

Requirements Analysis with Use Cases Theory (9 Lessons)

Shane Sendall and Alfred Strohmeier
Swiss Federal Institute of Technology in Lausanne
Software Engineering Lab

Course Objectives

- ▲ Upon completion of this course, participants should be able to:
 - ◆ write use cases that capture functional requirements of a system under development;
 - ◆ understand the role of use cases in requirements analysis;
 - ◆ understand the importance of capturing the functional requirements without going into design/implementation detail;
 - ◆ understand the relationship between use cases and non-functional requirements;
 - ◆ understand the relationship of use cases to business process modeling;
 - ◆ understand what makes an effective use case;
 - ◆ understand the limitations of use cases and be aware of other models available that can make use cases more precise and rigorous.

Course Overview

▲ Theory

- ◆ Requirements Engineering and Use Cases
- ◆ Motivation for Use Cases
- ◆ Use Case Basics
- ◆ Use Cases Tips and Tricks
- ◆ Use Cases in UML
- ◆ Advanced Issues in Writing Use Cases
- ◆ Relating Use Cases with Business Process Modeling
- ◆ Relating Use Cases with Non-Functional Requirements
- ◆ User Interface Description with Conversations

▲ Exercises

▲ Case Studies

Bibliography

▲ Books

- F. Armour, and G Miller; *Advanced Use Case Modeling*. Object Technology Series, Addison-Wesley 2001.**
- D. Bellin and S. Suchman Simone; *The CRC Card Book*. Addison-Wesley, 1997.
- G. Booch, J. Rumbaugh, I. Jacobson; *The Unified Modeling Language User Manual*. Addison-Wesley 1999.
- J. Carroll; *Scenario-based Design: Envisioning Work and Technology in System Development*. Wiley, 1995.
- A. Cockburn; *Writing Effective Use Cases*. Addison-Wesley 2000.**
- L. Constantine and L. Lockwood; *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. ACM Press, Addison-Wesley 1999.
- M. Fowler; *UML Distilled: Applying the Standard Object Modeling Language*. Second Edition, Addison-Wesley, 1999.
- D. Gause and G. Weinberg; *Exploring Requirements: Quality Before Design*. Dorset House 1989.
- M. Hammer and J. Champy; *Reengineering the Corporation*. Harperbusiness, 2001.
- IBM Object-Oriented Technology Center; *Developing Object-Oriented Software: An Experience-Based Approach*. Prentice Hall 1996.
- I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard; *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley 1992.
- B. Kovitz; *Practical Software Requirements: A Manual of Content and Style*. Manning 1999.
- D. Kulak and E. Guiney; *Use Cases: Requirements in Context*. ACM Press, Addison-Wesley, 2000.**
- D. Leffingwell, and D. Widrig; *Managing Software Requirements: A Unified Approach*. Object Technology Series, Addison-Wesley 2000.**
- S. Robertson, and J. Robertson; *Mastering the Requirements Process*. Addison-Wesley 2000.**
- D. Rosenberg and K. Scott; *Use Case Driven Object Modeling with UML: A Practical Approach*. Addison-Wesley, 1999.
- R. Ross; *The Business Rule Book: Classifying, Defining and Modeling Rules*. V 4.0, Business Rules Solutions Inc. 1997.
- G. Schneider and J. Winters; *Applying Use Cases: A Practical Guide*. Addison-Wesley, 1998.
- P. Texel and C. Williams; *Use Cases Combined with Booch/OMT/UML*. Prentice Hall, 1997.

Bibliography

▲ Electronic Resources

Alistair Cockburn home page

<http://members.aol.com/acockburn/index.html>

Cetus -- Architecture and Design: Unified Modeling Language (UML)

http://www.cetus-links.org/oo_uml.html

McBreen Consulting -- Use Case Articles

<http://www.mcbreen.ab.ca/papers/UseCases.html>

OMG Unified Modeling Language Revision Task Force; *OMG Unified Modeling Language Specification*; Version 1.4, June 2001.

<http://www.celigent.com/omg/umlrtf/artifacts.htm>

Resources on Usage-Centered Design (Constantine Lockwood Ltd.)

<http://www.foruse.com/Resources.htm>

Software Development Magazine Online

<http://www.sdmagazine.com/articles/>

UseCases.org

<http://www.usecases.org/>

Use Case Zone

<http://www.pols.co.uk/usecasezone/>

Wirfs-Brock Associates: Resources

<http://www.wirfs-brock.com/pages/resources.html>

Bibliography

▲ Articles

- A. Cockburn; *Structuring Use Cases with Goals*. Journal of Object-Oriented Programming (JOOP Magazine), Sept-Oct and Nov-Dec, 1997.
- E. Ecklund, L. Delcambre and M. Freiling; *Change cases: use cases that identify future requirements*. OOPSLA '96 - Proceedings of the eleventh annual conference on Object-oriented programming systems, languages, and applications, 1996. pp. 342 - 358.
- M. Fowler; *Use and Abuse Cases*. Distributed Computing Magazine, 1999. Available at <http://www.martinfowler.com/articles.html>
- M. Glinz; *Problems and Deficiencies of UML as a Requirements Specification Language*. Proceedings of the Tenth International Workshop on Software Specification and Design, San Diego, 2000, pp. 11-22.
- T. Korson; *The Misuse of Use Cases*. Object Magazine, May 1998.
- S. Lilly; *Use case pitfalls: top 10 problems from real projects using use cases*. TOOLS 30, Proceedings of Technology of Object-Oriented Languages and Systems, 1999, pp. 174-183
- R. Malan and D. Bredemeyer; *Functional Requirements and Use Cases*. June 1999. Available at <http://www.bredemeyer.com/papers.htm>
- J. Mylopoulos, L. Chung and B. Nixon; *Representing and Using Nonfunctional Requirements: A Process-Oriented Approach*. IEEE Transactions on Software Engineering, Vol. 23, No. 3/4, 1998, pp. 127-155.
- A. Pols; *Use Case Rules of Thumb: Guidelines and lessons learned*. Fusion Newsletter, Feb. 1997.
- S. Sendall and A. Strohmeier; *From Use Cases to System Operation Specifications*. UML 2000 - The Unified Modeling Language: Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, S. Kent, A. Evans and B.Selic (Ed.), LNCS (Lecture Notes in Computer Science), no. 1939, 2000, pp. 1-15.
- R. Thayer and M. Dorfman (eds.); *Software Requirements Engineering*. 2nd Edition, IEEE Comp. Soc. Press, 1997.
- R. Wirfs-Brock; *The Art of Designing Meaningful Conversations*. Smalltalk Report, February, 1994.

Requirements Analysis with Use Cases

Theory (9 lessons)

Topics To Cover

- ▲ L1: Requirements Engineering and Use Cases
- ▲ L2: Use Case Basics I
- ▲ L3: Use Case Basics II
- ▲ L4: Use Case Tips & Tricks
- ▲ L5: Use Cases in UML
- ▲ L6: Advanced Issues in Writing Use Cases I
- ▲ L7: Advanced Issues in Writing Use Cases II
- ▲ L8 Relating Use Cases with Business Process Modeling & Non-Functional Requirements
- ▲ L9: User Interface Descriptions with Conversations

Requirements Engineering
and Use Cases

Use Case Basics

Lesson 1

Requirements Engineering and
Use Cases

Requirements Engineering— Definition

SWEED

▲ What is Requirements Engineering?

“...RE is concerned with identifying the purpose of a software system, and the contexts in which it will be used.

Hence, RE acts as the bridge between the real world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies.”

Definition

- ▲ What is a Requirement?
 - ◆ “A condition or capability needed by a user to solve a problem or achieve an objective.”
 - ◆ “A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.”
- ▲ “The set of all requirements forms the basis for subsequent development of the system or system component.”

Kinds of Requirements

▲ Functional requirements

- ◆ capture the intended behavior in terms of services, tasks or functions the system is required to perform. [Malan et al. 1999]
- ◆ Problem: if too general, ambiguity reigns; if too specific, design is stifled and leads to a large document;
- ◆ Techniques for writing them
 - Use cases,
 - Requirements List—“Shall” statements,
 - ...

Kinds of Requirements

- ▲ Non-functional requirements (or system qualities)
 - ◆ capture required properties or qualities of the system
 - ◆ often means: how well some behavioral or structural aspect of the system should be accomplished [Malan et al. 1999]
 - ◆ two categories:
 - Observable at runtime, e.g., performance, security, reliability, availability, usability, etc.
 - Not observable at runtime, e.g., extensibility, portability, reusability, etc.

Other Things that Need to be Taken into Account

SWEED

▲ Project constraints

- ◆ define how the eventual system must fit into the world and what rules must be followed in its development.
 - Organizational constraints, e.g., deadlines, budget, process standards, business rules;
 - Operational constraints, e.g., mandated technologies, interfaces to hardware and other software;
 - Legislative and Ethical constraints, e.g., safety, privacy, health regulations/standards.

Other Things that Need to be Taken into Account

SWEED

▲ Project drivers

- ◆ are the driving forces for the system:
 - Purpose of the System;
 - Client, Customer and other (non-user) Stakeholders;
 - Users of the System.

Other Things that Need to be Taken into Account

SWEED

▲ Project issues

- ◆ define the ideas, concerns, and issues related to the project:
 - Open issues
 - Installation and transition issues
 - Risks
 - Estimated cost
 - Change Cases
 - Ideas for solutions and off-the-shelf options

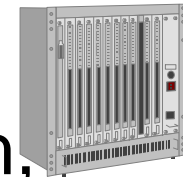
Where Do All these Things Come from?

They originate from various sources:

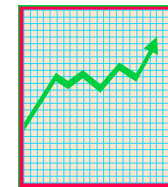
- ▲ People: Identify all the people who have a stake in the system. Remember that this includes non-users.



- ▲ Systems: What systems must the software interface with?



- ▲ Documents: This includes market research, standards, domain analysis, business process models, etc.



Requirements Engineering

- ▲ Feasibility Studies
- ▲ Requirements (Elicitation and) Analysis
- ▲ Requirements Specification
- ▲ Requirements Validation

Feasibility Studies

- ▲ Collect information pertinent to the following questions:
 - ◆ Does the system contribute to the overall objectives of the organization?
 - ◆ Is it possible to achieve the objectives of the project?
 - Is there a risk with high probability or big impact that makes the project too risky?
 - Can the system be implemented using current technology and within given cost and schedule constraints?
 - ◆ (follow ...)

Feasibility Studies

- ▲ Collect information pertinent to the following questions (continued):
 - ◆ Can the system be integrated with other systems which are already in place?
 - ◆ Can you reach agreement on the context of the work?
- ▲ The report should make a recommendation about whether or not the development should continue.

Requirements Elicitation and Analysis

SWEED

- ▲ Requirements Elicitation:
 - ◆ the activity of learning about the problem domain and extracting the requirements (and the other ones mentioned)
- ▲ Elicitation feeds analysis
- ▲ Tasks:
 - ◆ getting an understanding of the application domain
 - ◆ finding the right people to talk to (identifying the stakeholders)
 - ◆ asking them pertinent questions
 - ◆ (cont'd)

Requirements Elicitation and Analysis

SWEED

- ▲ Tasks (cont'd):
 - ◆ identifying and resolving contradictions between statements made by different people and terminology differences,
 - ◆ identifying and agreeing upon system boundaries,
 - ◆ keeping the project from going beyond its scope,
 - ◆ collecting information from other sources, and
 - ◆ making the gathered information more precise.

Requirements Elicitation and Analysis

SWEED

- ▲ Requirements elicitation is about communication between people.
- ▲ Use Cases are a popular choice as a tool.
- ▲ Techniques (not necessarily exclusive):
 - ◆ interviewing/questionnaires/surveys
 - ◆ analysis of existing documentation
 - ◆ workshops
 - ◆ storyboarding
 - ◆ role playing
 - ◆ prototyping



Requirements Specification

- ▲ The specification activity formalizes the results from the elicitation and analysis activity in a document
 - ◆ refines and elaborates the results of the analysis phase
- ▲ Two forces at different levels:
 - ◆ Stakeholders must agree to document (contract + statement of needs)
 - ◆ Developers must be able to use the document to design/implement the system (contract + documentation of system)
- ▲ Degree of formality depends on the approach

Example Table of Contents for *SWEED* Requirements Document

1. The Purpose of the Document
2. The Purpose of the System
3. Stakeholders of the System
4. Naming Conventions, Definitions, and Assumptions
5. Project Constraints
6. The Scope of the Work and System
7. Functional Requirements
8. Non-functional Requirements
9. Project Issues



Use Cases are found here

Requirements Validation

- ▲ Shows that the requirements actually define the system which the customer wants.
- ▲ Work is performed on complete draft of requirements specification.
- ▲ Different types of checks should be carried out:
 - ◆ validity checks
 - ◆ consistency checks
 - ◆ completeness checks
 - ◆ correctness checks
 - ◆ realism and necessity checks
 - ◆ verifiable checks

Requirements Validation

- ▲ Techniques for validation:
 - ◆ Reviews (inspections, walkthroughs, etc.)
 - ◆ Prototyping/simulating/testing from specification
 - ◆ Automated consistency analysis (tools)

So where do use cases
fit into all that?

SWEED

Use Cases

- ▲ A complete set of use cases specifies all the different ways to use the system, and thus defines all behavior required of the system, bounding the scope of the system. [Malan et al.'99]
- ▲ User goals summarize system functions (functional requirements) in verifiable terms of use that users, executives, and developers can appreciate and validate against their interests. [Cockburn '97]

Why do we use
use cases anyway?

SWEED

Use Cases

- ▲ Use cases offer a “familiar” representation to stakeholders
 - ◆ informal, easy to use, and story-telling-like style encourages them to be actively involved in defining the requirements;
 - ◆ thus, easier to validate with stakeholders;
 - ◆ allows common understanding between developers, system end users, and domain experts—“Is this what you want?”.
- ▲ They are scalable:
 - ◆ Use cases can be decomposed/composed—each step is *ideally* a sub-goal.

Use Cases

- ▲ Being a black-box view of the system, use cases are a good approach for finding the *What* rather than the *How*.
 - ◆ A black-box matches users view of the system: things going in and things coming out.
- ▲ Use cases force one to look at exceptional as well as normal behavior.
 - ◆ helps us to surface hidden requirements
- ▲ Use cases can help formulate system tests.
 - ◆ “Is this use case built into the system?”

