CSE 403

Software Engineering

Requirements and Use cases

Project assignments are done!

Logistics: starting this week

Deliverables

- **Milestone** due every **Tuesday 11:59pm**
- **Progress report and agenda** due every **Wednesday 11:59pm**

Meetings

- **Team meeting** every **Tuesday** 1:30pm -- 2:20pm
- **Project meeting** every **Thursday** 1:30pm -- 2:20pm

Logistics: for procrastinators :)

Deliverables

- **Milestone** due every **Tuesday 10pm**
- **Progress report and agenda** due every **Wednesday 10pm**

Meetings

- **Team meeting** every **Tuesday** 1:30pm -- 2:20pm
- **Project meeting** every **Thursday** 1:30pm -- 2:20pm

Logistics: suggested process

- **Tuesday:** team meeting and submission
  - Final milestone checks (all tasks done before the team meeting)
  - Submit current milestone.
- **Wednesday:** preparation
  - Read the next assignment!
  - Post general assignment questions on Slack (technical or nontechnical).
  - Submit progress report and agenda with project-specific questions.
- **Thursday:** project meeting
  - Resolve project-specific questions.
  - Are tasks clear? Any issues or blockers?
- ...

Requirements

Recap: Life-cycle stages

Virtually all SDLC models have the following stages:

- **Requirements** $\longleftarrow$ **Our focus this week**
- Design
- Implementation
- Testing
- Maintenance

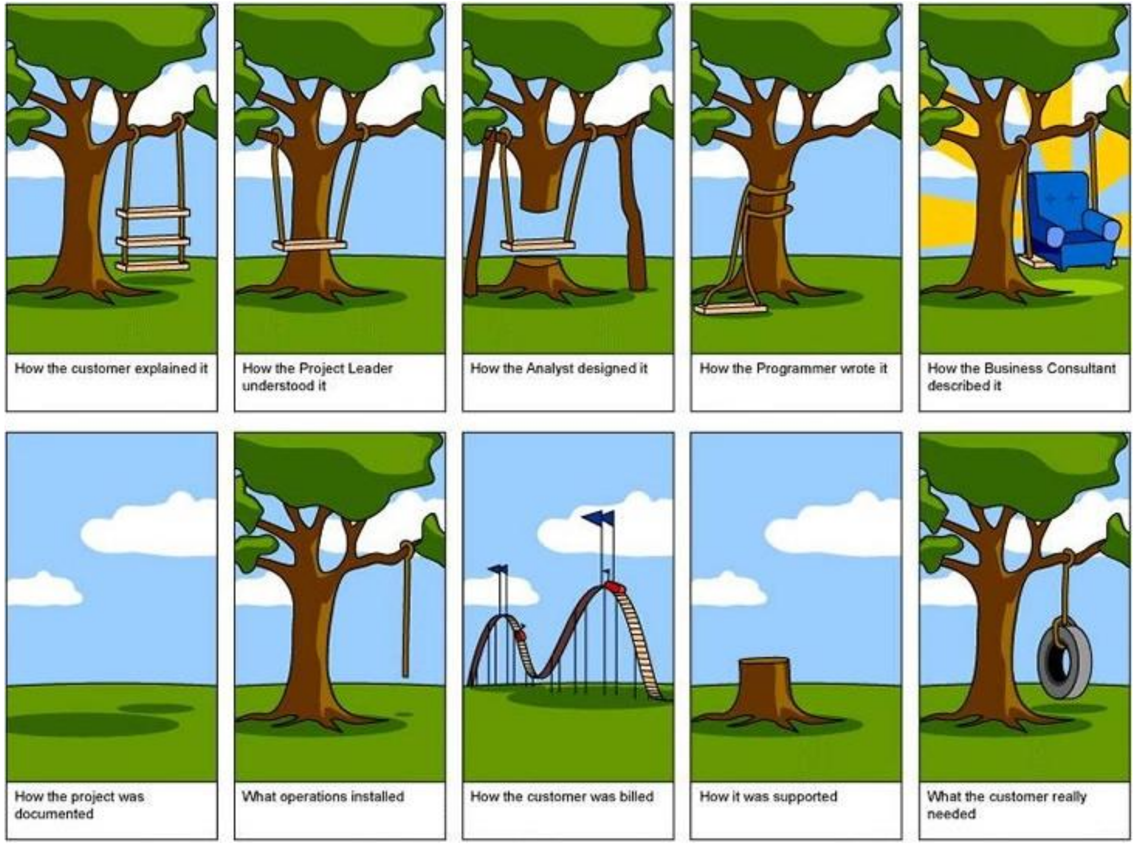
Traditional models:

