Chapter 6 – Use Cases

Use case → text stories of some actor using a system to meet goals and record requirements. That's not a diagram, it must more details or structured.

Informally, a UC is a collection of related success and failure scenarios that describe an actor using a system to support a goal.

UCs can in turn influence many other analysis design, implementation, project management and test artifacts.

- Jacobson's definition- A set of UC instances (scenario), where each instance (scenario) is a sequence of actions a system performs that yields an observable result of value to a particular actor
- Encourages focus on achieving user goals
- A UC is a dialogue between an actor (aka 'user') and the system .
- The dialogue is presented as an ordered sequence of steps

Example of use cases→ 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.

Actor → something with behavior, such as a person (identified by role e.g. cashier), computer system, or organization that triggers the UC execution. Actors include system under discussion (SuD) itself when it calls upon the services with other systems.

Scenario → also use case instance→ is a specific sequence of actions (or steps) and interactions between actors and the system. Each UC is one particular story of using a system, or one path through the UC. e.g. a scenario of purchasing items with cash.

Use Cases and the Use-Case Model

Use cases are text documents, not diagrams, and use-case modeling is primarily an act of writing text, not drawing diagrams.

The use-case Model may optionally include a UML use case diagram to show the names of use cases and actors, and their relationship. This gives a nice **context diagram** of a system and its environment.

UC strengths & uses →Motivation: Why Use Cases?

[1] UCs emphasize user goals & objectives
[2] UCs decompose system functionality into a set of discrete tasks (divide & conquer)
[3] UCs are easy for users to understand
[4] UCs can be reused for user documentation
[5] UCs are basis for planning work during each iteration
[6] UCs guide developers during implementation
[7] Test cases can be taken directly from UCs
[8] UCs are independent of implementing technology

What are 3 kinds of Actors?

[1] **Primary actor →** has user goals fulfilled through using services of the SuD. For example, the cashier.

[2] **Supporting actor →** provides a service (for example, information) to the SuD. The automated payment authorization service is an example. Often a computer system, but could be an organization or person.

[3] **Offstage actor →** has an interest in the behavior of the use case, but is not primary or supporting; for example, a government tax agency.

What are 3 common use case formats?

Use cases can be written in different formats and levels of formality

[1] **Brief →** terse one paragraph summary, usually main success scenario.
[2] **Casual →** Informal paragraph format, usually multiple paragraphs that cover various scenarios.
[3] **Fully dressed →** all steps and variations are written in detail and there are supporting sections such as preconditions and postconditions.

How to Find Use Cases

[1] Choose the system boundary. Is it just a software application, the hardware and application as a unit, that plus a person using it, or an entire organization?
[2] Identify the primary actors.
[3] Identify the goals for each primary actor.

[4] Define use cases that satisfy user goals.

Applying UML: Use Cases Diagrams

The UML provide use case diagram notation to illustrate the name of use cases and actors, and the relationship between them

The UML include a diagram useful to visualize workflows and business processes: activity diagram.

Generic UC Structure

[1] **Name:** 'Verb Noun' format e.g. Open Account, Enroll Student
[2] **Scope:** The system under design
[3] **Level:** "user goal" or "subfunction"
[4] **Description:** 1 paragraph high-level description of UC
[5] Actors → List all Actors who can trigger UC
  a. **Primary Actor:** calls on sys to deliver its services
  b. **Stakeholders and interests:** who cares about the UC and what do they want.
[6] **Preconditions:** What must be true before UC can be triggered + Perhaps another UC must precede this one
[7] **Postconditions (Success Guarantee):** What must be true upon completion of the UC
[8] **Base Scenario (Main Success Scenario):** Steps describing the sequence of actions that must occur assuming no alternative flows or error conditions + 'Sunny day scenario'
[9] **Alternative Scenarios (Extensions):** State condition that precipitates branching + Specify sequence of steps that occur as a result
[10] **Additional Information (Special Requirements):** Any other information or constraints that impact UC + Key terms should be defined in a glossary, or possibly data dictionary

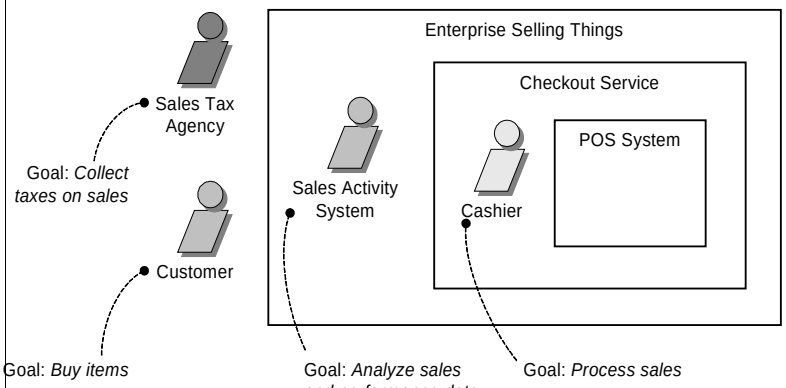
Use Cases-Base Scenario

- **Essential course guidelines for writing UCs**

[1] Minimize verbosity; keep it simple, stupid!
[2] Use terminology from business domain
[3] Avoid IT jargon / implementation concepts
[4] Never refer to mouse clicks, GUI widgets, screens, databases, etc.
[5] Each step begins with 'System' or 'User'
[6] Avoid using 'System' & 'User' in same step
[7] Avoid articles ('a', 'an', 'the')
[8] Present tense
[9] Active voice
[10] 1-2 actions/step w/ no if/else logic
[11] Adhere to triplet structure (discussed below)

