
COMP 354: INTRODUCTION TO SOFTWARE ENGINEERING

Requirements Engineering

Daniel Sinnig, PhD
d_sinnig@cs.concordia.ca

Department for Computer Science
and Software Engineering

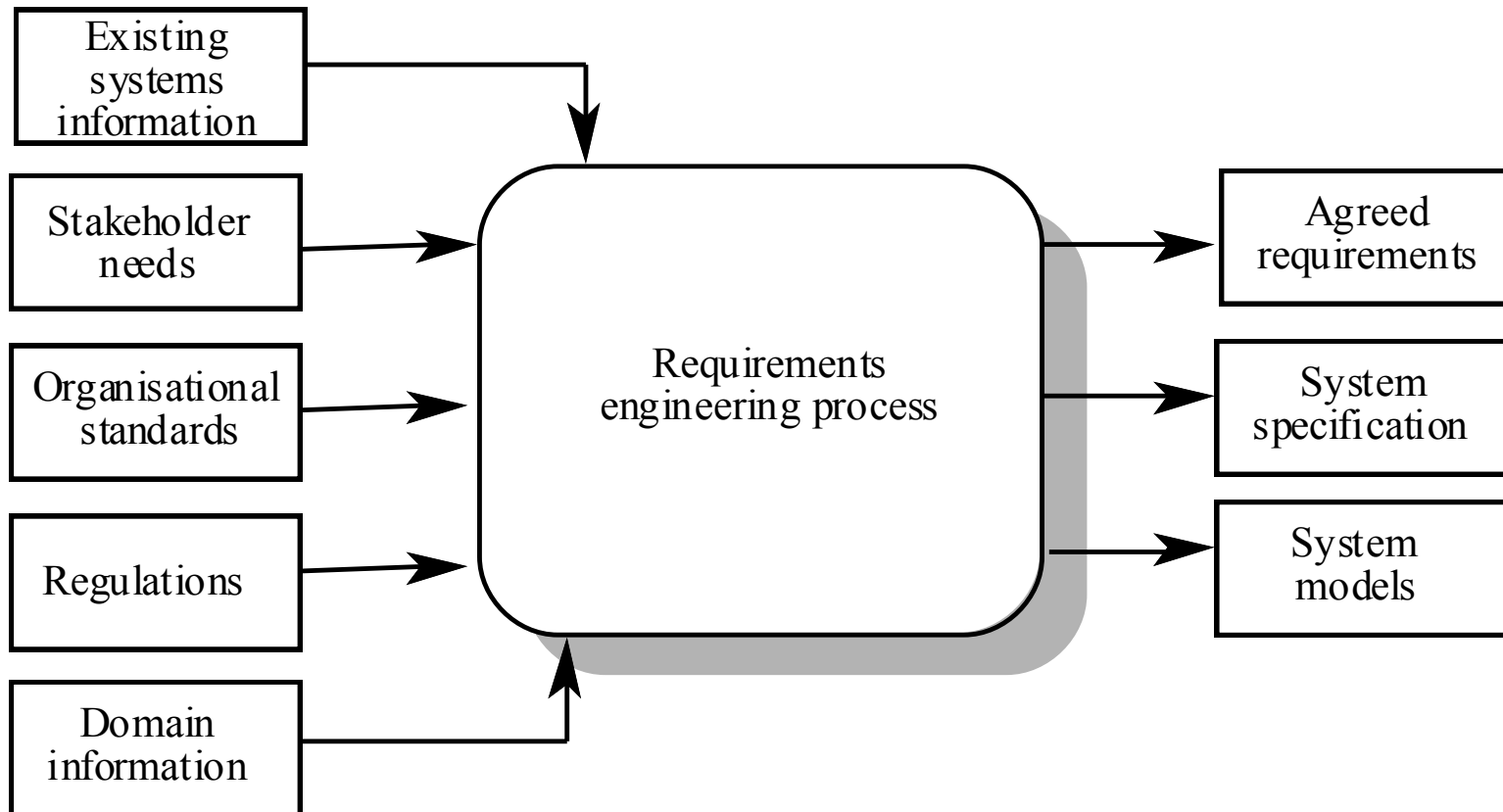
21-May-14



Requirements Engineering

“**Requirements engineering** is the **branch of software engineering** concerned with the **real-world goals** for functions of and constraints on software systems. It is also concerned with the relationship of these factors within precise specifications of software behavior, and to their evolution over time and across software families.” -- Pamela Zave