Use Cases

- ▲ Replace the monotonous requirements list
 - ◆ use cases define all functional requirements
 - ◆ easier (and more intrinsically interesting) to extract user goals than list a bunch of “shall” statements
- ▲ Use case templates facilitate interviewing and reviews
- ▲ Ease an iterative development lifecycle
 - ◆ levels of precision for a use case by refinement
- ▲ Support an incremental development lifecycle
 - ◆ E.g. “Acme” Release 1: use cases 1-20;
“Acme” Release 2: use cases 1-29.

Use Case Example

Use Case: Deposit Money  

Scope: Bank Accounts and Transactions System

Level: User Goal

Intention in Context: The intention of the Client is to deposit money on an account. Clients do not interact with the System directly; instead, for this use case, a Client interacts via a Teller. Many Clients may be performing transactions and queries at any one time.

Primary Actor: Client

Main Success Scenario:

1. Client requests Teller to deposit money on an account, providing sum of money.
2. Teller requests System to perform a deposit, providing deposit transaction details*.
3. System validates the deposit, credits account for the amount, records details of the transaction, and informs Teller.

Use Case Example

Extensions:

2a. Client requests Teller to cancel deposit: use case ends in failure.

3a. System ascertains that it was given incorrect information:

3a.1. System informs Teller; use case continues at step 2.

3b. System ascertains that it was given insufficient information to perform deposit:

3b.1. System informs Teller; use case continues at step 2.

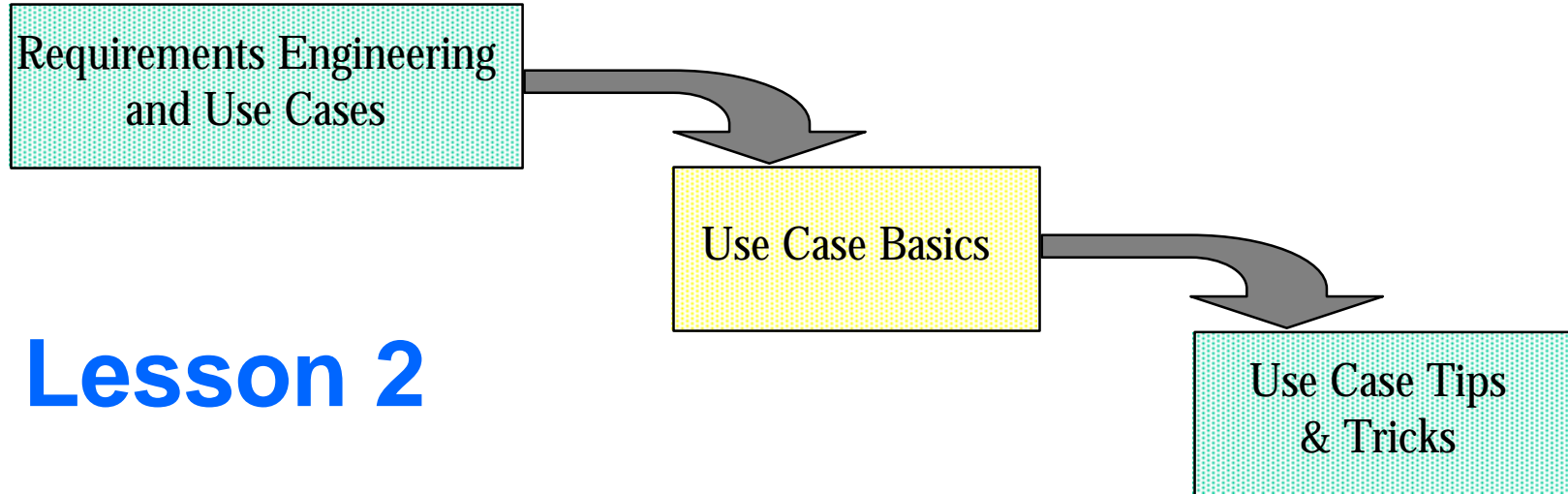
3c. System is not capable of depositing (e.g. transaction monitor of System is down)**:

3c.1. System informs Teller; use case ends in failure.

Notes:

* a hyperlink to a document that contains data details and formats.

** this is an example of an IT infrastructure failure, we only write it in a use case if there is a corresponding project constraint that states a physical separation, e.g., transaction section depends on a legacy system which is located somewhere else.



Lesson 2

Use Case Basics I

Use Case Basics

- ▲ Actors
- ▲ System Boundary
- ▲ Stakeholders and their Interests
- ▲ Scenarios
- ▲ Use Cases

What are they?

SWEED

Actors

- ▲ “The actors represent what interacts with the system.” [Jacobson ‘92]
- ▲ An actor represents a **role** that an external entity such as a user, a hardware device, or another system plays in interacting with the system.
- ▲ A role is defined by a set of characteristic needs, interests, expectations, behaviors and responsibilities. [Wirfs-Brock ‘94]

Actors

- ▲ An actor communicates by sending and receiving messages to/from the system under development.
- ▲ A use case is not limited to a single actor.
- ▲ Sources:
 - ◆ Documentation: user manuals and training guides are often directed at roles representing potential actors
 - ◆ People: Workshops, Meetings, etc.

What to Look for...

- ▲ Look for external entities that interact with the system
 - ◆ Which persons interact with the system (directly or indirectly)? Don't forget maintenance staff!
 - ◆ Will the system need to interact with other systems or existing legacy systems?
 - ◆ Are there any other hardware or software devices that interact with the system?
 - ◆ Are there any reporting interfaces or system administrative interfaces?

Actors Categories

- ▲ Jacobson (1992) categorized actors into two types:
- ▲ Primary Actor:
 - ◆ actor with goal on system
 - ◆ obtains value from the system
- ▲ Secondary Actor:
 - ◆ actor with which the system has a goal
 - ◆ supports “creating value” for other actors

Ask the following questions!

SWEED

After Identifying Actors...

▲ Primary and Secondary

- ◆ What part in the functioning of the business or organization does the actor's interaction with the system have?

▲ Primary

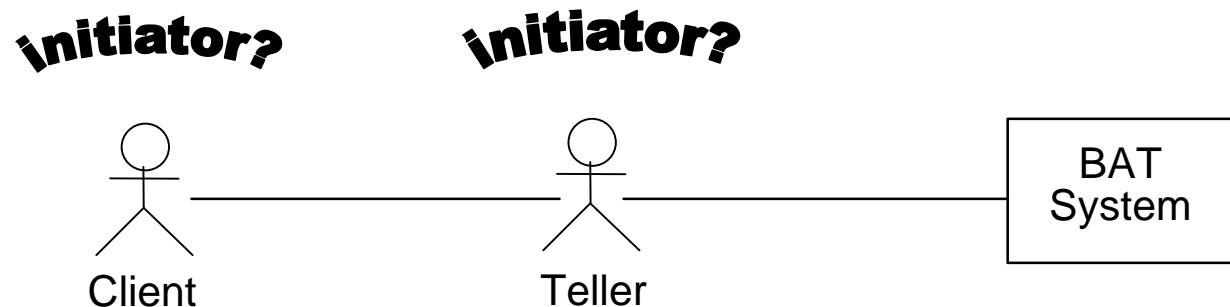
- ◆ What is the measurable value provided to the actor?
- ◆ What behavior must the system provide to satisfy this value?

▲ Secondary

- ◆ What value is the actor supporting in the use case?

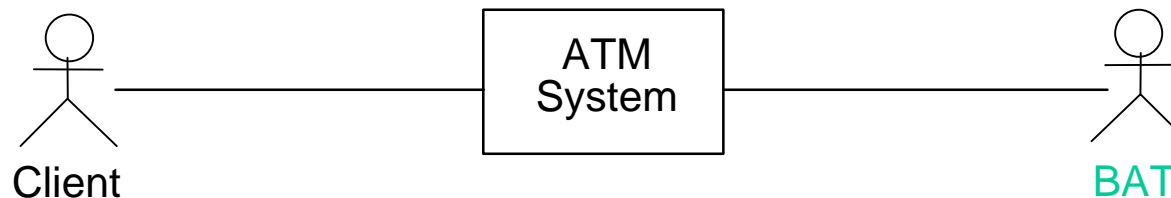
Actor Personalities

- ▲ An actor can have multiple personalities within a use case and across use cases
 - ◆ Initiator: initiates the use case
 - e.g. Client in “Deposit Money” use case?

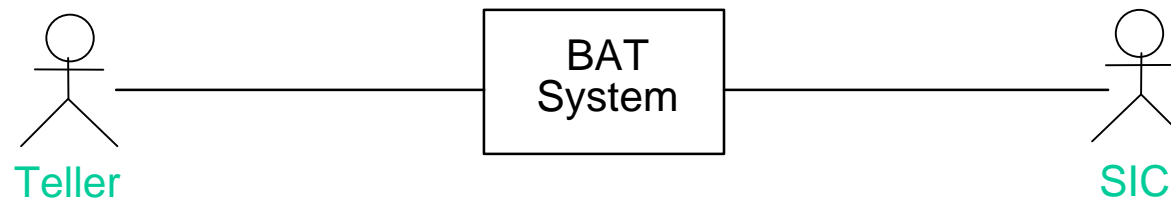


Actor Personalities

- ◆ **Server:** provides a service to the system in a use case.
 - e.g. “BAT System” in “Identify Client” use case for ATM system

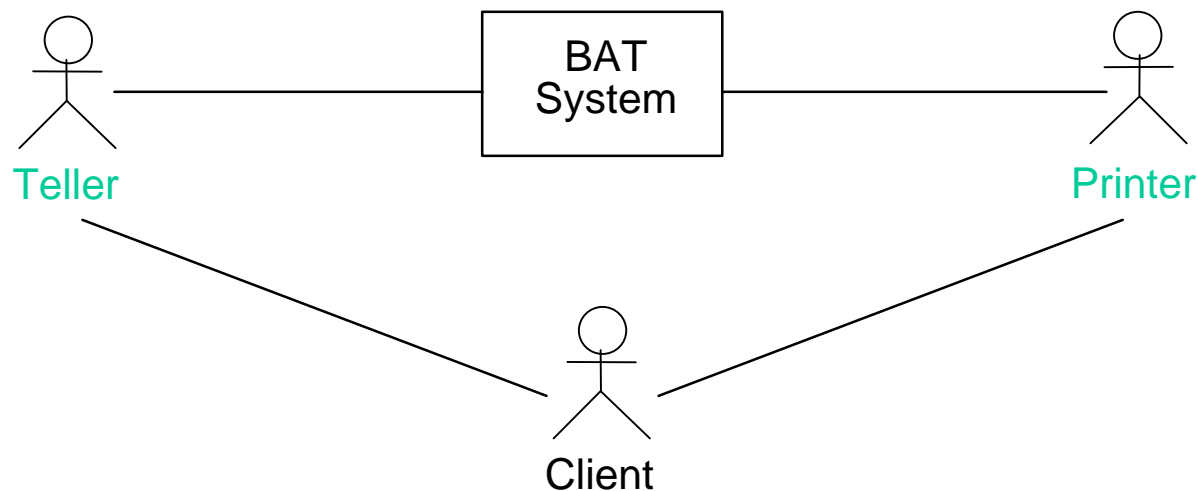


- ◆ **Receiver:** receives a notification from the system in a use case.
 - e.g. Teller and “Statistical Information Collector” (SIC) in the “Deposit Money” use case



Actor Personalities

- ◆ Facilitator: supports another actor's interaction with the system (or the inverse) in a use case.
 - e.g. Teller in the “Deposit Money” use case and inversely “Printer” in “Send Out Monthly Statements”



When Identifying Actor Personalities...



SWEED

▲ Initiator

- ◆ What is the initiation protocol with the system?

▲ Server

- ◆ What service does the actor provide? How is it related to the value the use case provides to another actor?

▲ Receiver

- ◆ What information does this actor receive? Why does this actor need the information?

When Identifying Actor Personalities...

▲ Facilitator

- ◆ What services does the facilitator perform?
- ◆ What restrictions does the facilitator place on the primary actor's interactions with the system?
- ◆ Does a special interface have to be built to accommodate this actor?

▲ Initiator, Server, Receiver, Facilitator

- ◆ What are the interface requirements for the actor-system interaction (including data format)?
- ◆ If the actor is a system, is its behavior sufficient? If not, does the system or actor need to be modified?



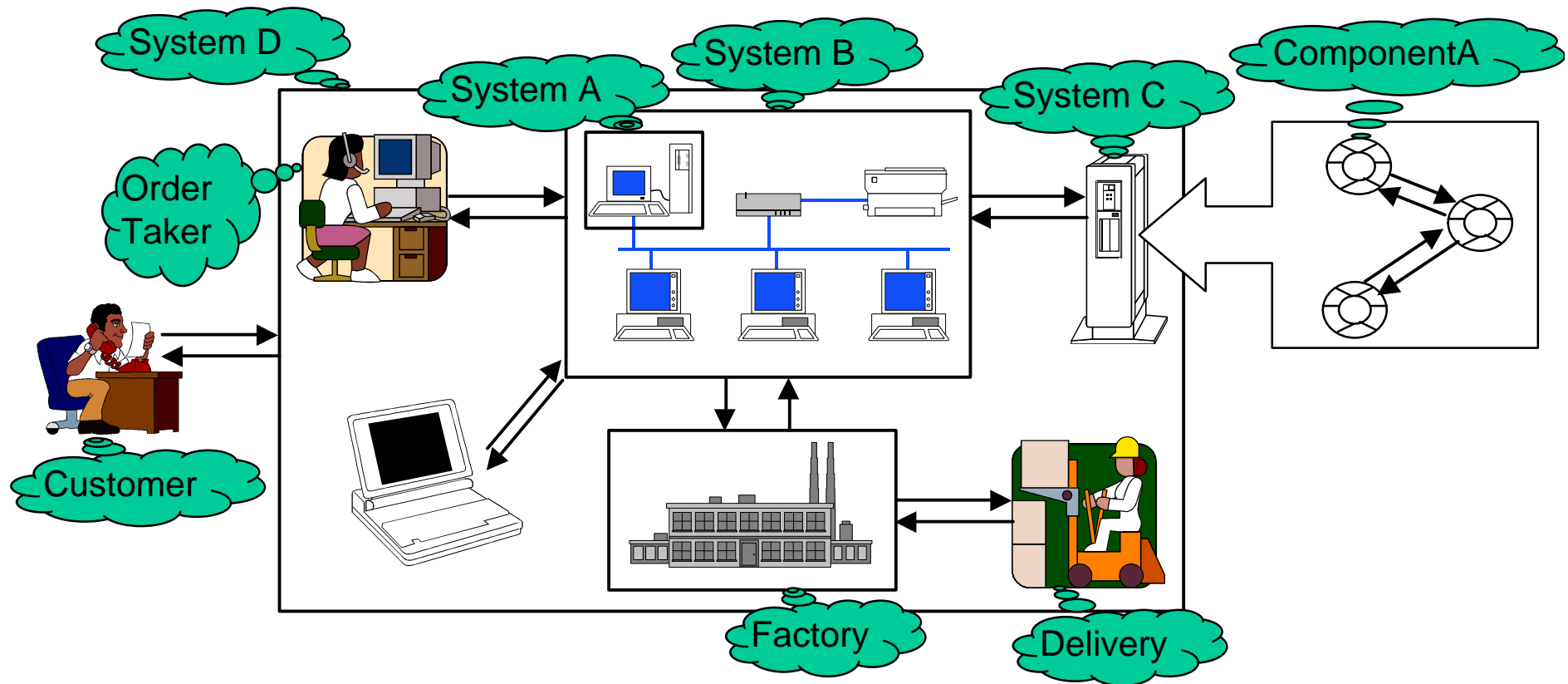
What is it?

