found in the system, notify the cashier and suggest manual entry of the identifier code.
- If the system detects failure to communicate with the external accounting system... (etc.)

---

**Fully Dressed Details...**

- **Example of a fully dressed use case**
- **Stakeholders and Interests**
  - Important: defines what the use case covers ("all and only that which satisfies the stakeholders’ interests")
- **Preconditions**
  - What must always be true before beginning a scenario in the use case (e.g., "user is already logged in")

---

**Details...[2]**

- **Success Guarantees (Postconditions)**
  - What must be true on successful completion of the use case; should meet needs of all stakeholders
- **Basic Flow**
  - Records steps: interactions between actors; a validation (by system); a state change (by system)

---

**Details...[3]**

- **Extensions (Alternative Flows)**
  - Scenario branches (success/failure)
  - Longer/more complex than basic flow
  - Branches indicated by letter following basic flow step number, e.g. "3a"
  - Two parts: condition, handling
- **Special Requirements**
  - Non-functional considerations (e.g., performance expectations)

---

**Details...[4]**

- **Technology & Data Variations**
  - Non-functional constraints expressed by the stakeholders (e.g., "must support a card reader")
Goals & Scope of Use Case

• What is a proper, valid use case?
  Granularity level

• For Course registration application which one of the following is a proper use case
  – Verify password
  – log in
  – Add a course

EBP Guideline

For requirement analysis for a computer application focus on use case at the elementary business process (EBP)

Now what does this French means?

EBP Guideline

• A common use case mistake is defining use cases at too low level or sub function level
• An EBP level use case is known as a user goal level use case. i.e. user wants to do something that adds business value.
• It is not necessary that 100% of your use cases will follow the EBP guideline, but a predominant portion of your use cases will

Use Cases & Goals

• Actors has goals (record sale, add course, issue book) and they use computer application to achieve that.
  • EBP is therefore a user level goal
  • While figuring out use cases try to raise the level of a goal to a user level goal.

Raising the Goal level

• Analyst: what u want to do with the system
  • PCO: log in etc. (a sub function)
  • Analyst: why would you want to login ?
    • (raising the level of goal)
  • PCO: to add course, find student’s GPA
    • (That’s what user want to do)
  • Analyst: why would you want to do that ?
  • PCO: To run the students registration process smoothly
    • (More of an enterprise level goal and not a user level goal)

Reasonable EBP Violations

• Some times a sub function is repeated in several base use cases.
  • Log in , Pay by credit
• Though the user goal or business process is somewhat higher (login to add course, recording a sale and payment is by credit) yet the common sub function is often separated and shown as a separate use case
• If there are too many use cases. Sometimes the CRUD (create, retrieve , update & delete) cases are combined to form one use case (Manage User). Although they represent four different user goals
Essential Vs. Concrete Style

- Try to write use case in the essential user interface free style. i.e. do not commit to the user interface details. E.g. user will provide pswd Vs. user will provide pswd in the dialogue box.

- What we do practically?

Finding Actors, Goals and Use Cases

- Choose the system boundary
- Identify the primary actors
- For each primary actor identify the goals and raise it to a user goal level.
- Define use cases that satisfy user goal. this will be a use case
- Note (Sometimes a use case leads to identification of an actor)

Finding Actors…

- Who gives inputs get outputs?
- Who starts & stops the system?
- Who does user management and security?
- Who evaluates system logs, performance?
- Who does system administration?
- Is time an actor?

Actors & Goals via Event Analysis

<table>
<thead>
<tr>
<th>External Event</th>
<th>From Actor</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter sale line</td>
<td>Cashier</td>
<td>Process Sale</td>
</tr>
</tbody>
</table>

More Guidelines

- Make an actor goal list
- Name the use case similar to the user goal
- The name of the use case should start with a verb.
- Use cases are about writing text, so greater emphasize should be on writing text rather diagramming or relationships
- In RUP
  - Identify most of the use cases in inception
  - For 10/20 % high risk or high priority use cases, write fully dressed versions
  - For the remaining majority write brief versions

Actor-Goal List

<table>
<thead>
<tr>
<th>Actor</th>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cashier</td>
<td>Process sales</td>
</tr>
<tr>
<td></td>
<td>Process rentals</td>
</tr>
<tr>
<td></td>
<td>Handle returns</td>
</tr>
<tr>
<td>Manager</td>
<td>Start up</td>
</tr>
<tr>
<td></td>
<td>Shut down</td>
</tr>
<tr>
<td>System Admin</td>
<td>Add user</td>
</tr>
<tr>
<td></td>
<td>Modify user</td>
</tr>
</tbody>
</table>
Use Cases vs. Feature Lists

- Feature Lists: “the system shall…”
  - More traditional approach
  - Less preferable for user-oriented s/w
    - Only mention function, not flow
    - If flows are written elsewhere, then redundant = hard to maintain in parallel
- High-level feature lists are okay
  - Include in Vision document
  - “executive summary of use cases”
**UP Artifacts and Timing**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Modeling</td>
<td>Domain Model</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td>Use-Case Model</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Video</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supplementary Specification</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Glossary</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>Design Model</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SW Architecture Document</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Class Model</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>Implementation Model</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SW Development Plan</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td>Test Model</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment</td>
<td>Development Case</td>
<td>s</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Influence of Artifacts On Each Other**

Design can be thought of as "Use Case Realization".

**Benefits of Use Cases**

- Use cases are the primary vehicle for requirements capture in RUP.
- Use cases are described using the language of the customer (language of the domain which is defined in the glossary) therefore act as an easily-understood communication mechanism.
- Use cases provide a contractual delivery process (RUP is Use Case Driven).
- Use Cases provide bases for cost estimation and testing.
- Use cases provide When requirements are traced, they make it difficult for requirements to fall through the cracks.
- Use cases provide a concise summary of what the system should do at an abstract (low modification cost) level.

**Difficulties with Use Cases**

- As functional decompositions, it is often difficult to make the transition from functional description to object description to class design.
- Reuse at the class level can be hindered by each developer "taking a Use Case and running with it". Since UCs do not talk about classes, developers often wind up in a vacuum during object analysis, and can often wind up doing things their own way, making reuse difficult.
- Use Cases make stating non-functional requirements difficult (where do you say that X must execute at Y/sec?).
- Testing functionality is straightforward, but unit testing the particular implementations and non-functional requirements is not obvious.

**Assignments**

- Reading assignment
  - Chapter 1-6 with great emphasize on chap. 6
- Written assignment
  - Identify the use cases for the given case study.
Questions and Discussion

References

- Craig Larman
- SW Engineering for IT Course at CMU