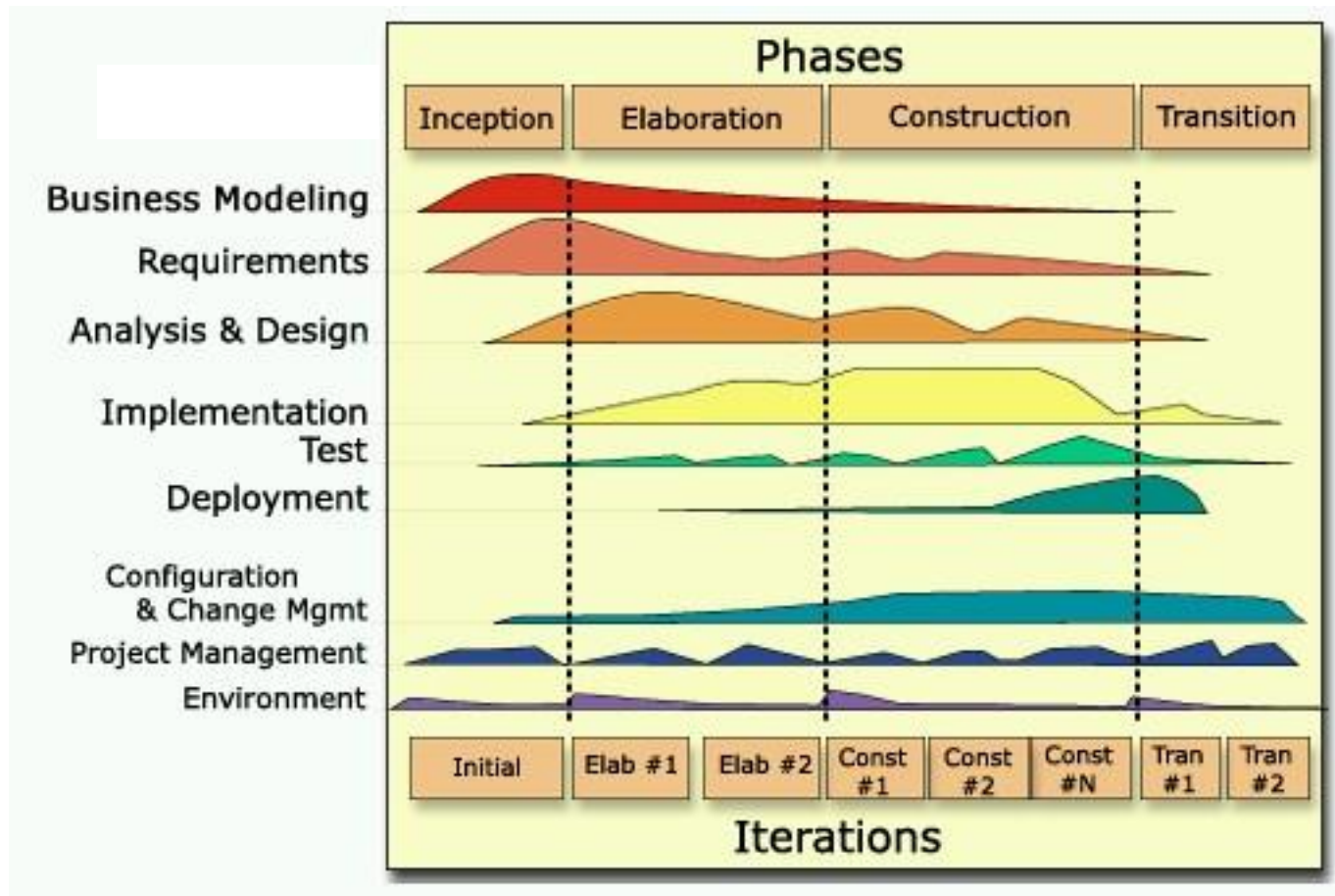
RE process - inputs and outputs



Software Development Processes: Where do ‘Requirements’ fit?

- Where do ‘requirements’ fit into an overall SE process?
(1 min)

Unified Process (UP)



How Important are Requirements?

“Done well, requirements engineering presents an opportunity to reduce costs and increase the quality of software systems. Done poorly, it could lead to a software project failure.” -- Software Engineering Institute (SEI)

How Important are Requirements?

“The hardest single part of building a software system is deciding what to build. No other part of the conceptual work is so difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.” -- Frederick P. Brooks (1975)

Done well....

- Communicate and clarify the real needs of customers
 - Ability to accommodate requirements changes
 - Definition of a repeatable engineering process
 - Development of quality software that is bound to the user's needs, on time and to budget
- ➔ Customer satisfaction

Done Poorly...

- Cost and time explosion
- Requirements errors are extremely costly (budget + time) to fix
 - Solution for the wrong problem
 - Errors likely to be found during user testing
 - ➔ Design and code re-work

Building the Right System

vs.

Building the System Right

Requirements vs. Design

- What *vs.* How
- Building the right system *vs.* Building the system right
- It is not the nature of the statement itself that makes it a requirement or not. It (always) depends on the context
- It is not the level of detail
- Requirements or Design?
 - “If the alarm system is ringing, then the elevators (lifts) will proceed to the ground floor, open their doors and suspend further operations.”
 - “The project will be implemented in Java.”
 - “The system will use an array to hold the invoices.”