SWEED

System Boundary

- ▲ The system boundary defines the separation between the system and its environment.
- ▲ Important to clearly define the system boundary.
 - ◆ Movement of the system boundary often has a large effect on what should be built.
 - ◆ A common area of conflict between stakeholders arises when they refer to different systems.
- ▲ Example: see next page.

System Boundary



Stakeholders and their Concerns

- ▲ A stakeholder is an individual, group or organization that has a vested interest in the system under development.
- ▲ The system enforces a contractual agreement between stakeholders, one of whom is the primary actor.
 - ◆ The use case describes how the system protects all of the stakeholders' interests under different circumstances, with the primary actor driving the scenario.

What are they?

How do they relate
to use cases?

SWEED

Scenarios

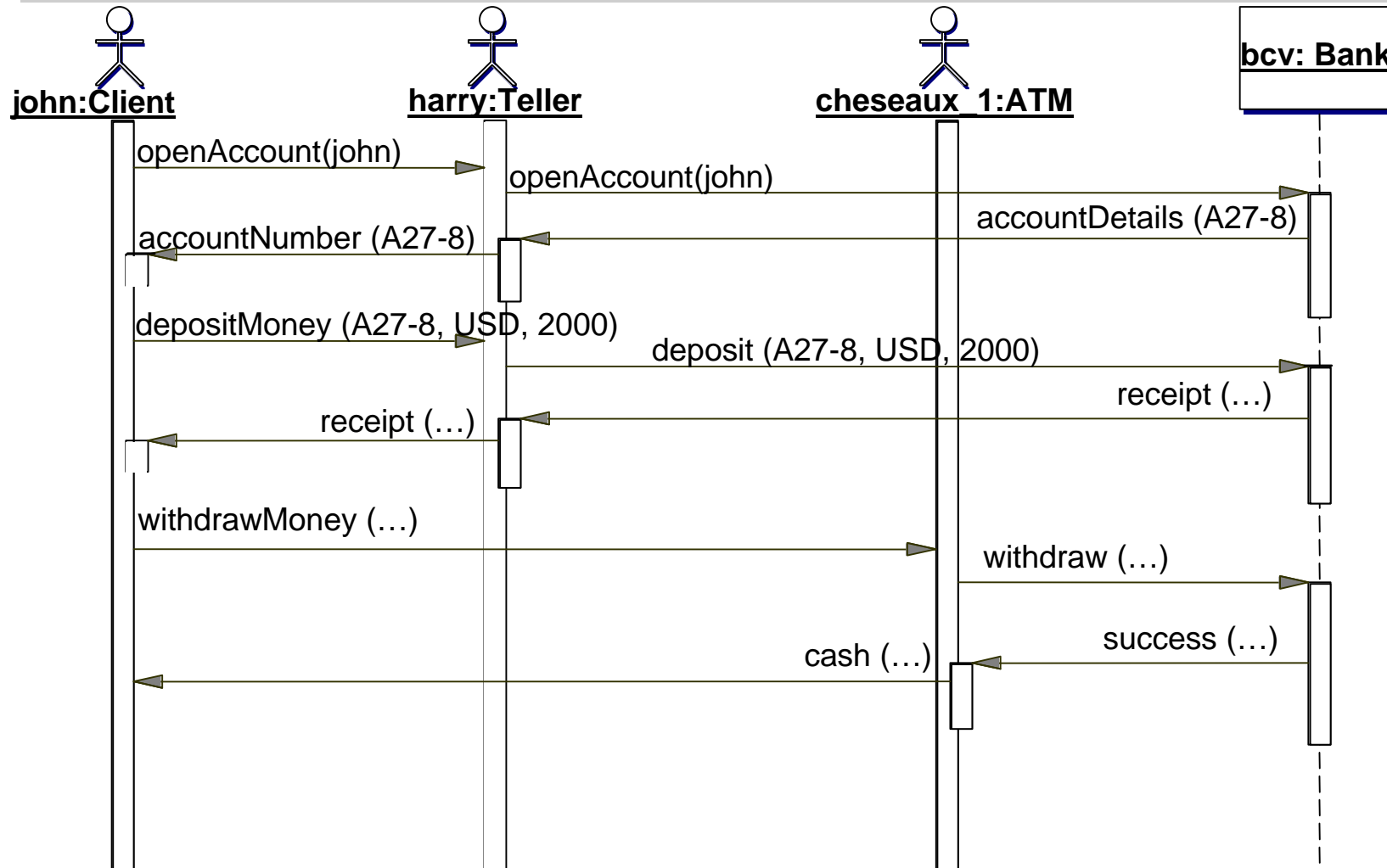
- ▲ A scenario is an ordered set of interactions between a system and a set of actors external to the system. It is comprised of a concrete “sequence” of interaction steps, where all specifics are given names.
- ▲ A scenario is a particular performance of a use case (instance), and represents a single path through the use case.

Scenarios

- ▲ Scenarios are a useful tool.
 - ◆ provide a paper prototype
 - ◆ maybe easier to start with (concrete) scenarios and then generalize (for users also)
 - ◆ are used for testing
- ▲ Scenarios are commonly depicted using UML sequence diagrams.

Scenario using a UML Sequence Diagram

SWEED



What is it?

SWEED

Use Case

- ▲ Use cases capture who (actor) does what (interaction) with the system, with what purpose (goal), without dealing with system internals. [Malan et al.'99]
- ▲ A use case:
 - ◆ achieves a single, discrete, complete, meaningful, and well-defined task of interest to an actor
 - ◆ is a pattern of behavior between some actors and the system—a collection of potential scenarios
 - ◆ is written in domain vocabulary
 - ◆ defines purpose and intent (not concrete actions)
 - ◆ is generalized and technology-free

Use Case

- ▲ Cockburn (2001) highlights that effective use cases are goal-based:
 - ◆ A use case is a description of the possible sequences of interaction between the system under discussion and external actors, related to the **goal of one particular actor**.

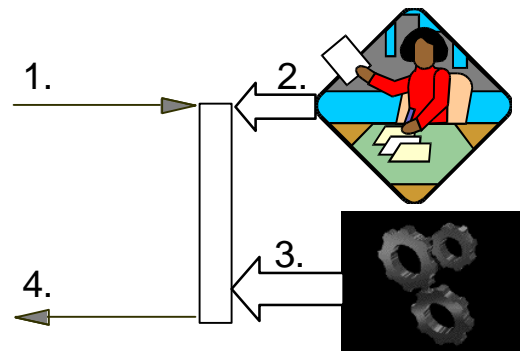
Use Case

Jacobson (1992):

- ◆ A use case is a sequence of *transactions* performed by a system, which yields an observable result of value for a particular actor.
- ◆ A *transaction* consists of a set of actions performed by a system. A transaction is invoked by a stimulus from an actor to the system, or by a timed trigger within the system.

Use Case

- ▲ A *transaction* consists of 4 steps :
 - ◆ 1. The primary actor sends the request and the data to the system;
 - ◆ 2. The system validates the request and the data;
 - ◆ 3. The system alters its internal state;
 - ◆ 4. The system replies to the actor with the result.



Use Case Description

- ▲ Use cases are primarily textual descriptions.
 - ◆ More than just an ellipse drawn in a UML diagram!
- ▲ Use case steps are written in an easy-to-understand structured narrative using the vocabulary of the application domain.
- ▲ Use cases are clear, precise, generalized, and technology-free descriptions.
- ▲ A use case sums up a set of scenarios:
 - ◆ Each scenario goes from trigger to completion.


Use Case Description

- ▲ It includes
 - ◆ How the use case starts and ends
 - ◆ The context of the use case
 - ◆ The actors and system behavior described as intentions and responsibilities
 - ◆ All the circumstances in which the primary actor's goal is reached and not reached
 - ◆ What information is exchanged

Granularity of Use Cases

- ▲ Cockburn (2001) identified three different goal levels:
 - ◆ summary level is the 50,000 feet perspective,
 - ◆ user-goal level is the sea-level perspective,
 - ◆ subfunction is the underwater perspective.

Granularity of Use Cases

- ▲ Summary level use cases: 
 - ◆ are large grain use cases that encompass multiple lower-level use cases; they provide the context (lifecycle) for those lower-level use cases.
 - ◆ they can act as a table of contents for user goal level use cases.

Example: Summary Goal Level Use Case

SWEED

Use Case: Manage Funds By Bank Account  

Scope: Bank Accounts and Transactions System

Level: Summary

Intention in Context: The intention of the Client is to manage his/her funds by way of a bank account. Clients do not interact with the System directly; instead all interactions go through either: a Teller, a Web Client, or an ATM, which one depends also on the service.

Primary Actor: Client

Main Success Scenario:


1. Client opens an account.

Step 2 can be repeated according to the intent of the Client

2. Client performs task on account.

3. Client closes his/her account.

Granularity of Use Cases

- ▲ User-goal level use cases: 
 - ◆ describe the goal that a primary actor or user has in trying to get work done or in using the system.
 - ◆ are *usually* done by one person, in one place, at one time; the (primary) actor can normally go away happy as soon as this goal is completed.

Example: User Goal Level Use Case SWEED

Case

Use Case: Deposit Money  

Scope: Bank Accounts and Transactions System

Level: User Goal

Intention in Context: The intention of the Client is to deposit money on an account. Clients do not interact with the System directly; instead, for this use case, a client interacts via a Teller. Many Clients may be performing transactions and queries at any one time.

Primary Actor: Client

Main Success Scenario:

1. Client requests Teller to deposit money on an account, providing sum of money.
2. Teller requests System to perform a deposit, providing deposit transaction details*.
3. System validates the deposit, credits account for the amount, records details of the transaction, and informs Teller.

Example: User Goal Level Use Case

Case

SWEED

Extensions:

2a. Client requests Teller to cancel deposit: use case ends in failure.

3a. System ascertains that it was given incorrect information:

3a.1. System informs Teller; use case continues at step 2.

3b. System ascertains that it was given insufficient information to perform deposit:

3b.1. System informs Teller; use case continues at step 2.

3c. System is not capable of depositing (e.g. transaction monitor of System is down)**:


3c.1. System informs Teller; use case ends in failure.

Notes:

* a hyperlink to a document that contains data details and formats.

** this is an example of an IT infrastructure failure, we only write it in a use case if there is a corresponding project constraint that states a physical separation, e.g., transaction section depends on a legacy system which is located somewhere else.

Granularity of Use Cases

- ▲ Subfunction level use cases 
 - ◆ provide “execution support” for user-goal level use cases; they are low-level and need to be justified, either for reasons of reuse or necessary detail.

Example: Sub-Function Goal Level Use Case

SWEED

Use Case: Identify Client  

Scope: Automatic Teller Machine (ATM for short)

Level: Sub-Function

Intention in Context: The intention of the Client is to identify him/herself to the System. A project (operational) constraint states that identification is made with a card and a personal identification number (PIN).

Primary Actor: Client

Main Success Scenario:

1. Client provides Card Reader with card; Card Reader informs System of card details*
2. System validates card type.
3. Client provides PIN to System.
4. System requests BAT System to verify identification information*
5. BAT System informs System that identification information is valid, and System informs Client.

Example: Sub-Function Goal Level Use Case

SWEED

Extensions:

(1-6)a. (at any time) Client cancels the identification process.

(1-6)a.1. System requests Card Reader to eject card; use case ends in failure.

2a. System ascertains that card type is unknown:

2a.1. The System informs the Client and requests the Card Reader to eject the card; the use case ends in failure.

2b. System informs Client that it is currently "out of service": use case ends in failure.

3a. System times out on waiting for Client to provide PIN:

3a.1. System requests Card Reader to eject card; use case ends in failure.

5a. BAT System informs System that password is incorrect:

5a.1a. System informs Client and prompts him/her to retry; use case continues at step 3.

5a.1b. System ascertains that Client entered an incorrect PIN for the third time:

5a.1b.1. System swallows card and informs Client to see Bank for further details; use case ends in failure.

Example: Sub-Function Goal Level Use Case

SWEED

5b. BAT System informs System that card information is invalid:

5b.1. System informs Client and requests Card Reader to eject card; use case ends in failure.