- **Quality control guidelines**

[1] No violations of writing guidelines
[2] It is functionally correct & complete
  a. No missing scenarios
  b. For each scenario, all steps in proper sequence with no missing steps
[3] Users can verify with confidence



Tests to find useful UCs:

- The Boss Test: the boss ask questions about the UCs scenarios to ensure they fulfills the needs. Your Boss asks, "What have u been doing all the day?" You reply: "Logging in".. Then the UCs fails!
- The EBP Test: Elementary Business Process: a task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves the data in a consistent state, e.g. Approve Credit
- The Size Test: observe the size of main scenario, how many steps.. A common mistake is modeling some UC with only a single step such as "Enter Item ID"

Chapter 7 – Other Requirements

← related to Ch.5: How are reqs organized in UP Artifacts?

Functional:

[1] **Use-Case Model** → This artifact contains a set of typical scenarios of using a system. They are primarily for functional (behavioral) reqs.

Non-Functional:

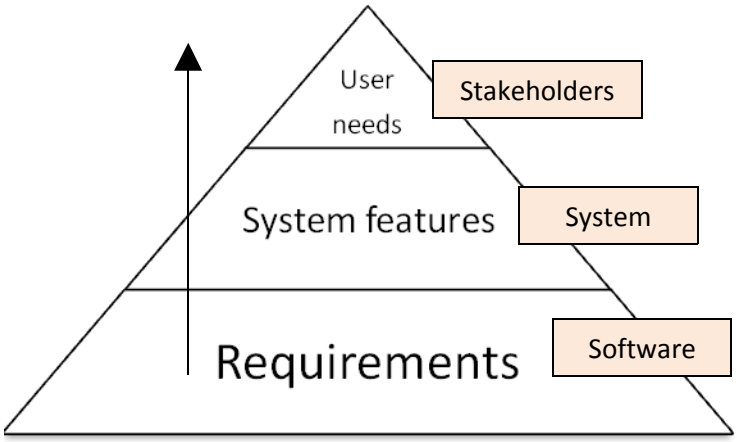
[1] **Supplementary specification** → This artifact captures and defines non-functional requirements or quality attributes +URPS (Usability, Reliability, Performance, Supportability and more) such as reports, documentation, packaging, supportability, licensing etc. It is also a place to record functional features not expressible as UCs such as report generation.
[2] **Glossary** → This artifact defines noteworthy terms and definitions. It can also play the role of **data dictionary**,

which records reqs related to data such as validation rules, acceptable values.etc

[3] **Vision** → This artifact summarizes high-level reqs that are elaborated in the UC-Model and SS, and summarizes the business case for the project. It gives executive summary for project's Big Ideas.

[4] **Business Rules** → **also Domain Rules** → captures long-living and spanning rules or policies that are required in the domain or business, such as tax laws, that transcend one particular application.

System Features → **in the UP (Vision)** → an externally observable services provided by the system which directly fulfill stakeholders needs.



Chapter 8 – Elaboration

Elaboration → the initial set of iterations during which:

[1] Core, risky architecture is implemented & tested
[2] Majority of reqs are discovered & stabilized
[3] Major risks mitigated or solved
[4] Sound cost estimate

Elaboration is NOT TO:

[1] create throwaway prototypes
[2] do 'quick & dirty' programming

Order of UC implementation based on the following

[1] **Risk** → Technical complexity, degree of uncertainty
[2] **Coverage** → Want UCs that provide 'wide and shallow' functionality for system + The base scenarios of core UCs
[3] **Criticality** → High business value UCs

Key artifacts in Elaboration →

Artifact	Comment
Domain model	Visualization of the important concepts in the business domain
Design model	Set of diagrams showing logical design. This include SW class diagram, sequence diagrams (object interactions diagrams), package diagrams
Data model	Includes DB schemas & relationships, and Mapping from objects to DB schema
UC storyboard/ UI Prototypes	Description of UI, storyboards (screen flow model), user navigation paths through system, usability models …etc
SW architecture model	Summary of: - Key architectural issues and their resolution in the design ➢ design ideas & their motivation

Key elaboration errors →

[1] Not tackling high risk/high value UCs
[2] Executable architecture not delivered
[3] Minimal feedback or user participation
[4] System not tested as its coded

Chapter 9 – Domain Model

Domain Model → **also conceptual model, domain object model and analysis object model** → a visual representation of conceptual classes or real-situation objects in a domain, not sw objects.

UP Domain Model is specialization of the UP **Business Object Model (BOM)** "focusing on explaining 'things' and products important to a business domain".