Requirements Management: Of Course!

- “It’s difficult for anyone to argue rationally against the idea of managing and documenting the requirements of a system in order to ensure that we deliver what the customer really wanted. However ...” [Leffingwell and Widrig, 1999, p.28]

Requirements Management: Reality?

- “... data demonstrates that, as an industry, we often do a poor job ...”

Requirements Management: Reality?

Mis-management due to

- Lack of stakeholder (esp. user) involvement =>
 - Incomplete req. (lack of input)
 - Inaccurate req. (lack of consultation/feedback)
- Change in requirements & specifications and ridged development processes

Requirements Management: Reality?

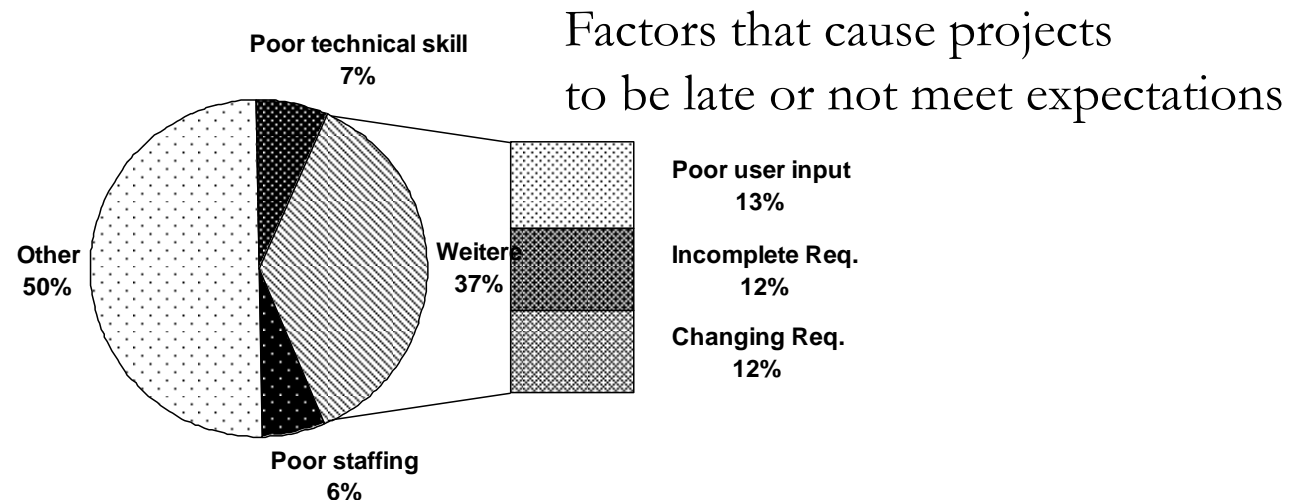
- Common attitude:

“even if we’re not really sure of the details of what we want, it’s better to get started with implementation now, because we’re behind schedule and in a hurry. We can pin down requirements later.”

- ➔ Often leads to **chaos**: no one quite knows what user wanted or system should do!