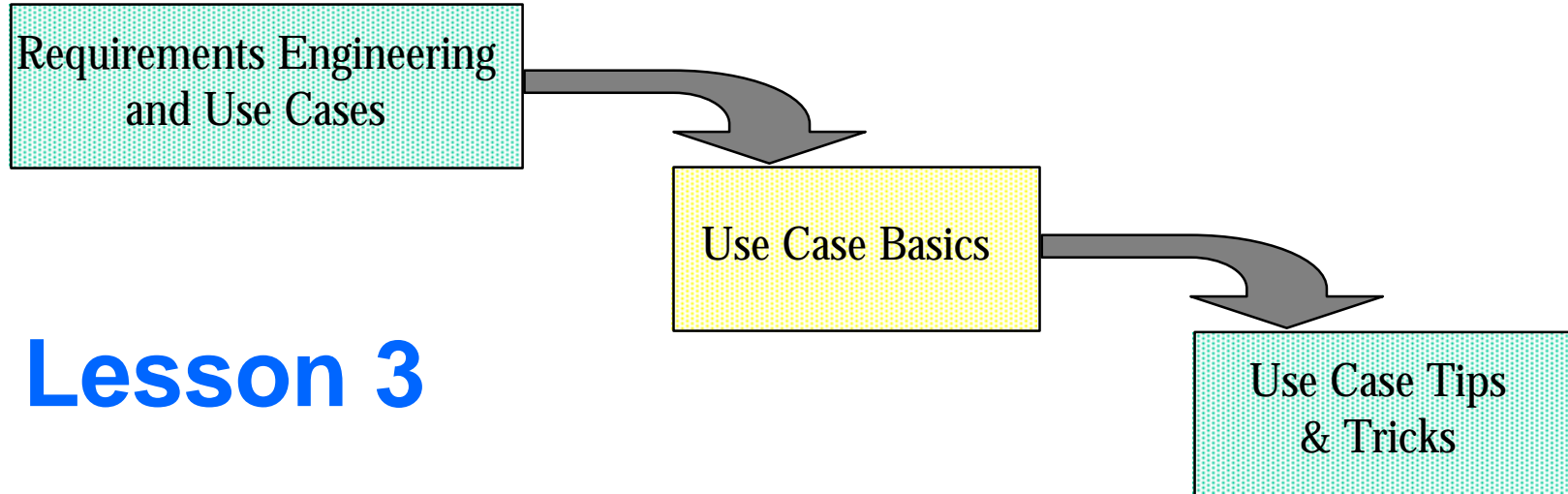
5c. System is unable to communicate with BAT System:

5c.1. System informs Client that it is now out of service and requests Card Reader to eject card; use case ends in failure**.

Notes:

* Data details and formats are recorded in another document (I would normally provide a hyperlink to this information; do not clutter the use case with this information)

** this is an open issue on what the System is to do when confronted with this situation, e.g., does it go "out of service" and poll BAT System? or does it just go "out of service" until Maintenance comes to put it back online?

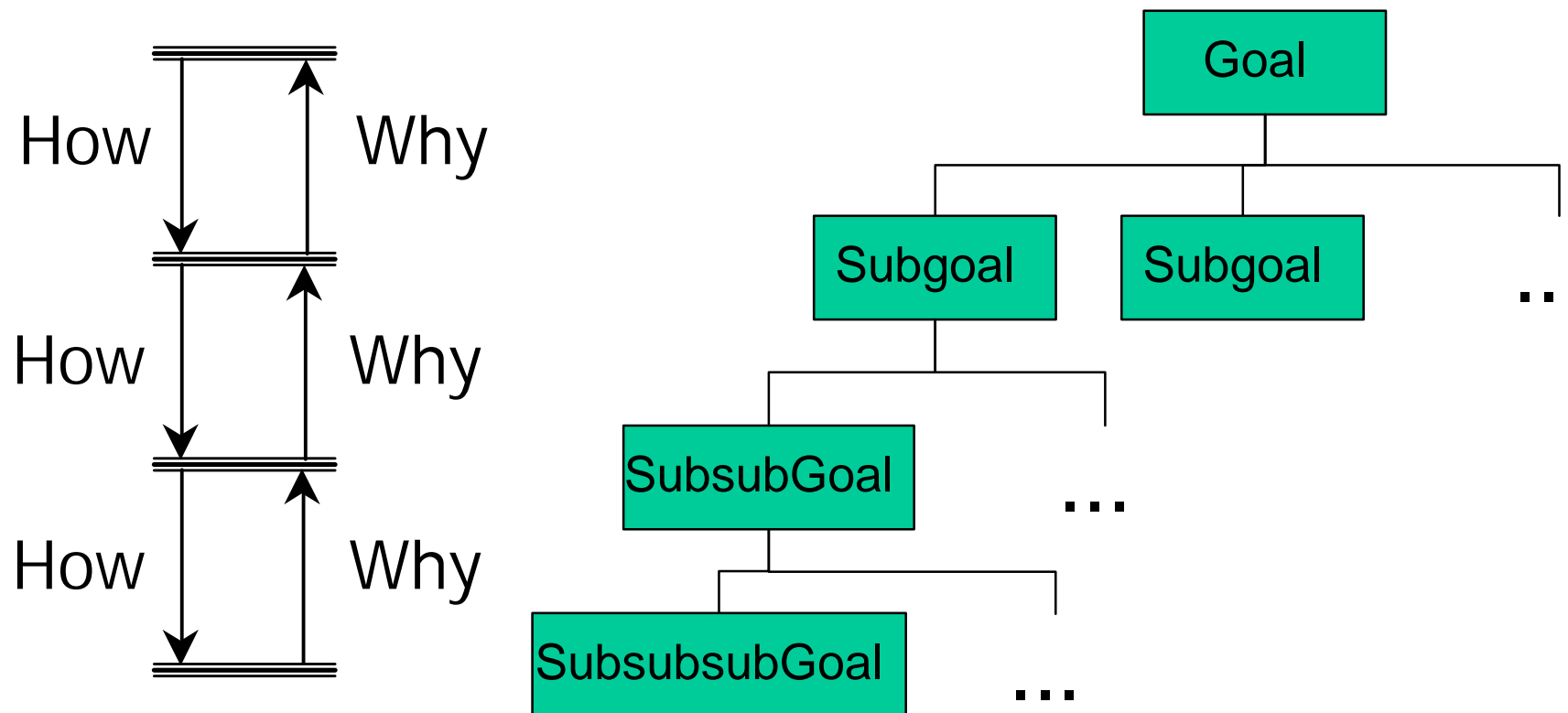


Lesson 3




Use Case Basics II

What versus How

- ▲ The “How” at one level of abstraction forms the “Why” for the next level down

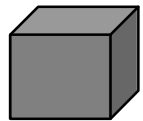


Scope of Use Cases

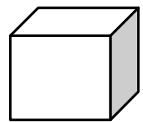
- ▲ The **enterprise** as the system boundary for use cases (also known as **business use cases**).
 - ◆ provides the context in which the system under development is involved
 - ◆ shows how the system under development will add value to the outside world
- ▲ Transparency of system internals:
 - ◆ Black-box: if you want to treat the whole enterprise as a black-box; interaction is strictly between system and external actors. 
 - ◆ White-box: if you talk about the staff, the sections (or departments), and systems within the organization.  

Scope of Use Cases

- ▲ The **software system** as boundary for use cases (also known as **system use cases**).
- ▲ Choice in transparency of system internals for use cases:



- ◆ Black-box: if you want to treat the software system as a black-box; interaction is strictly between system and actors.



- ◆ White-box: if you reveal the components of the system, which may consist of other systems and subsystems.



Scope of Use Cases



- ▲ A **component of the software system** to develop as boundary for use cases
- ▲ Always a black-box description

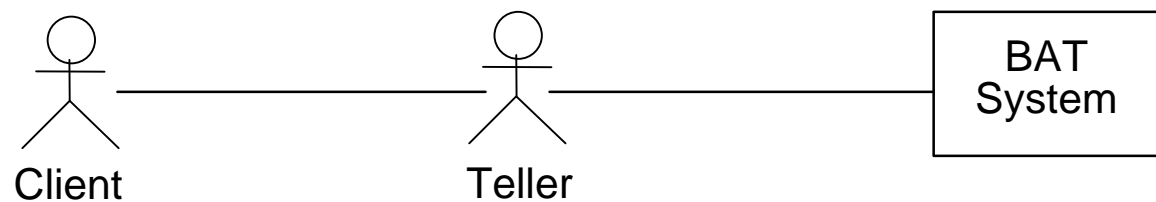
Abstraction Levels of Actors

▲ Business Actors

- ◆ the business entity that interacts with the business

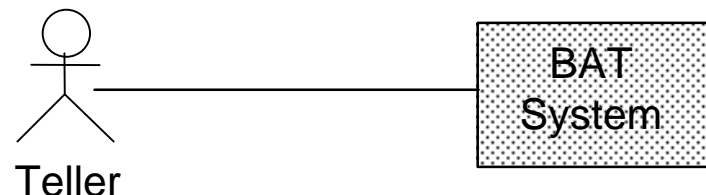
▲ System Actors

- ◆ has direct interaction with the system



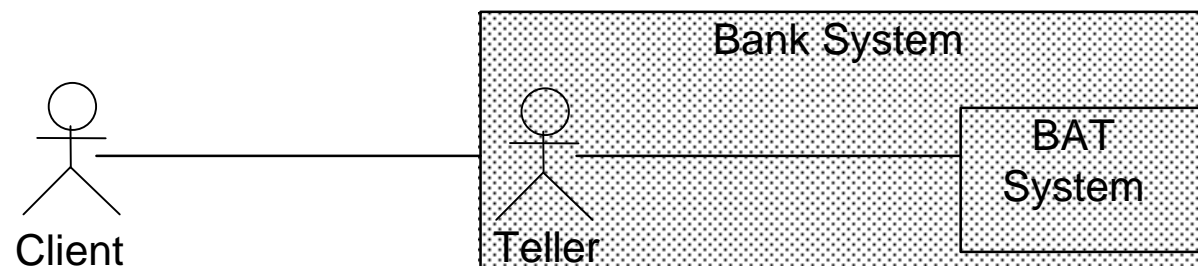
Abstraction Levels of Actors

1. The Teller requests the system to perform a deposit, providing the deposit transaction details.
2. The System validates the deposit, credits the account for the amount, records the details of the transaction, and informs the Teller.



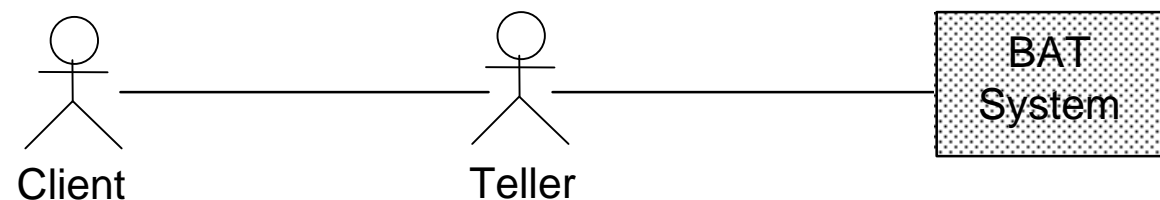
Abstraction Levels of Actors

1. The Client requests System to deposit money on an account, providing a certain amount of money.
2. The System validates the deposit, credits the account for the amount, records the details of the transaction, and informs the Client of a successful deposit.



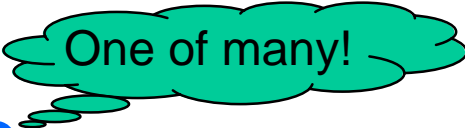
Abstraction Levels of Actors

1. The Client requests the Teller to deposit money on an account, providing a certain amount of money.
2. The Teller requests System to deposit, providing the deposit transaction details.
3. The System validates the deposit, credits the account for the amount, records the details of the transaction, and informs the Teller.



Differentiating the Two Levels

- ▲ Does the entity primarily just relay information? Or is significant value added?
- ▲ Does the definition of the actor provide critical context for the use case?
- ▲ Is it likely that the actor will be replaced by an automated interface?
- ▲ Is the event, from the actor's perspective, focused just on the system, or does it encompass multiple systems?



One of many!

SWEED

Use Case Template

- ▲ Our preference: A slightly adapted version of Cockburn's "fully dressed" use case template:
 - ◆ One column (no table)
 - ◆ Sequenced: Numbered steps (Dewey decimal numbering system) and extensions to main scenario use alphabetic letters to differentiate from main steps
 - ◆ The clauses *Stakeholders' interests* through to *Trigger*, and *Extensions* and *Notes* are optional



One of many!

SWEED

Use Case Template

Use Case: Name of the use case. This is the goal stated by a short active verb phrase.

Scope: The scope of the “system” being considered (black-/white-box and enterprise/system/component).

Level: Summary, User-goal, or Subfunction

Intention in Context: A statement of the primary actors intention and the context within which it is performed.

Primary Actor: The primary actor of the use case.

Stakeholders’ Interests: The list of stakeholders and their key interests in the use case.

Precondition: What we can assume about the current state of the system and environment.

Use Case Template

Minimum Guarantees: How the interests of the stakeholders are protected in all circumstances.

Success Guarantees: The state of the system and environment if the goal of the primary actor succeeds.

Trigger: What event starts the use case.

Main Success Scenario:

<step_number> "." <action_description>

The numbered steps of the scenario, from trigger to goal delivery, followed by any clean-up.

Conditions and alternatives are shown in the extension part.

Extensions:

<step_altered> <condition> ":" <action_description> or <sub-use_case>

Use Case Template

(Extensions cont'd)

Steps can be nested. Dewey numbers are then used, e.g. 3a.1

An extension always refers to a step of the Main Success Scenario.

An extension step takes place in addition to the respective main step,

notation: 2 ||,

or as an alternative,

notation: 2a.

An extension might correspond to regular behavior, exceptional behavior that is recoverable, or unrecoverable erroneous behavior.

Notes:


Provide additional noteworthy information.

Use Case Template

- ▲ The list of clauses is not strict; other clauses could be added:
 - ◆ Frequency Of Occurrence
 - ◆ Technology and Data Variations
 - ◆ UI Links
 - ◆ Calling Use Cases
 - ◆ Open Issues
 - ◆ Version (Number, Date, Author, Reviewers)
 - ◆ Secondary Actors
- ▲ General Rule: Keep your template as slim as possible.

Example of Fully Dressed Use Case SWEED

Case

Use Case: Buy items online  

Scope: Web Ordering System

Level: User Goal

Intention in Context: The intention of the Shopper is to shop for goods on Acme's online shopping site.

Primary Actor: Shopper

Stakeholders' Interests:

Shopper: Get desired items for a good price.

Acme: Sell as many items as possible (fixed price).

Minimum Guarantees: The System has a log of all the item selections and queries made by the Shopper*.

The ordered items will only be delivered once payment has been transferred to Acme's account.

Success Guarantees: System has received payment confirmation and the System has notified the Warehouse for delivery of the ordered items to the Shopper.

Example of Fully Dressed Use Case

Case

SWEED

Precondition: Shopper has identified him/herself to the System.

Trigger: Shopper requests information on a particular product.

Main Success Scenario:

The Shopper repeats steps 1-4, as many times as desired, to navigate to and select different items, adding them to the shopping cart as desired.

1. Shopper requests System for information on product.
2. System provides Shopper with requested information on product.
3. Shopper requests System to add it to his/her shopping cart.
4. System adds it to shopping cart and presents a view of shopping cart and items in it*.
5. Shopper requests System to check-out his/her shopping cart.
6. System debits Shopper's credit card with purchase price, and passes order on to Warehouse for delivery to Shopper.