- Waterfall, Prototyping, Spiral, etc.

Agile models:

- eXtreme Programming, Scrum, etc.

Requirements in one picture


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Software requirements

Requirements specify what to build

- describe **what, not how**
- describe the problem, not the solution
- reflect system design, not software design

“What” vs. “how” is relative

One person’s **what** is another person’s *how*:

- Input **file processing** is the **what**, *parsing* is the *how.*
- **Parsing** is the **what**, a *stack* is the *how*.
- **Stack** is the **what**, a *linked list* is the *how*.
- A **linked list** is the **what**, *Node** is the *how*.

Requirements: Goals and roles

Goals when eliciting requirements:

- **Understand** precisely what is required of the software.
- **Communicate** this understanding precisely to all involved parties.
- **Control** production to ensure that system meets specification.

Roles of requirements:

- **Customers**: what should be delivered (contractual base).
- **Managers**: scheduling and monitoring (progress indicator).
- **Designers**: a spec to design the system.
- **Coders**: a range of acceptable implementations.
- **QA / Testers**: a basis for testing, verification, and validation.

How to elicit requirements?

Do:

- Talk to the users -- to learn how they work.
- Ask questions throughout the process -- "dig" for requirements.
- Think about why users do something in your app, not just what.
- Allow (and expect) requirements to change later.

Don't:

- Be too specific or detailed.
- Describe complex business logic or rules of the system.
- Describe the exact user interface used to implement a feature.
- Try to think of everything ahead of time. (You will fail!)
- Add unnecessary features not wanted by the customers.

Requirements engineering

Eliciting, analyzing, documenting, and maintaining requirements.

One way to classify requirements

- Functional requirements — examples: input-output behavior
- Non-functional requirements — examples: security, privacy, scalability
- Additional constraints — examples: languages, frameworks, infrastructure

Cockburn's requirements template

1. Purpose and scope
2. Terms (glossary)
3. **Use cases (the central artifact of requirements)**
4. Technology used
5. Other
   a. Development process: participants, values (fast-good-cheap), visibility, competition, dependencies
   b. Business rules (constraints)
   c. Performance demands
   d. Security, documentation
   e. Usability
   f. Portability
   g. Unresolved (deferred)
6. Human factors (legal, political, organizational, training)

Strategies for eliciting requirements

Common strategies

- Interviews
- Observations
- Use cases
- Feature list
- Prototyping (e.g., UI)

Challenges and common mistakes

Challenges

- Unclear scope and unclear requirements.
- Changing/evolving requirements.
- Finding the right balance (depends on customer):
  - Comprehensible vs. detailed.
  - Graphics vs. tables and explicit and precise wording.
  - Short and timely vs. complete and late.

Common Mistakes

- Implementation details instead of requirements.
- Projection of own models/ideas.
- Feature creep/bloat.

Feature creep/bloat

Feature creep:

- Gradual accumulation of features over time.
- Often has a negative overall effect on a large software project.

Why does feature creep happen? Because features are fun!

- Developers like to code them.
- Sales teams like to brag about them.
- Users (think they) want them.

Why is it bad?

- Too many options, more bugs, more delays, less testing, ...

Can you think of any products that have had feature creep?

Use Cases

What is a use case?