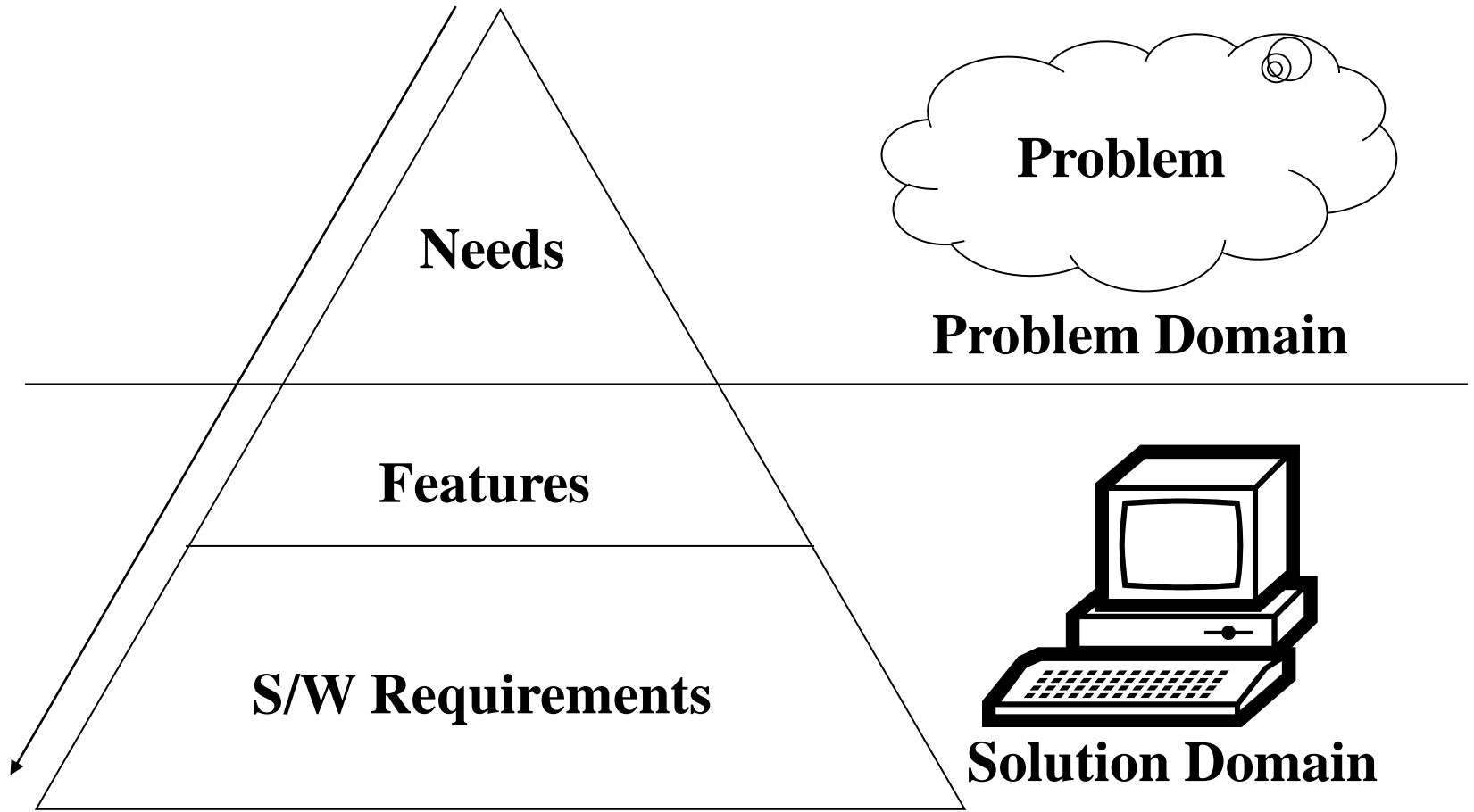
Observations

- 52.7% of projects will cost 189% of the original estimates



- Most significant factor for (partial or total) failure is related to *mis-management of requirements.*

Road Map to Software Requirements



Stakeholder Needs

- Key High-level goals and problems of stakeholder [Larman]
- Reflection of business, personal, or operational problem (or opportunities)
- Must be addressed to justify project
- Expressed independent from concrete solution
- Examples:
 - Students need less overhead for course registration
 - Professors need immediate access to student grades