Example of Fully Dressed Use Case

Case

SWEED

Extensions:

(1-4)||a. Shopper requests System to change contents of shopping cart:

(1-4)||a.1. System permits Shopper to change quantities, remove items, or go back to an earlier point in selection process; use case continues from where it was interrupted.

(1-5)a. System detects that Shopper has left: use case ends in failure.

4||a. System detects that item is not in stock:

4||a.1. System requests Warehouse to replenish the stocks for the item and informs User that item is on back-order; use case continues from where it was interrupted.

6a. System fails to debit Shopper's credit card:

6a.1. System informs Shopper; use case ends in failure.

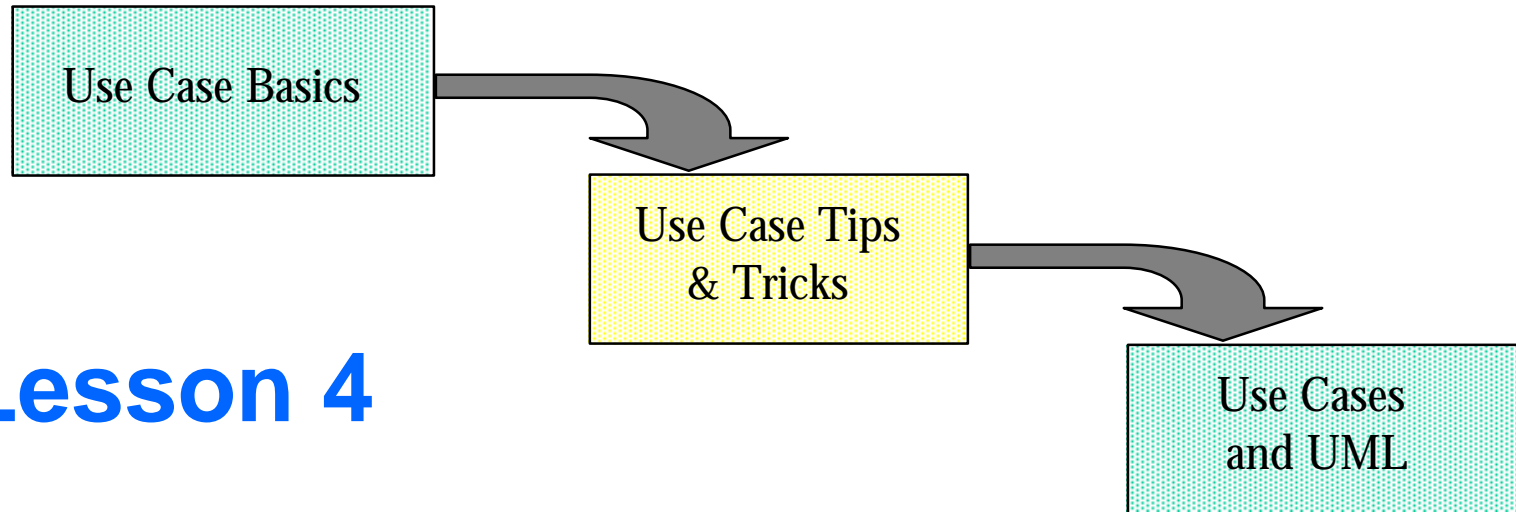
Notes:

* Details on what information is required/offered is given in data details and formats documents (preferably hyperlinked)

Use Cases as a Goal-based Approach

SWEED

- ▲ Focus on why system is being constructed
 - ◆ high level objectives of business or organization right through to fine-grained goals of users
 - ◆ take into account stakeholder interests
 - ◆ explicit declaration of goals and interests provides basis for conflict resolution
- ▲ Primary actors interact to achieve their goals, the system supports these goals and possibly provides alternatives (as backups)
- ▲ But must keep boundary & constraints in mind:
 - ◆ project constraints: e.g. business rules, mandated COTS, existing interfaces



Lesson 4

Use Case Tips and Tricks

Use Case Tips and Tricks

- ▲ General & Style Tips
- ▲ Process Suggestions
- ▲ FAQs
 - ◆ When to Stop Decomposing
 - ◆ How Many is Enough
 - ◆ How Formal Do the Use Cases Have to Be
 - ◆ How Large is a Use Case
 - ◆ When are Use Cases Not Suitable
 - ◆ Avoiding Functional Decomposition Design
- ▲ CRUD Use Cases
- ▲ Commonly Forgotten Functionality

General Tips

- ▲ Make sure the context is clear!
- ▲ Make sure the use case is clear to each and every stakeholder
- ▲ Iteration is the key to effective use cases: precision, consistency, and readability
- ▲ Make a clear distinction between business and system use cases
 - ◆ Remember two-level contract
- ▲ Describe use cases from primary actor's point of view

General Tips

- ▲ Make a clear separation between actor's intentions and responsibilities, and those of the system, thus highlighting the system boundary.
- ▲ Hyperlinks are very useful for relating use cases to other documents (business rules, data details and formats, etc.)
 - ◆ Tool Issue (the dream): browse use cases via the web; use cases, business rules, etc. stored in database (or at least version control); integrated with UML tool.

Style Tips

- ▲ Name use cases appropriately
 - ◆ name => primary actor's intention/goal
 - ◆ (when possible) do not include name of actor in use case name, but state it from actor's point of view; but there are exceptions: Identify User versus Identify Myself, e.g.
 - ◆ name use cases as “verb + noun phrase” (active verb in imperative mood, when possible)
- ▲ There may be more than a single success scenario
 - ◆ choose most probable and branch off others into the extension clause

Style Tips — Steps

- ▲ Use Simple Grammar (active voice)
 - ◆ GOOD “1. Actor requests System for rum-flavored popcorn.”
 - ◆ BAD “1. The System was requested by the Actor for some of the rum-flavored variety of popcorn.”
- ▲ State who does what to whom, i.e., the participants in the interaction should always be clear.
- ▲ State primary actor’s intent.
- ▲ Start each step “System ...” or “Actor ...”.

Style Tips — Steps

- ▲ Join parts of Jacobson's transaction whenever possible/reasonable.
 - ◆ e.g., "2. System calculates sum and prompts User for payment details."
- ▲ Use solution-free narrative (language of the application domain), unless a particular solution is required.
- ▲ Avoid "if" statements. Factor out into extensions instead.

Style Tips — Extensions Clause

- ▲ Failure scenarios can be recoverable or non-recoverable.
 - ◆ defined in terms of a variation point to the main scenario (extensions clause)
- ▲ Recoverable alternatives rejoin main scenario.
 - ◆ e.g. “use case continues at step X”, or “use case continues from where it was interrupted”
 - ◆ or join at end, e.g., “use case ends in success”
- ▲ Non-recoverable alternatives end use case in failure.

Style Tips — Extensions Clause

SWEED

- ▲ Look for ways that each step in the main scenario can fail.
 - ◆ a single step may have several alternatives
- ▲ Do not get into a white-box view just because you are dealing with failures.
 - ◆ address “business” failures, rather than IT failures
 - ◆ but: if boundaries between systems have been defined, then identify system-level exceptions

Style Tips — Extensions Clause

SWEED

- ▲ Remove all scenarios that are impossible according to preconditions,
- ▲ Remove all scenarios that cannot be detected or acted upon by the system (only for system-level use case).

Process. Actors

Task 1: Actors

- ▲ Brainstorm actors and primary actor goals
 - ◆ Taking into account the questions for identifying Primary and Secondary actors, and Initiators, Servers, Receivers, and Facilitators.
 - ◆ Make a list with each primary actor and its goals.

Process. Actors

▲ Work product 1: Actor Description

Actor	Role	Brief Description
Client	Primary	A customer of the bank that will use the system to perform transactions and queries on his/her accounts.
Bank Manager	Primary	...
Printer	Secondary	
ATM	Facilitator	
Teller	Facilitator	

Process. Actors

▲ Work product 2: Actor with Goal List

Actor	Goal
Client	open a savings account
	open a high transaction account
	deposit money on to an account
	transfer money from one account to another
	withdraw money from an account
	close an account
	get an overdraft
Bank Manager	...

Process. Stakeholders

Task 1 || (background): Stakeholders

- ▲ Brainstorm stakeholders and their interests.
 - ◆ identify stakeholders and their key interests in the system with respect to the use case
- ▲ Questions:
 - ◆ Who has a vested interest in the System?
 - Look out for individuals, groups of people, organizations, etc.
 - ◆ Are there any regulations/policies to deal with?

Process. Stakeholders

▲ Work product 3: Stakeholder with Concern List

Stakeholder	Concern
Client	keep money secure
	have high interest
	pay low bank charges
	have high access possibilities
Bank	make the largest possible profit: - low interest - high charges
	ensure good reputation with customers: - secure - good service - ...
Government	not allow money laundering
	...

Process. UC Outlines

Task 2: Use Case Outlines

- ▲ Construct (summary and/or user-goal) use cases briefs for each actor goal on the system, making the actor the primary one.
- ▲ Always ask “why” in order to find the next level up!
- ▲ Questions:
 - ◆ What measurable value/service is needed by the actor?
 - ◆ What are the actors intentions?
 - ◆ Why do the actors do what they do?

Process. UC Outlines

Work Product 4: Prioritized Use Case List

Actor	Goal	Goal Description	Business Need	Difficulty	Priority	UC #
Client	open a savings account	open a new savings account with the bank	Medium	Simple	3	1
	open a high transaction account		Top	Simple	1	2
	deposit money on to an account		High	Medium	2	3
	transfer money from an account to another		High	Difficult	4	4
	withdraw money from account		High	Medium	2	5
	close an account		Top	Simple	1	6
Bank Manager	...					

Process. UC Bodies

Task 3: Use Case Bodies

- ▲ Capture each actor's intent and responsibility—from trigger to goal delivery.
- ▲ For each use case, fill in the main success scenario before the extensions.
 - ◆ The extensions take the most time; brainstorming activities with group members are a good way to find alternatives—successful and erroneous ones (recoverable or non-recoverable).
 - ◆ Identify all failure conditions before failure scenarios.
 - ◆ Ask “what can go wrong?”
 - ◆ Iteration/refactoring is the key: use cases are always better next time around.

Process. UC Structuring

Task 4: Use Case Structuring

- ▲ For each use case:
 - ◆ if the main success scenario of the use case is greater than 9 steps, collect steps that encapsulate a sub-goal of the primary actor and create a new lower-level use case with the steps.
 - ◆ Inversely, if the use case is smaller than 3 steps, think about expanding it or putting it back in the calling use case.
- ▲ The most important thing is that the steps of the use case have a consistent level of description, no matter what the level!

Process. Checks

Task 5: Checks

- ▲ Apply the checklist to each use case (shown next slide); see also [Armour et al. 2001, Appendix A]
- ▲ Review each use case:
 - ◆ Is its purpose and intent clear?
 - ◆ Is its context clear?
 - ◆ Is it written in a clear and precise way?
 - ◆ Is it written using the vocabulary of the application domain and abstract away from technology?
 - ◆ Is it complete, correct, consistent, verifiable?
 - ◆ Does it achieve a single, discrete, complete, meaningful, and well-defined task of interest to an actor?

Check-list

Field	Question
Use case title.	<p>1 Is the name an active-verb goal phrase, that expresses the goal of the primary actor?</p> <p>2 Can the enterprise/system/component deliver that goal?</p>
Scope.	<p>3 Is the system boundary clear, i.e., do the developers have to develop everything in the Scope, and nothing outside it?</p>
Level.	<p>4 Does the use case content match the goal level stated in Level?</p> <p>5 Is the goal really at the level mentioned?</p>
Intention in Context	<p>6 Has it been clearly stated what other use cases may be executing at the same time?</p>
Primary actor.	<p>7 Does the named primary actor have behavior?</p> <p>8 Does the primary actor have a goal against the system under development that is a service promise of the system?</p>

Check-list

Field	Question
Preconditions	9 Are they assumptions and not guards?
Minimum Guarantees	10 Are all the stakeholders' interests protected?
Success Guarantees	11 Are all stakeholders' interests satisfied?
Main success scenario	12 Does it have less than 10 steps? 13 Does it run from trigger to delivery of the success guarantee?
Each step in any scenario	14 Is it phrased as a goal that succeeds? 15 Does the process move distinctly forward after successful completion of the step? 16 Is it clear which actor is operating the goal? 17 Is the intent of the actor clear?

Check-list

Field	Question
Each step in any scenario	<p>18 Is the goal level of the step lower than the goal level of the overall use case? Is it, preferably, just a bit below the use case goal level?</p> <p>19 Are you sure the step does not describe the user interface design of the system?</p> <p>20 Is it clear what information is being passed?</p> <p>21 Does the step "validate", as opposed to "check", a condition?</p>
Extension condition.	<p>22 Can and must the system detect and handle it?</p>
Overall use case content.	<p>23 To the sponsors and users: "Is this what you want?"</p> <p>24 To the sponsors and users: "Will you be able to tell, upon delivery, whether you got this?"</p> <p>25 To the developers: "Can you implement this?"</p>

FAQs

- ▲ When to Stop Decomposing?
 - ◆ User goals are the aim of the game, only go lower if you can justify it (necessary detail and reuse of commonality)
- ▲ How Formal Do the Use Cases Have to Be?
 - ◆ Depends on project (type & size) and stage in development (goes from informal to formal)
 - ◆ Formalize when:
 - project members work separately but communicate through models
 - more than 20 use cases

FAQs

▲ How Many is Enough?

- ◆ Cockburn: How many user goals does the system have?
- ◆ Anderson and Fertig: no more than 80 for any subsystem

▲ When are Use Cases Not Suitable?

- ◆ Use cases without an end user ? (e.g., clock-triggered)
- ◆ Systems with few actors and long running processes
 - Systems which are “all” algorithm, e.g., scientific computation
 - Continuous process/control systems, e.g., stream treatment

FAQs

▲ How Large is a Use Case?

- ◆ Under 10 steps
- ◆ scope: business, system, component
- ◆ level: summary, user-goal, sub-function

▲ Do my use cases have a sufficient level of detail?

- ◆ Ask the following questions:
 - Could system or acceptance test scripts be generated easily from the use cases?
 - Do you have sufficient information to move onto the next development activity (at least for the high-priority use cases)?