A **written description** of a **user's interaction with** the software **system to accomplish** a **goal**.

- It is an **example behavior** of the system
- Written from an **actor's point of view**, not the system's
- **3-9 clearly written steps** lead to a "main success scenario"

Terminology

- **Actor:** someone (or another system) interacting with the system
- **Primary actor:** person who initiates the action
- **Goal:** desired outcome of the primary actor

Use cases capture **functional requirements** of a system!

Benefits of use cases

- Establish an understanding between the customer and the developers of the requirements (**success scenarios**)
- Alert developers of special cases (alternatives) and error cases (exceptions) to test (**extension scenarios**)
- Capture a level of functionality (**list of goals**)

What is an extension?

A possible **branch** in a use case, e.g., **triggered by an error**; useful for identifying what **edge cases** need to be **handled/tested**

Do

- Think about how every step of the use case could fail
- Give a plausible response to each extension from the system
- Response should either jump to another step of the case, or end it

Don’t

- List things outside the use case ("User's power goes out")
- Make unreasonable assumptions ("DB will never fail")
- List a remedy that your system can't actually implement

4 steps for creating a use case

1. **Identify actors and goals**

- Actors: What users and (sub)systems interact with our system?
- Goals: What does each actor need our system to do?

4 steps for creating a use case

1. **Identify actors and goals**
2. **Write the main success scenario**

- Main success scenario is the preferred "happy path”
  - Easiest to read and understand
  - Everything else is a complication on this
- Capture each actor's intent and responsibility, from trigger to goal
  - State what information passes between actors
  - Number each step (line)

4 steps for creating a use case

1. **Identify actors and goals**
2. **Write the main success scenario**
3. **List the failure extensions**

- Many steps can fail (e.g., denied credit card, out of stock)
  - Note each failure condition separately, after the main success scenario
- Describe failure-handling
  - recoverable: back to main scenario (low stock + reduce quantity)
  - non-recoverable: fails (out of stock)
  - each scenario goes from trigger to completion
- Label with step number (success scenario line) and letter
  - 5a <failure condition>; 5a.1 <fail with error message>
  - 5b <failure condition>; 5b.1 <action>; 5b.2 <continue at failure step 7>

4 steps for creating a use case

1. **Identify actors and goals**
2. **Write the main success scenario**
3. **List the failure extensions**
4. **List the variations**

- Steps can have alternative behaviors
    - Label alternatives with step number (success scenario line) and symbol
        - 5’ <Alternative 1 for step 5>
        - 5” <Alternative 2 for step 5>

Qualities of a good use case

- **Focuses on interaction**
  - Starts with a request from an actor to the system
  - Ends with the production of all the answers to the request
- **Focuses on essential behaviors, from actor’s point of view**
  - Does not describe internal system activities
  - Does not describe the GUI in detail
- **Concise, clear, and accessible to non-programmers**
  - Easy to read
  - Summary fits on a page
  - Main success scenario and extensions

Use cases vs. other requirements

Which of the following requirements should be directly represented as a use case?

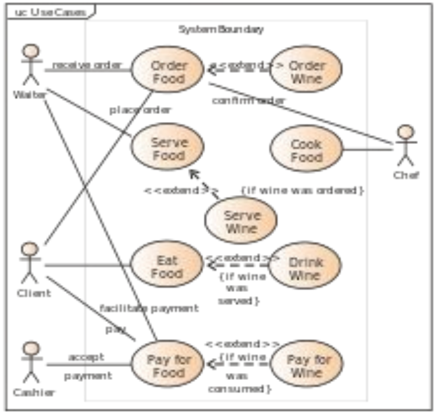
- Special deals may not run longer than 6 months.
- Customers only become preferred after 1 year.
- A customer has one and only one sales contact.
- Database response time is less than 2 seconds.
- Web site uptime requirement is 99.8%.
- Number of simultaneous users will be 200 max.