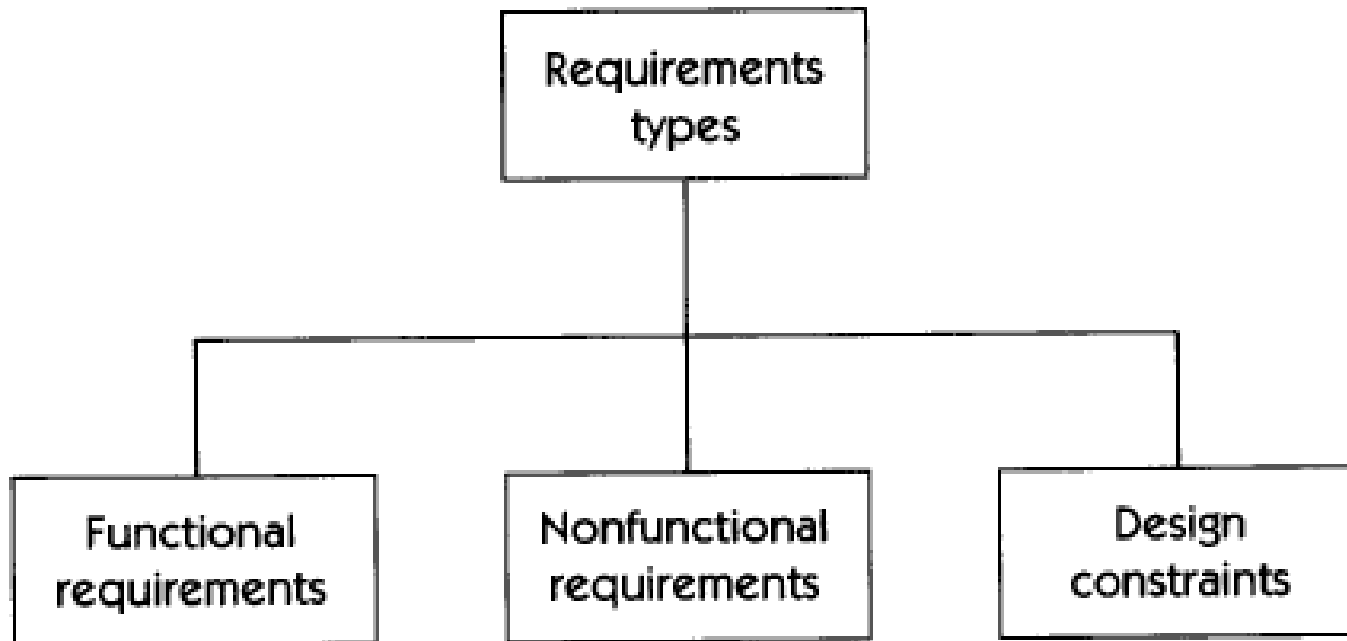
System Feature

- Externally observable service or attribute provided by the system
- Directly fulfills a stakeholder need
- Should pass linguistic test [Larman, 2002]:
 - The system shall do *[Feature]*
- Examples:
 - System shall do payment authorization
 - System shall ensure that courses are not overbooked

Software Requirements

- A software capability needed by the user to solve a problem or to achieve an objective
- or
- A software capability that must be possessed by the system to satisfy a contract, a standard or a specification

Requirements Types



(most general classification)

FURPS+ Classification [Larman, 2002]

- **F**unctional
 - Functionality or services that the system is expected to provide
 - Addresses the *input-output behavior* of a system
- **U**sability
 - Human factors, help, documentation, etc.
- **R**eliability
 - Frequency of failure, recoverability, predictability, etc.
- **P**erformance
 - Response time, availability, resource usage, etc.
- **S**upportability
 - Maintainability, adaptability, configurability, etc.

FURPS+ Classification Cont.

+ indicates additional quality attributes such as:

- Interfaces
 - i.e. user interface, component interfaces
- Implementation Constraints
 - Resource limitations, languages and tools, hardware, etc.
- Etc.

Requirements Engineering Steps

1. **Analyzing the problem (and its root causes)**
2. Requirements elicitation (understanding user and stakeholder needs)
3. Requirements specification (defining the system)

Why Do Problem Analysis?

- As engineers we feel compelled to solve a problem as soon as we catch a glimpse of one.
- I.e. often our focus is on applying a solution, even before understanding the real problem.
- But, you cannot effectively solve a problem unless you know what *it* is!
- In fact, there might not even be a problem!

Product Always Solve a Problem?

- Not every product solves a problem.
- Some target market opportunities.
- Often we can describe an opportunity in terms of a problem (and vice versa).
- We will use the “problem” point-of-view.

Analyzing the Problem – the Five Steps

1. Gain agreement on the problem definition.
2. Understand the root causes—the problem behind the problem.
3. Identify the stakeholders, especially users.
4. Define the solution system boundary.
5. Identify the constraints to be imposed on the solution.

Steps in Problem Analysis: Order, ... ?

- Order of the steps is not crucial.
- Doing all of the steps is important.
- Iterate among steps.

Step 1: Agreement on Problem Definition

- Write it down!
- Make sure everyone(?) agrees.

- Do not try to get the perfect problem definition.
- Aim to capture the essence of the problem.
- Definition may be revised afterwards.

Problem Definition “Template”

The problem of

(describe the problem)

affects

(the stakeholders affected by the problem).

The impact of which is

(what is the impact of the problem on stakeholders).

Benefits of a solution ...

(list some key benefits of a successful solution).

Step 2: Understand Root Causes

- Do not mistake a symptom for the problem.
- Distinguishing between real problem and symptoms.
- Domain experts can contribute best.
- What happens if one “solves” the symptom?
- E.g. headache: problem or symptom.