FAQs

- ▲ What does Functional Decomposition Design have to do with use cases and how does one avoid it?
 - ◆ Decomposing use cases is a kind of functional decomposition
 - ◆ Don't fall into the trap of a naïve mapping between use cases and system structure
 - The temptation is to base the design on use cases; the results are usually enormous control objects, no reuse of functionality and duplication of objects.
 - ◆ Remember design is not supposed to be easy
 - decisions and trade-offs must be made

Create Read Update Delete (CRUD) Use Cases

SWEED

- ▲ Create, Read, Update and Delete are performed on a common (business) object, but each one corresponds to a separate goal.
- ▲ Cockburn suggests starting with a higher-level use case (often summary), Manage *<business object>*
 - ◆ Easier to track
 - ◆ Break out any complex CRUD units into a new use case
- ▲ But in the hunt, don't put CRUD use cases first, instead keep focused on use cases that provide the most value to the primary actors.

Commonly Forgotten Functionality

SWEED

- ▲ Security
 - ◆ Authentication and authorization of users
- ▲ Audit
 - ◆ Logs of online or batch activity
- ▲ Backup and Recovery
 - ◆ Creating and maintaining copies of system data
- ▲ Remote Users
 - ◆ Interactions of customers or supply chain partners
- ▲ Reporting requirements
 - ◆ queries and reports

Use Case Tips
& Tricks

Use Cases
and UML

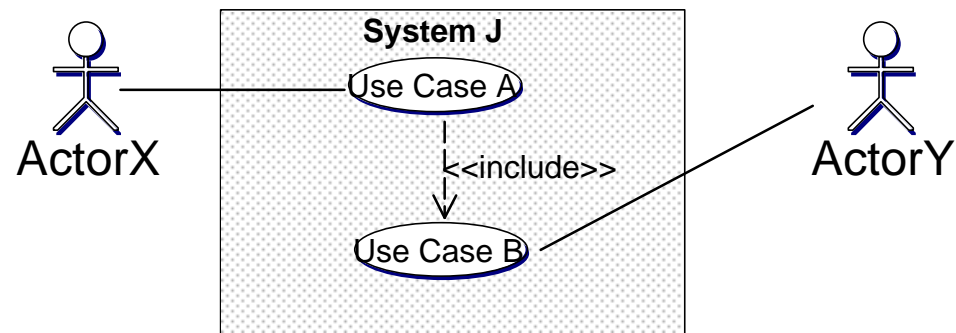
Advanced Issues in
Writing Use Cases

Lesson 5

Use Cases in UML

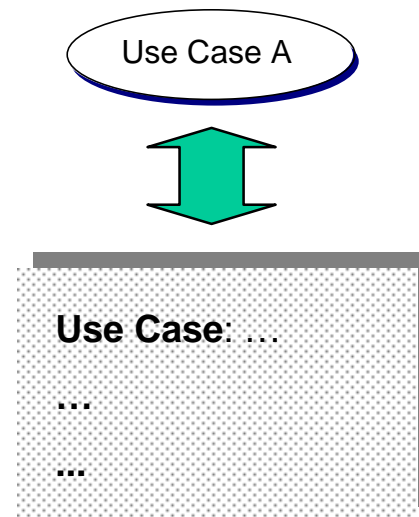
Use Cases in UML

- ▲ UML provides a graphical representation for use cases called the use case diagram.
- ▲ It allows one to graphically depict:
 - ◆ actors,
 - ◆ use cases,
 - ◆ associations,
 - ◆ dependencies,
 - ◆ generalizations,
 - ◆ packages,
 - ◆ and the system boundary.



Use Case Model

- ▲ A Use Case Model consists of:
 - ◆ a use case diagram and
 - ◆ use case descriptions



Use Case Diagram

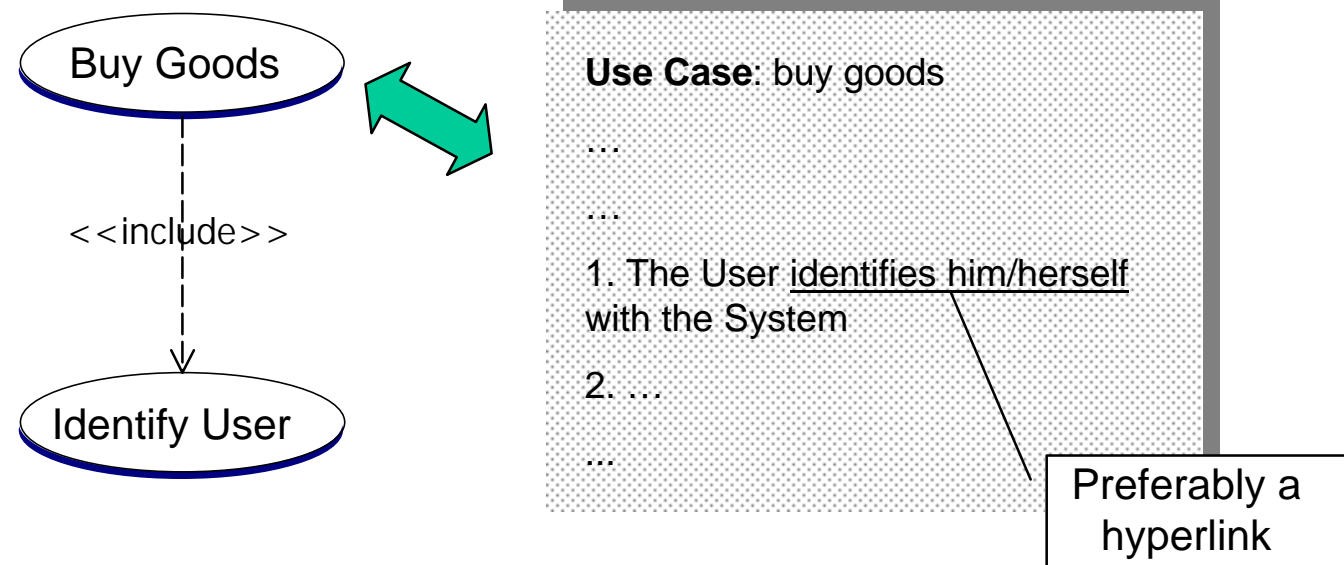
- ▲ A Use Case Diagram is used in UML to give an overview of the use cases in focus, from which allocation of work can be partitioned, for example.
- ▲ Association:
 - ◆ Unbroken line between actor and use case
- ▲ Dependency:
 - ◆ Broken directed line between two use cases
- ▲ Generalization:
 - ◆ As usual, an unbroken directed line with closed arrow either between use cases or between actors

Relationships between Use Cases

- ▲ Three relationships that can be used to structure use cases: extends, includes, and generalization/specialization.
 - ◆ help to avoid duplication of work and the related inconsistencies
 - ◆ try to direct one towards a more object-oriented view of the world rather than towards functional decomposition
 - ◆ For a good discussion of these relationships, see [Armour et al. '01]

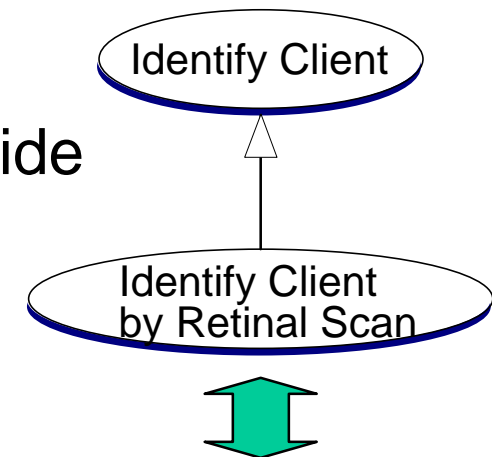
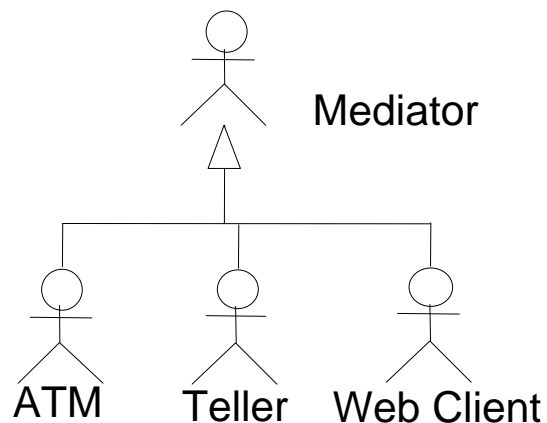
Includes Relationship

- ▲ An include relationship means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base.



Generalization Relationship

- ▲ “A generalization relationship is between a general thing and a more specific kind of that thing” [Booch ‘99]
 - ◆ it means the child may add to or override the behavior of its parent

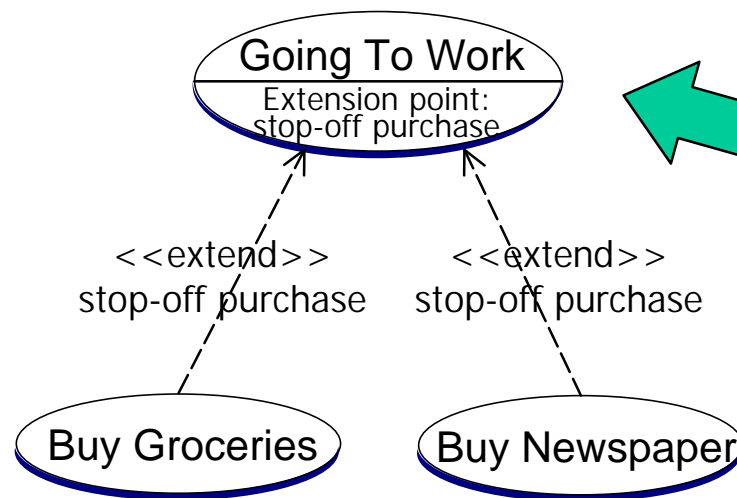


Use Case: identify client by retinal scan is a identify client

....

Extend Relationship

- ▲ An extend relationship means that the base use case implicitly incorporates the behavior of another use case at a specified location.



Use Case: going to work

...

Main Success Scenario:

...

4. Worker leaves train station

...

Extensions:

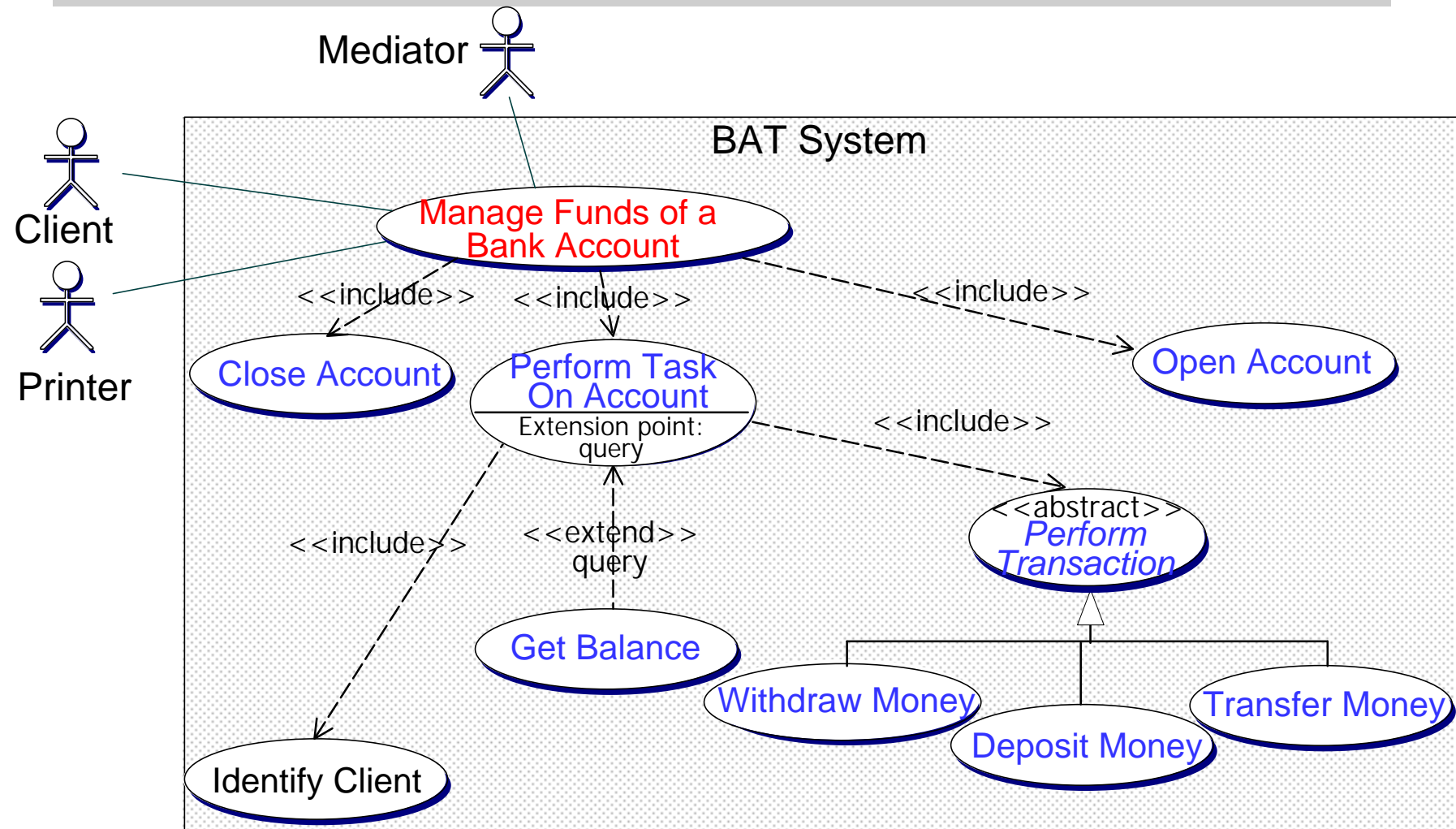
...

4||a. The Worker makes a purchase
[**extension point:** stop-off purchase]

...