Styles of use cases

- Use case diagram (often in UML)
- Textual use case
  - Informal use case
  - Formal use case (≠ formal specification)

Use case diagram

“For reasons that remain a mystery to me, many people have focused on the stick figures and ellipses in use case writing since Jacobson's first book came out, and neglected to notice that use cases are fundamentally a text form.”
[Writing Effective Use Cases, Alistair Cockburn, 2000]



Informal use case: example

Patron loses a book

The **library patron** reports to the librarian that she has lost a book. The **librarian** prints out the library record and asks patron to speak with the head librarian, who will arrange for the patron to pay a fee. The **system** will be updated to reflect lost book, and patron's record is updated as well. The **head librarian** may authorize purchase of a replacement book.

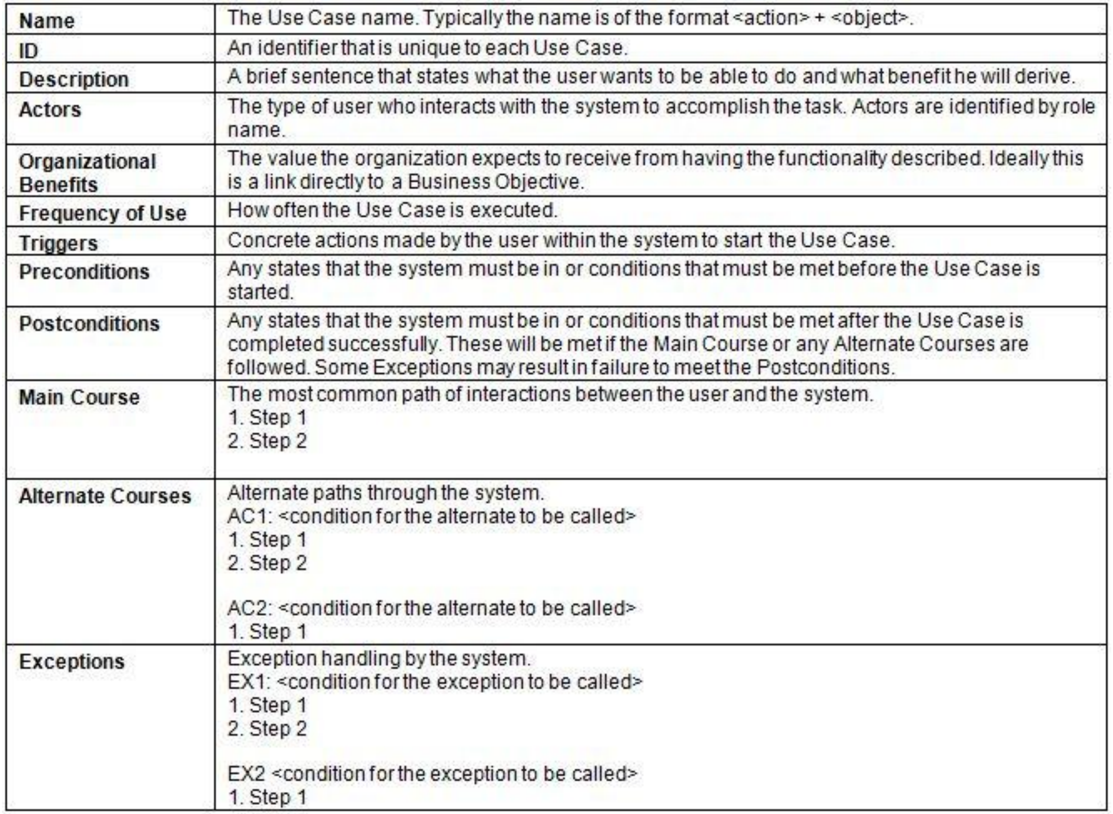
Informal use case: example

Patron loses a book

The **library patron** reports to the librarian that she has lost a book. The **librarian** prints out the library record and asks patron to speak with the head librarian, who will arrange for the patron to pay a fee. The **system** will be updated to reflect lost book, and patron's record is updated as well. The **head librarian** may authorize purchase of a replacement book.