Root Causes

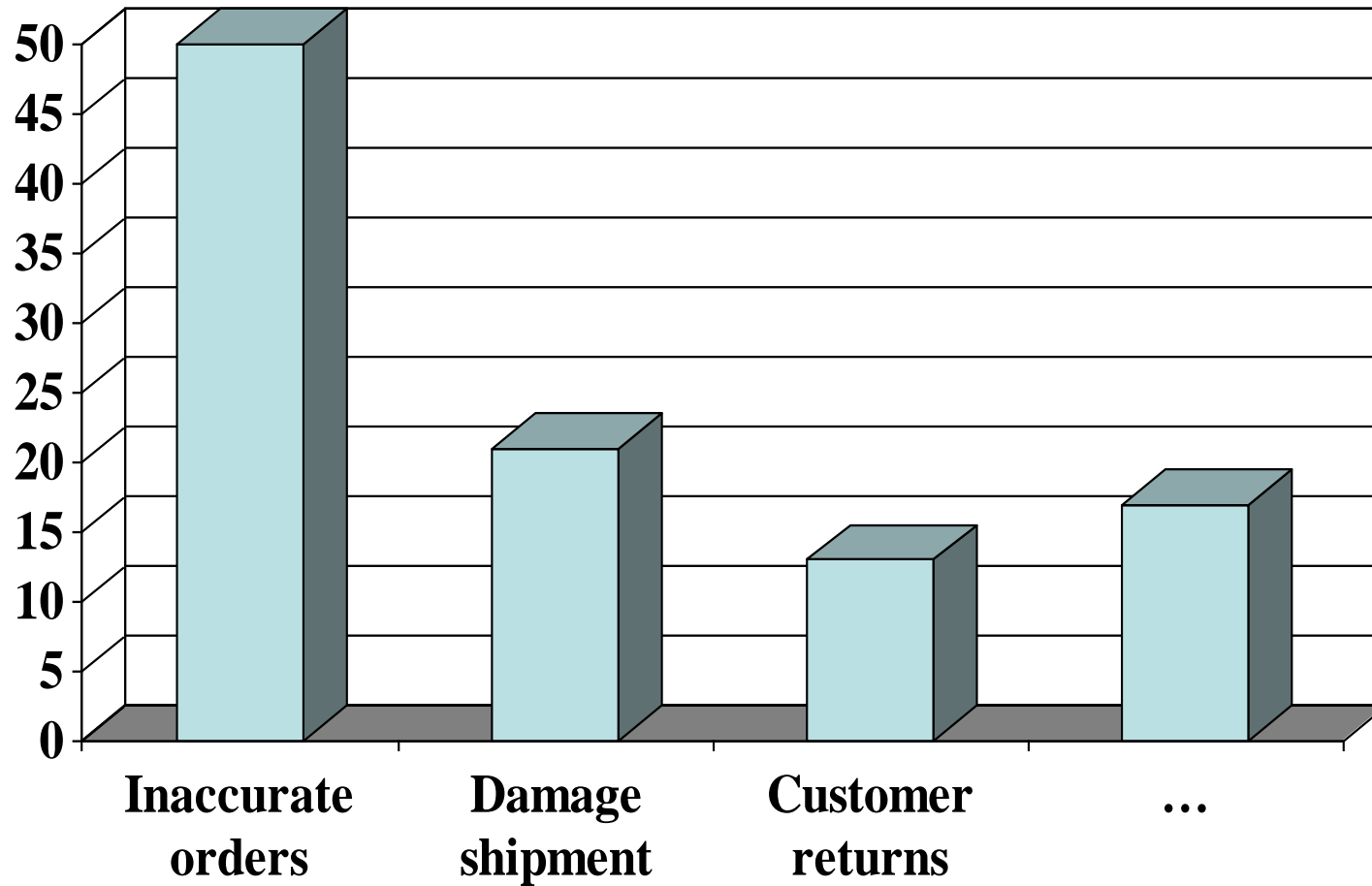
- Generally multiple causes.
- We cannot solve all causes ... (often difficult to resist).
- Choose most significant factor(s).

Sample Problem: GoodsAreUs

- “... a mail-order catalog company ... sells a variety of inexpensive, miscellaneous items for home and personal use.”
- **Problem: insufficient profitability.**

GoodsAreUs Root Causes

- Inaccurate sales orders
- Damaged in shipment.
- Customer returns.
- Finished-goods obsolescence.
- Manufacturing defects.
- Other.



Problem Definition: GoodAreUs

The problem

of inaccuracies in sales orders

affects

sales order personnel, customers, manufacturing, shipping, and customer service.

the impact of which is

increased scrap, excessive handling costs, customer dissatisfaction, and decrease in profitability.

Problem Definition: GoodAreUs

Benefits of a new system

to address the problem include

- Increased accuracy of sales orders at point of entry.
- Improved reporting of sales data to management.
- And, ultimately, higher profitability.

Step 3: Identify stakeholders and users

- Stakeholder: anyone who could be affected by the new system or has input to provide in the development of the new system.
- Different stakeholders that have different viewpoints on the problem.
 - Users.
 - Managers.
 - IT people.
 - External regulators.
 - System developers.

Step 4: Define System Boundary

- World partitioned (i.e. no sharing) in two
 - Our system
 - Things that interact with our system (its environment).

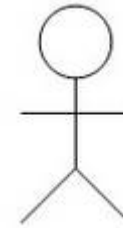
- How to determine if something is
 - Within the system solution boundary?
 - Outside of system boundary (e.g. an actor)?

Step 4: Define System Boundary

- It is *inside* your solution boundary if the development organization either creates or can modify the component.

Actors

- “Things that interact” with the system are called *Actors*.
- Actors are *external* to the system.
- People or other systems can be Actors.



Step 5: Identify Constraints on System

- Constraint: a restriction on the degree of freedom we have in providing a solution.
- As important as requirements:
 - what the system should not do, or what the system should not be.