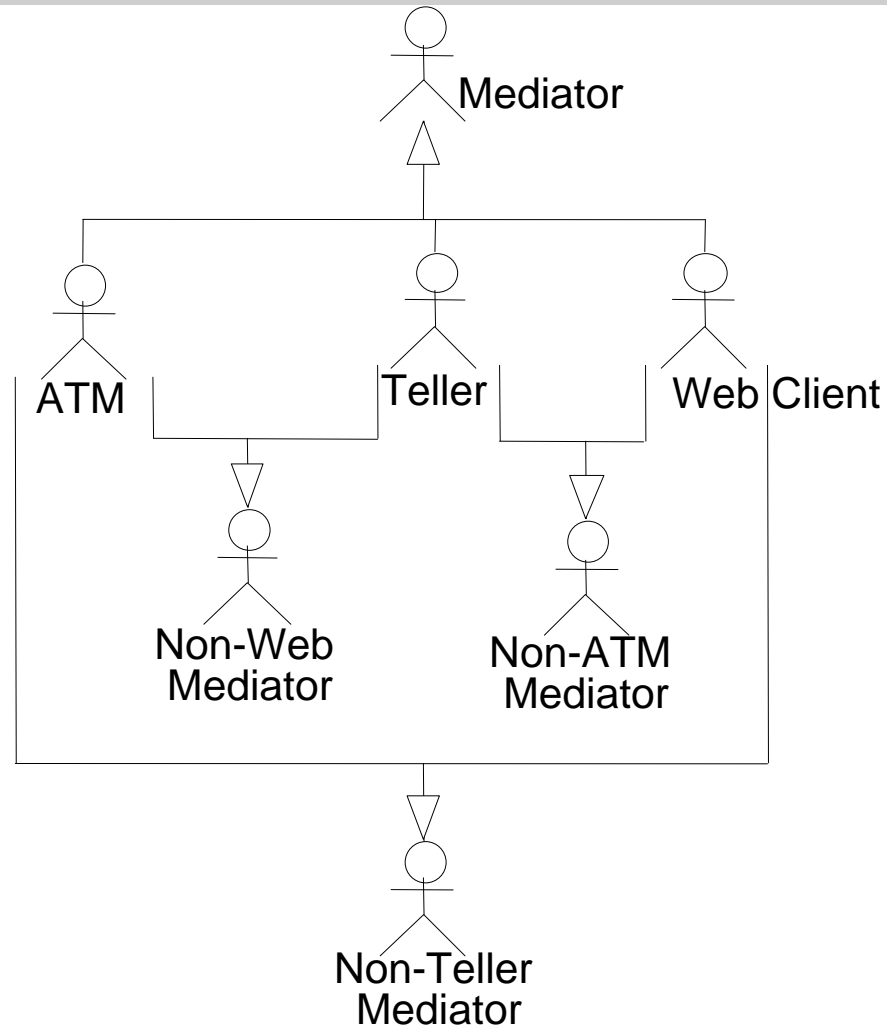
UML Use Case Diagram for BAT System

SWEED



UML Use Case Diagram for BAT System

SWEED



Scheduling and Organizing Use Cases

SWEED

▲ Scheduling use cases:

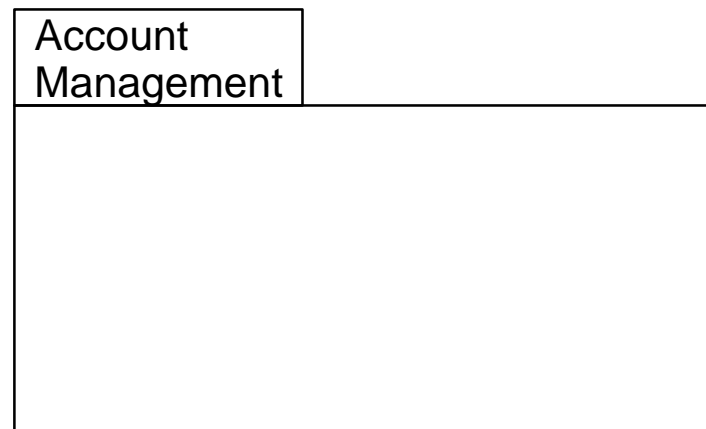
- ◆ Use cases can be prioritized and given release numbers (can use different colors on UML diagram).

▲ Organizing use cases:

- ◆ Large use case models may result in a mass of information that can be difficult to follow, and it might be hard to pinpoint the right information quickly.
- ◆ Use cases can be organized into logical groupings.
- ◆ Structuring is useful for both a bottom-up and a top-down approach

Packaging Use Cases in UML

- ▲ Packages in UML can be used for partitioning use cases into logical groupings.
- ▲ A package name should reflect the properties common to its contents.

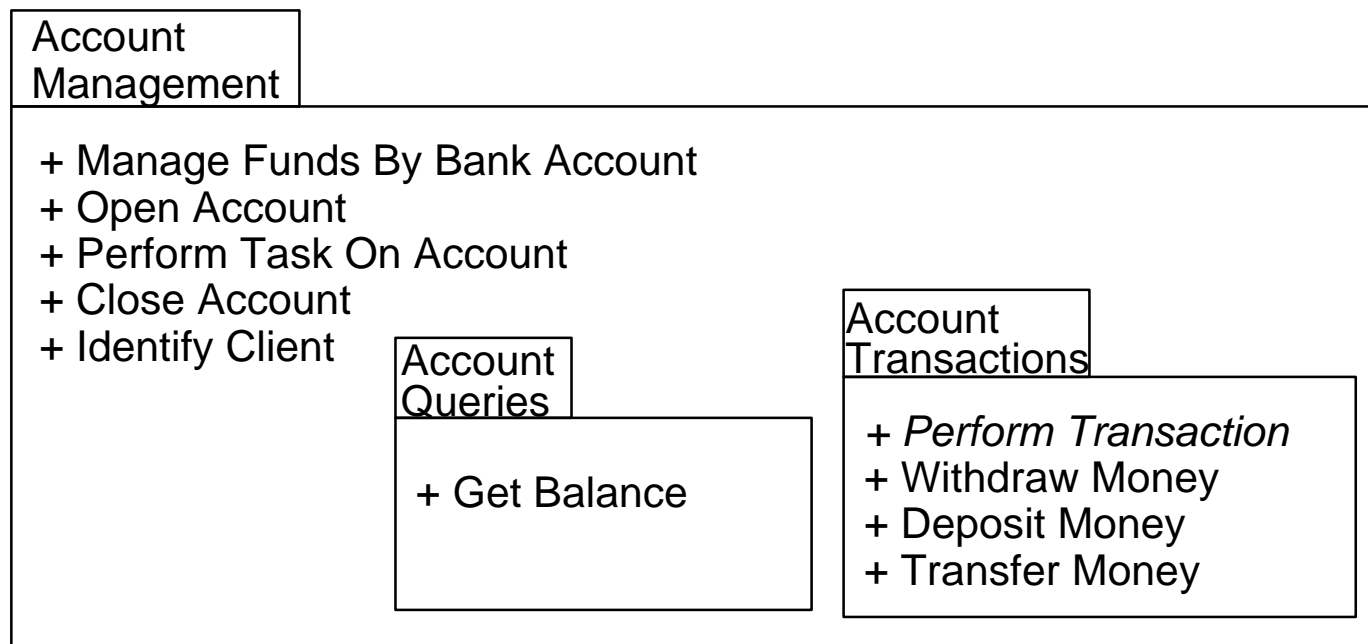


Packaging Use Cases in UML

- ▲ Clustering techniques:
 - ◆ By (Primary) Actor
 - as long as the ratio primary actor - use case is fair (not more than 80 - 100 use cases)
 - ◆ By Summary Level Use Case
 - use cases naturally cluster by their lifecycle
 - ◆ By Development Team and Release
 - clustering use cases by development team and release number simplifies work tracing

Packaging Use Cases in UML

- ▲ Clustering techniques (continued):
 - ◆ By Subject Area
 - subject areas are usually intuitive



Use Cases
and UML

Advanced Issues in
Writing Use Cases

Relating Use Cases with
BPM & with NFRs

Lesson 6

Advanced Issues in Writing Use Cases

|

Advanced Issues in Writing Use Cases

SWEED

- ▲ Use Case Reuse and Parameterization
- ▲ Change Cases
- ▲ Relating Use Cases to Other Development Activities
- ▲ Relation of Preconditions and Use Case Failures
- ▲ Limitations of Use Cases
- ▲ Formalizing Use Cases

Use Case Reuse and Parameterization

- ▲ Use Cases can be reused between projects (as well as within)
 - ◆ level of reuse is strongly related to the abstraction level and application domain
- ▲ Parameterized Use Cases
 - ◆ use cases that occur often in different situations that just refer to a different thing
 - ◆ E.g. UC: Find a *something* [Cockburn '01]

Change Cases

- ▲ [Ecklund '96] proposed Change Cases, i.e. Use Cases with a special purpose.
- ▲ The idea is that some system changes can be anticipated.
 - ◆ Change cases allow one to anticipate future requirements and build a better software architecture
- ▲ For each use case and business rule, there should be a note explaining potential changes and their reasons:
 - ◆ maybe even use cases beyond the scope of the current release in development

Relating Use cases to Other Development Activities

SWEED

- ▲ How do Use Cases relate to Object-Orientation
 - ◆ Use Cases may be part of UML but that does not make them object-oriented!
 - ✓ Generalization/Specialization
 - ✗ Non-seamless transition to “design” objects
 - A use case model does not force one to build an O-O system

Relating Use cases to Other Development Activities

SWEED

- ▲ Relating Use Cases to O-O Analysis
 - ◆ Use cases name the concepts needed in domain modeling and vice versa (validate each other)
 - Domain analysis is very important for establishing common vocabulary; also helps in finding the right level of detail in use cases. The result ranges from a data dictionary to a full-fledged domain class model.

Relating Use cases to Other Development Activities

SWEED

- ▲ Relating Use Cases to O-O Analysis (cont'd)
 - ◆ The approach of the Software Engineering Lab at EPFL:
Fondue Specification Work Products:
 - System Context Model
 - Analysis Class Model
 - System Operation Model
 - System Interface Protocol

Relating Use cases to Other Development Activities

- ▲ How do Use Cases relate to O-O Design
 - ◆ Use Cases express the to-come solution as it is perceived. This perception is not a full-fledged design, except for very simple systems.
 - ◆ Distribution/allocation of behavior to objects needs to be addressed, because “effective” use cases say nothing about how behavior is allocated among objects.
 - ◆ Possibilities for relating behavior to objects:
 - Entity, Interface, Control Objects [Jacobson ‘92]
 - CRC cards bounded by use cases [Bellin ‘97]
 - UML collaboration diagrams or sequence diagrams

Relating Use cases to Other Development Activities

SWEED

▲ How do Use Cases relate to O-O Design

◆ Some issues to be addressed:

- Don't assume that the extend/generalize relationships shown in the use cases will translate into inheritance relationships in the design class diagram.
- Don't assume included use cases will translate to specific classes that should be extracted and assigned the corresponding responsibilities.
- Care must be taken that use cases are not too abstract (developers need to know all the requirements), or
- the inverse, too concrete, leaving no room for design freedom.

Relating Use cases to Other Development Activities

SWEED

- ▲ How do Use Cases relate to Testing
 - ◆ Use Cases are a good source for black-box test cases.
 - ◆ There should be a test case for all important scenarios.
 - ◆ Extension conditions lead to test cases that need to be created to ensure that the named condition is correctly handled by the system.

Testing with Use Cases

- ▲ For each Test Case we need to define 4 parts:
 - ◆ Initial System State:
 - system state (or part of it) before start of use case in order to deliver expected results and resulting final system state,
 - values that are derived from preconditions for use case and by inference from inputs and final system state,
 - ◆ Inputs from the actors:
 - data provided by all of the actors that should cause desired result for test case,
 - values that are derived directly from use case steps,
 - ◆ (cont'd)

Testing with Use Cases

- ◆ Final System State:
 - system state after the use case has completed,
- ◆ Expected outputs:
 - the data that will have been output as use case ran through the test case.

Use Cases
and UML

Advanced Issues in
Writing Use Cases

Relating Use Cases with
BPM & with NFRs

Lesson 7

Advanced Issues in Writing Use Cases

||

Relation of Preconditions and Use Case Failures SWEED

- ▲ A precondition is an assumption which must be true before a use case is “executed”.
- ▲ Preconditions are not checked *within* a use case. Therefore, the violation of a precondition need not be considered in the extensions clause, e.g.

Precondition: All chickens have permission to cross road.

Main Success Scenario: ...

2. User requests System to move chicken across road

3. System moves chicken across road. ...

Extensions:

... 3a. System ascertains that chicken does not have permission to cross road ... -- **INCORRECT!!!**

Use Cases: Limitations

- ▲ Does not explicitly capture domain knowledge
- ▲ Difficult to find redundant and conflicting behavior between use cases
- ▲ No rules for controlling decomposition (e.g. when use cases are decomposed into sub-use cases)
- ▲ Pushing use case decomposition too far leads to:
 - ◆ a functional decomposition design
 - ◆ design details that are best expressed in a more suitable notation
- ▲ Being a primary actor goal-oriented approach, secondary actors tend to be neglected

Use Cases: Limitations

- ▲ Are not a precise specification:
 - ◆ do not provide support for a consistent level of precision,
 - ◆ are prone to ambiguity and redundancy in their descriptions,
 - ◆ do not provide adequate means for dealing with interactions between use cases,
 - a well-known problem in telecommunication systems, called feature interaction
 - ◆ cannot express state-dependent system behavior adequately.

Formalizing Use Cases: The Fondue Approach

SWEED

- ▲ Use Cases + **Operation Schemas** offer some fixes to previous problems
 - ◆ Operation schemas are concerned with clarifying what the system offers, i.e., they expand on the system responsibilities (defined by the use cases) in a precise way.
 - ◆ The two views complement each other nicely: use cases provide the informal map of interactions between the system and actors, whereas operation schemas precisely describe a particular atomic system action, called a system operation.