It is almost always better to use structured text
or a template for formal use cases.

Formal use case



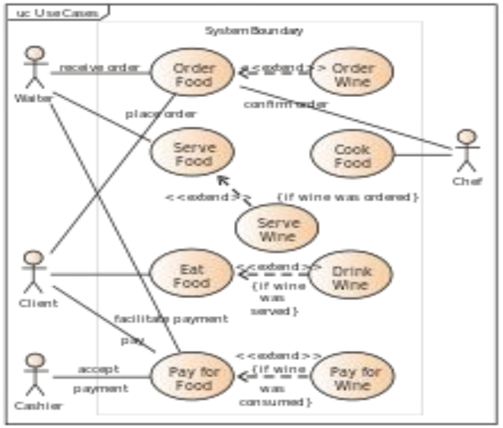
Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

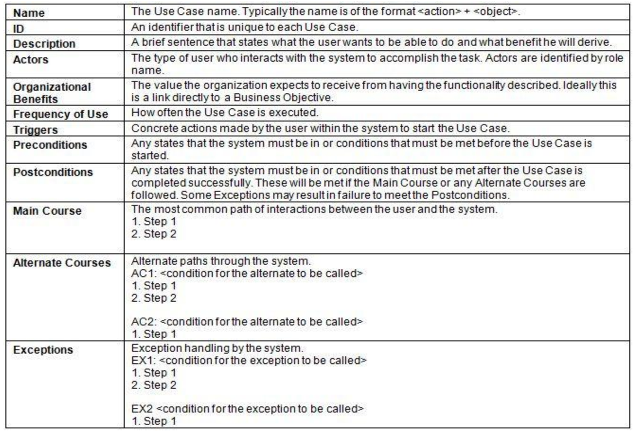
Formal use case: example

Goal	Patron wishes to reserve a book using the online catalog
Primary actor	Patron
Scope	Library system
Level	User
Precondition	Patron is at the login screen
Success end	Book is reserved
Failure end	Book is not reserved
Trigger	Patron logs into system

Main success scenario	1. Patron enters account and password 2. System verifies and logs patron in 3. System presents catalog with search screen 4. Patron enters book title 5. System finds match and presents location choices 6. Patron selects location and reserves book 7. System confirms reservation and re-presents catalog
Extensions (error scenarios)	2a. Password is incorrect 2a.1 System returns patron to login screen 2a.2 Patron backs out or tries again 5a. System cannot find book 5a.1 …
Variations (alternative scenarios)	4. Patron enters author or subject

Use case diagram vs. textual use case





Name	The Use Case name. Typically the name is of the format <action> + <object>.
ID	An identifier that is unique to each Use Case.
Description	A brief sentence that states what the user wants to be able to do and what benefit he will derive.
Actors	The type of user who interacts with the system to accomplish the task. Actors are identified by role name.
Organizational Benefits	The value the organization expects to receive from having the functionality described. Ideally this is a link directly to a Business Objective.
Frequency of Use	How often the Use Case is executed.
Triggers	Concrete actions made by the user within the system to start the Use Case.
Preconditions	Any states that the system must be in or conditions that must be met before the Use Case is started.
Postconditions	Any states that the system must be in or conditions that must be met after the Use Case is completed successfully. These will be met if the Main Course or any Alternate Courses are followed. Some Exceptions may result in failure to meet the Postconditions.
Main Course	The most common path of interactions between the user and the system. 1. Step 1 2. Step 2
Alternate Courses	Alternate paths through the system. AC1: <condition for the alternate to be called> 1. Step 1 2. Step 2 AC2: <condition for the alternate to be called> 1. Step 1
Exceptions	Exception handling by the system. EX1: <condition for the exception to be called> 1. Step 1 2. Step 2 EX2 <condition for the exception to be called> 1. Step 1

Which one would you choose and why?