Potential Sources of Constraints [Leffingwell & Widrig, 2002]

Economics

- What financial or budgetary constraints apply?
- Are there costs of goods sold or any product pricing considerations?
- Are there any licensing issues?

Politics

- Do internal or external political issues affect potential solutions?
- Are there any interdepartmental problems or issues?

Technology

- Are we restricted in our choice of technologies?
- Are we constrained to work within existing platforms or technologies?
- Are we prohibited from using any new technologies?
- Are we expected to use any purchased software packages?

Systems

- Is the solution to be built on our existing systems?
- Must we maintain compatibility with existing solutions?
- What operating systems and environments must be supported?

Environment

- Are there environmental or regulatory constraints?
- Are there legal constraints?
- What are the security requirements?
- What other standards might restrict us?

Schedule and resources

- Is the schedule defined?
- Are we restricted to existing resources?
- Can we use outside labor?

Problem Analysis



**Problem
Domain**

1. Gain agreement on the problem definition.
2. Understand the root causes—the problem behind the problem.
3. Identify the stakeholders, especially users.

4. Define the solution system boundary.
5. Identify the constraints to be imposed on the solution.



**Solution
Domain**

Requirements Engineering Steps

1. Analyzing the problem (and its root causes)
2. **Requirements elicitation (understanding user and stakeholder needs)**
3. Requirements specification (defining the system)

Understanding user and stakeholder needs.

- “In doing so, we’ll also start to gain an understanding of the potential requirements for a system that we will develop to address these needs.”

also called **Requirements Elicitation**.

Requirements Elicitation ... **Do:**

- Elicit and/or propose.
- Document.

- **Confirm!**

Repeat.

Requirements Elicitation

- More challenging than you might expect!
- Characterized by “syndromes”
 - **“Yes but ...”** is human nature. Stakeholders have difficulty envisioning what they want.
 - Search for req. is like search for **“undiscovered ruins”** ... always some remain.
 - **“User and the developer”** are from different worlds; language, jargon, background, motivation, ... differ.