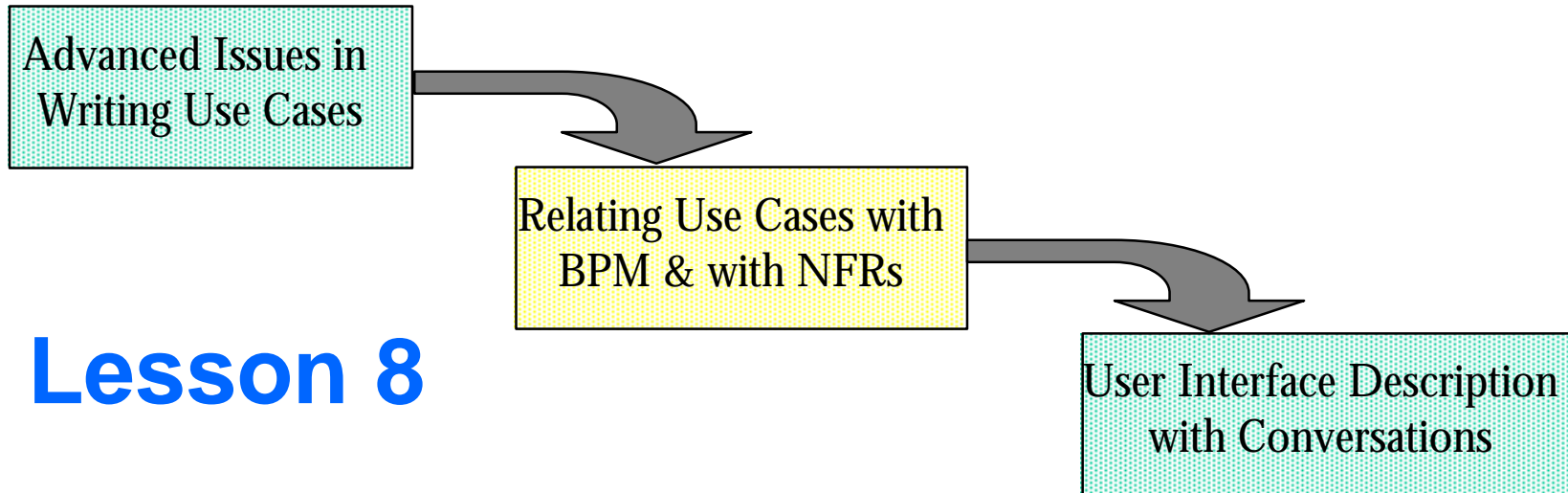
Formalizing Use Cases: The Fondue Approach

SWEED

- ▲ What do **Operation Schemas** offer?
 - ◆ Allow a more constant level of precision & clarify where one has to stop decomposition by focusing on system interface
 - ◆ Support for modeling concurrency and performance constraints
 - ◆ A more focused description for developers (less noise) — centered on system responsibilities (obligations)
 - ◆ Easier to schedule design activities because development increments relate better to system operations than to user goals

Operation Schemas

- ▲ A schema declaratively describes changes to an abstract description of system state by pre- and postconditions.
- ▲ *Precondition*: assumption about state before the execution of the operation.
- ▲ *Postcondition*: required state after operation execution + output events that were sent.
- ▲ Uses UML's Object Constraint Language, and is applied to a UML class model.
- ▲ There is a straight mapping between use cases and operation schemas. [Sendall & Strohmeier '00]



Relating Use Cases with Business
Process Modeling & with Non-
Functional Requirements

Relating Use Cases with Business Process Modeling

SWEED

- ▲ It is possible to utilize use cases to place the system under development in the context of the organization. It can be achieved by documenting the business process by enterprise scope use cases (white- and black-box use cases).
- ▲ However, the business process may need reengineering [Hammer et al. '01], in which case more specialized models should be used [IBM '96]

Relating Use Cases with Business Process Modeling

- ▲ What needs to be identified:
 - ◆ The stakeholders in the organization's behavior
 - ◆ The external primary actors whose goals you propose that the organization satisfy
 - ◆ The triggering events that the organization must respond to
 - ◆ The services the business offers, with success outcomes for the stakeholders
- ▲ This is also the bounding information for a use case

Relating Use Cases with Business Process Modeling

SWEED

- ▲ A business process either generates value for the business or alleviates costs to the business.
- ▲ Business Process Model
 - ◆ describes the business using a set of process flow diagrams
 - an ordering of activities to accomplish a business goal (activities may be manual or automated)
 - ◆ for application development, it provides a detailed understanding of the business area that will be supported or impacted by the new application — provides justification/rationale or the contrary

Relating Use Cases with Business Process Modeling

SWEED

- ▲ Connecting the BPM to Use Cases:
 - ◆ Establish the scope of the work
 - ◆ Establish the adjacent systems that surround the work
 - ◆ Identify the connection between the work and the adjacent systems
 - ◆ From the connections, identify the business events that affect the work
 - ◆ (cont'd)

Relating Use Cases with Business Process Modeling

SWEED

- ▲ Connecting the BPM to Use Cases (cont'd):
 - ◆ Study the response to the events (the work related to each business event; might be a chain of work)
 - ◆ Determine the best response that the organization can make for the event
 - ◆ Determine the system's role in the response
 - ◆ Determine the use cases for the system

Linking Business to System Use Cases

SWEED

- ▲ Two levels (and two audiences)
- ▲ First level: Business use case
 - ◆ describes business' responses to user goals; often contains no mention of technology (could be automated or manual)
 - ◆ audience: non-technical stakeholders, e.g., managers

Linking Business to System Use Cases

- ▲ Second level: System use case
 - ◆ describes primary actor's goal fulfillment but concentrates on system functionality, interested in only what is verifiable
 - ◆ audience: technical stakeholders, e.g., developers
- ▲ Questions to navigate between the 2 levels [Cockburn '01]:
 - ◆ Do the use cases form a story that unfolds from the highest- to the lowest-level goal?
 - ◆ Is there a context-setting, highest-level use case at the outermost "system" scope possible for each primary actor?

Relating Use Cases to Business Rules

- ▲ Business Rules are compiled into a catalogue that categorizes and lists them.
 - ◆ They can then be referenced by the use case (e.g. hyperlink)
- ▲ [Ross '97] suggests five categories for business rules:
 - ◆ Structural Facts
 - ◆ Action Restricting
 - ◆ Action Triggering
 - ◆ Inferences
 - ◆ Calculations

Relating Use Cases to Business Rules

- ▲ Why centralize Business Rules in a Catalogue?
 - ◆ making business rules explicit enables them to be reviewed, agreed and changed
 - ◆ enables business rules to be discussed out of context of particular applications
 - ◆ factors out and defines at a single place information which would otherwise be duplicated across many work-products

Relating Use Cases with Non-Functional Requirements

SWEED

- ▲ Non-Functional Requirements (NFRs) capture required properties or qualities of the system about how services have to be provided (rather than which ones).
- ▲ They often relate to the system as a whole rather than to a single feature.
- ▲ Failure to meet them can make the system unusable, where a missing function may just degrade the system.
- ▲ They should be identified while the functional requirements are explored.

Relating Use Cases with Non-Functional Requirements

SWEED

- ▲ NFRs are generally difficult to express in a measurable way, making them more difficult to analyze.
 - ◆ There has been some work on modeling NFRs as “soft” goals [Mylopoulos ‘92].
- ▲ NFRs often have a large effect on determining the architecture.
 - ◆ Two systems with the same use cases but very different NFRs may need very different solution architectures.

Relating Use Cases with Non-Functional Requirements

SWEED

- ▲ Make cross-references in the use case to related NFRs.
- ▲ Check:
 - ◆ Examine your NFRs to see if your use cases can address them. You might be able to refine, add or drop use cases based on this.
- ▲ Questions:
 - ◆ Are there timing, performance requirements, or other interface requirements associated with obtaining the service?

Referencing Other Documents *SWEED* from Use Cases

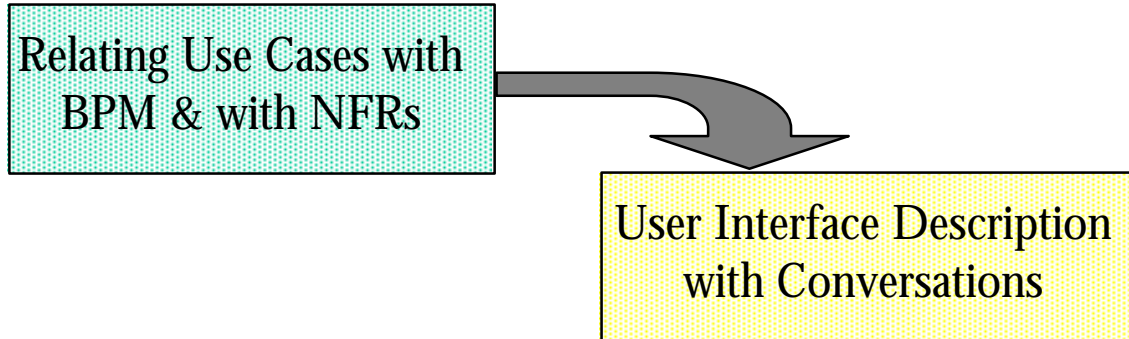
▲ Data Details and Format

- ◆ Abstract away from data details and formats in use cases (avoids inconsistencies due to redundancies and helps maintenance for the “tool challenged”)
- ◆ Store in a separate document and reference it in the use case (i.e. hyperlink it)

▲ Project Constraints

- ◆ Organizational
- ◆ Operational
- ◆ Legislative and Ethical

▲ UI guidelines and requirements



Lesson 9

User Interface Description with Conversations

User Interface Description

- ▲ To users, the user interface is the system.
- ▲ However, use cases don't describe the UI but they are close (cases of use).
- ▲ A finer-grained description of elementary user tasks that model UI interactions is needed.
- ▲ Layout and format issues must be added, e.g. by references.
- ▲ A *conversation* is a structured narrative that separates user intentions and system responses and that concentrates on interaction between users and the system.

Conversations

- ▲ Conversations are a special kind of use case that can be used to informally describe a dialog between user and system in terms of action detail.
- ▲ Conversations use a table format, which separates actor actions from system responses.
- ▲ Conversations are most commonly used to show (more) concrete behavior of a user with the system.

Example: Conversation for Order

SWEED

Actor Actions	System Responses
1. User places a new order.	
	2. System shows list of items.
3. User selects an item.	
	4. System provides pricing information for item, i.e. quantities with discounts.
5. User specifies the quantity.	
	6. System verifies that quantity is available.
	7. System prompts User to finalize order.
8. User confirms that he/she wants to finalize order.	
	9. System finalizes order.

Example: Conversation for Order (cont'd)

SWEED

Extensions: -- as in use cases

6a. System ascertains that quantity demanded by User is not available.

6a.1a. System informs the User that the item is out of stock; conversation ends in failure.

6a.1b. System informs the User about available quantity.

6a.1b.2a User agrees to this quantity; conversation continues at step 7.

6a.1b.2b User denies the offer; conversation ends in failure.

Precondition:

The customer has already been identified.

Tips for Writing Conversations

- ▲ Avoid presentation details (e.g., System displays a radio box for...),
 - ◆ instead, reference UI format, layouts, guidelines and requirements (e.g. with a hyperlink)
- ▲ Maintain a consistent level of detail.
- ▲ Don't mention objects in system responses.
- ▲ Conversations can be constructed by expanding use cases, but do NOT throw away the use cases thinking that they have now been "refined".

Extras

Lily's Top Ten Use Case Pitfalls

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem #1:

- ◆ The system boundary is undefined or inconsistent.
 - Be explicit about the scope, and label the system boundary accordingly.
 - Draw the system boundary.

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem #2:

- ◆ The use cases are written from the system's (not the actors') point of view.
 - Name the use cases from the perspective of the actor's goals.
 - Focus on what the system needs to do to satisfy the actor's goal, not how it will accomplish it.
 - Watch out when the use case model includes use cases that are not directly associated with an actor, but are associated with <<include>> or <<extend>> relationships.

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem #3:

◆ The actor names are inconsistent.

- Get agreement early in the project about the use of actor names (and other terms). Establish a glossary early in the project and use it to define the actors.
- Make sure that the granularity of the use cases is appropriate. Use cases should reflect "results of value" to the system's users -- the attainment of real user goals.

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem # 5:

- ◆ The actor-to-use case relationships resemble a spider's web.
 - The actors may be defined too broadly. Examine actors to determine whether there are more explicit actor roles, each of which would participate in a more limited set of use cases.

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem #6:

- ◆ The use case specifications are too long.
 - The granularity of the use case may be too coarse.

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem #7:

- ◆ The use case specifications are confusing.
 - Include a Context field in your use case specification template to describe the set of circumstances in which the use case is relevant. Make sure that the Context field puts each use case in perspective, with respect to the "big picture" (the next outermost scope). Don't just use it to summarize the use case.
 - (cont'd)

Lily's Top Ten Use Case Pitfalls

- Rewrite the steps to focus on a set of essential interactions between an actor and the system, resulting in the accomplishment of the actor's goal.
 - Break out conditional behavior ("If...") into separately described alternate flows.
 - Use case steps are not particularly effective for describing non-trivial algorithms, with lots of branching and looping. Use other, more effective techniques to describe complex algorithms (e.g., decision table, decision tree, or pseudo-code).
 - Make sure that the steps don't specify implementation. Focus on the external interactions. Consider expressing some of the behavior as "rules," rather than algorithms.

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem #8:

- ◆ The use case doesn't correctly describe functional entitlement.
 - Make sure that each actor associated with a use case is completely entitled to perform it. If an actor is only functionally entitled to part of the use case, the use case should be split.

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem #9:

- ◆ The customer doesn't understand the use cases.
 - Teach them just enough to understand.
 - Put a short explanation of use cases in the use case document, as a preface or appendix. The explanation should include a key to reading the model and specifications, and a simple example.
 - Lead a short training session when the use case document is distributed for review.
 - Think long and hard about using <<includes>> and <<extends>> relationships in the use case model. They are a modeling convenience, but are not at all intuitive to the inexperienced reviewer.
 - (cont'd)

Lily's Top Ten Use Case Pitfalls

SWEED

- Add information to tell the story:
 - Include a Context section in the use case template.
 - Add an overview section that provides context to a set of related use cases (e.g., a package), and use this section to "tell the story."
 - Include other kinds of models as needed. Often, a single use case will result in a state change to a major domain object, but the use case model alone won't tell the story of how the object changes state across many use cases over time. A state model (state transition diagram) of a major domain object may be an excellent way to show how several related use cases fit together over time.
- (cont'd)

Lily's Top Ten Use Case Pitfalls

SWEED

- Determine what strategy for organizing the use cases makes the most sense to the customer. Listen to how the customer describes the business.
- Watch out for computer slang that is not part of the customer's vocabulary.
- Deliver what the customer wants. This doesn't mean that use cases can't be used as a requirements elicitation technique (if they are really the right tool for the job). But they might not be a primary delivered work product.

Lily's Top Ten Use Case Pitfalls

SWEED

▲ Problem #10.

◆ The use cases are never finished.

- Don't get into user interface details.
- In the flows, focus on the essentials of what the actor does.
- Specify use case "triggering" events as preconditions (e.g., "user has selected a game, and requested to order tickets"), rather than screen navigation details. Keep the screen navigation information in a (separate) user interface design document, not in the use case model.
- (cont'd)

Lily's Top Ten Use Case Pitfalls

SWEED

- Watch out for "analysis paralysis." There is a point at which the requirements are adequately specified, and further analysis and specification does not add quality. Cover the "80%" cases; do your best on the rest within the allocated budget of time and money.
- Use cases have a simple, informal, and accessible format. Use cases are a mechanism for defining and documenting operational requirements, not magic.