Applying UML notation, a domain model is illustrated with a set of class diagrams in which no operations (method signatures) are defined. It provides a conceptual perspective. It shows:

- Domain objects or conceptual classes: idea, thing or object related to the business domain
- Associations between conceptual classes: a relationship that indicates some meaningful & interesting connection. It has:
  - Name => *ClassName-VerbPhrase-ClassName*
  - Each end of a association is called a "role"; it may have: multiplicity – name – navigability.
- Attributes of conceptual classes

Association → a description of a related set of links between objects of two conceptual classes.

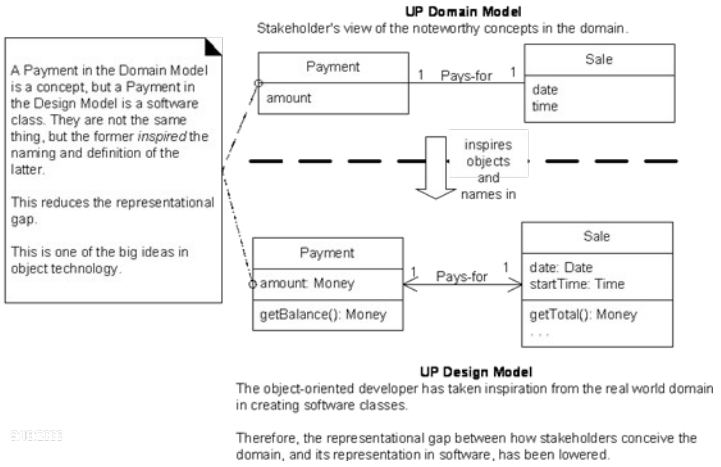
Attribute → a named characteristic or property of class.
Conceptual class → real-situation 'things' or objects in a domain, not sw objects.
Domain → a formal boundary that defines a particular subject or area of interest.

Multiplicity → the number of objects permitted to participate in an association.
Role→ a named end of an association to indicates its purpose.
Object → an instance of a class that encapsulates state and behaviour. More informally, an example of a thing.

Why we call DM a "Visual Dictionary"?

Because it visualizes and relates words or concepts in the domain.

Lower Representational Gap with OO modeling!

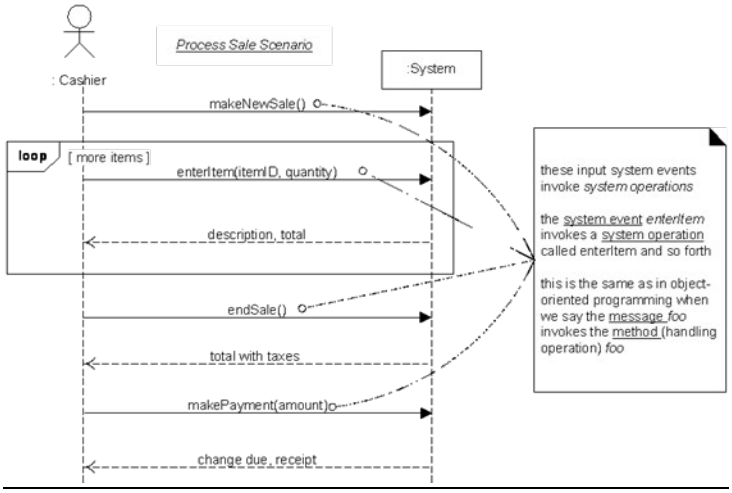


Chapter 10 – SSD

UCS describe how external actors interact with the system, during this interaction, an actor generates **system events** to a system, usually requesting some **system operation** to handle the event.

System Sequence Diagram (SSD) → a picture that shows, for one particular scenario of UC, the events that external actors generate, their order, and inter-system events. System is treated as a *black box*, to emphasize events that cross the system boundary from actor to system.

System events → events generated during the interaction between actors and the system. SSDs show system events or I/O messages relative to the system
System operations → handle system events. System operations are operations that the system as a black box component offers in its public interface.
Operation → (in UML)a specification of a transformation or query that an object may be called to execute. An operation has a signature, specified by its name and parameters, and it is invoked via a message. A method is specification of an operation with a specific algorithm.



Motivation: Why SSD?

- Useful to know external **system events**
- Useful to investigate and analyze **system behaviour** as a *black box* to know what a system does without explaining how it does it!

Applying UML in SSD → can use UML Sequence Diagram notation.
Interaction frames → used to show loops in SSD.

Chapter 11 – OC

Operation Contracts → more detailed or precise description of system behaviour. OCs use pre- and post- conditions to describe detailed changes to objects in the domain model, as a result of the system operation.

Sections of a contract →

Operation	Name of operation, and parameters
Cross Ref	UCs that this Op occurs within
Preconditions	Noteworthy assumption about the state of the system or objects in the DM before executing the operation
Postconditions	The most important section. The state of objects in the DM after the completion of the operation. Postconditions fall into three categories: [1] Instance creation & deletion [2] Attribute change of value [3] Associations formed & broken. Postconditions should be written in declarative, passive past tense form to emphasize the observation of a change.

Preconditions→ a constraint that must hold true before an operation is requested.
Postconditions→ a constraint that must hold true after the completion of an operation.