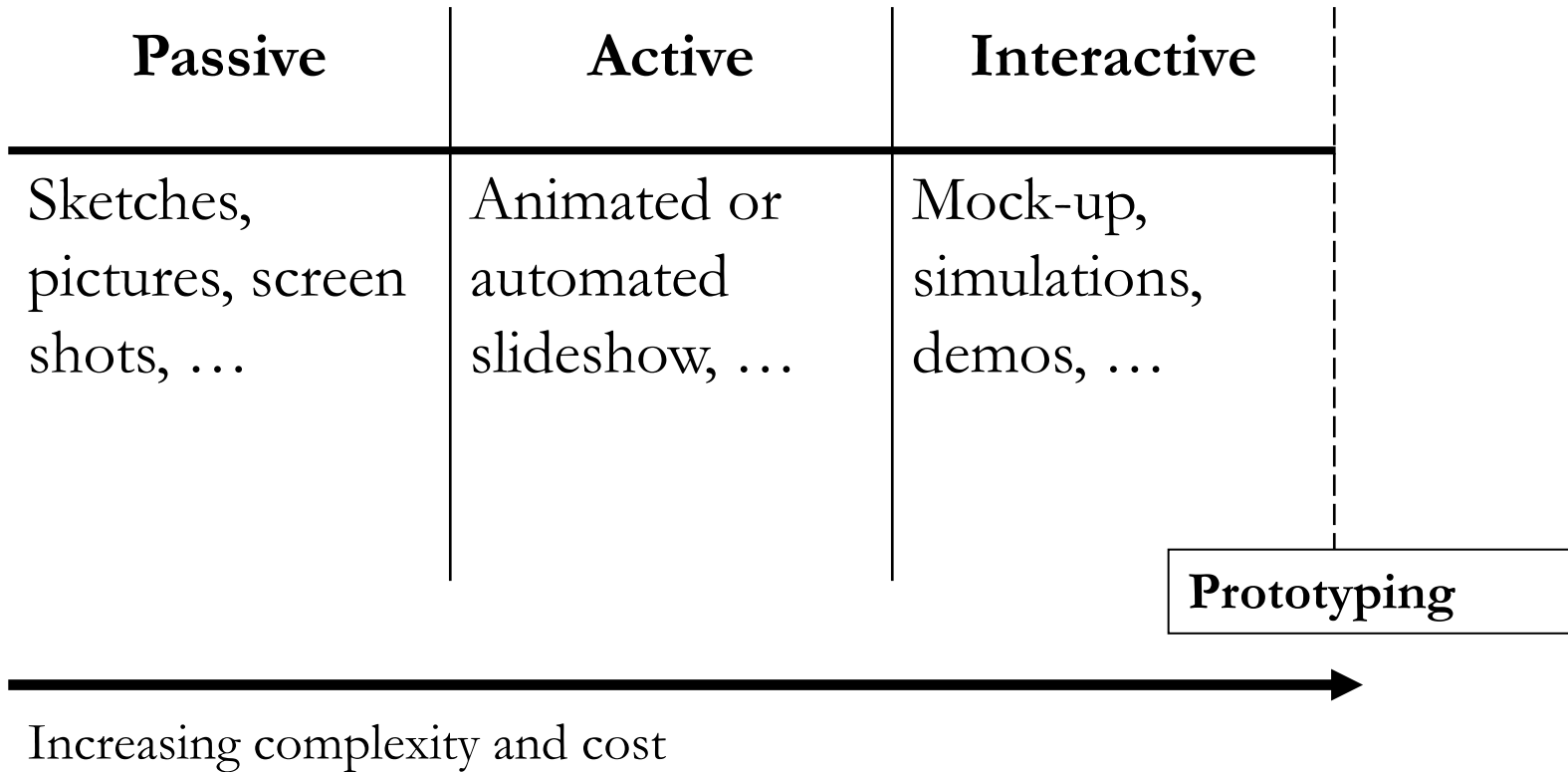
Requirements Elicitation Techniques

- **Storyboards**
- Interviews & questionnaires.
- Requirements workshops.
- Brainstorming sessions and idea reduction.
- Contextual Inquiry

Storyboarding

- Like for movies.
- Help elicit early “yes, but” reactions
- Kinds:
 - **Passive**: tell a story using sketches, pictures, screenshots, sample output.
 - **Active**: Show an animated view of system behavior.
 - **Interactive**: have the users interact with a throw-away prototype.

Storyboarding



Storyboarding Tips

- *Create them early and iteratively improve them (often).*
- Make storyboards easy to change ... by stakeholders.
- Don't invest too much in creating storyboard.
- Don't make the storyboard too good.
 - This is not meant to be UI design.
 - Waste of time/effort on your part.
 - Stakeholders might mistake for the real thing ...

Interviews

- Session of questioning of a person (stakeholder) during which information (requirements) is elicited
- Can be used in most circumstances for
 - Understanding the nature of the problem
 - Exploring potential solutions
- Different types:
 - Structured, **semi-structured**, unstructured, conversational

Interviewing Tips

- **Preparation**
 - Questions, problem statements, storyboards, use cases etc.
- **Avoid bias**
 - Be careful not to bias the interviewees answers by prematurely providing possible solutions
 - **Context-free questions**
 - About the nature of the user’s problem without context for a potential solution
 - E.g. Who is the user? Who is the customer? Are their needs different?
 - **Solution-context questions**
 - Purpose: Exploration of possible solutions
 - E.g. What if you could...? How would you rank the importance of...?
- Take notes
- Avoid technical jargon
- Follow-up

[Generic Interview Script]

Requirements Workshop

- The requirements workshop is perhaps the most powerful technique for eliciting requirements.
- It gathers all *key* stakeholders together for a short but intensely focused period.
- The use of an outside facilitator experienced in requirements management can ensure the success of the workshop.
- Brainstorming is the most important part of the workshop.

Preparing for the workshop

- Selling the workshop *concept* to stakeholders
- Ensuring the Participation of the Right Stakeholders
- Logistics
 - Includes travel, lighting, and even “afternoon sugar filled snacks.”
- Warm-up materials
 - Project-specific information

Role of the Facilitator

- Establish professional and objective tone to the meeting.
- Start and stop the meeting on time.
- Establish and enforce the “rules” for the meeting.
- Introduce the goals and agenda for the meeting.
- Manage the meeting and keep the team “on track.”
- Facilitate a process of decision and consensus making, but avoid participating in the content.
- Make certain that *all* stakeholders participate and have their input heard.
- Control disruptive or unproductive behavior.

Workshop Agenda

- Set an agenda before the workshop and publish it along with the other pre-workshop documentation.
- Balance is the key, try to stay on the agenda, but do not strictly obey it, especially if good discussion is going on.
- Order lunch in, and have a *light* working lunch. :-)


Running the Workshop

- Allow for human behavior, and have fun with it.
 - Do not “attack” other members.
 - Do not get on a soap box.
 - Do not come back late from a break.
- Workshop tickets
 - Give every stakeholder 3 workshop tickets
 - 1 for being late
 - 1 for “cheap shot”
 - 1 for “soap box”
 - Facilitator takes tickets when appropriate. If you do not have a ticket create a fund to add to, like \$1 to pot for after workshop activities.

Contextual Inquiry

- Combination of an (user) interview and observation session
- Goal is to better understand users, their needs, and work processes, and what supports or hinders them.
- Basic Technique:
 - Observation of how the user performs a given task at the user's workplace
 - While the user performs the task
 - Ask questions
 - Take notes
 - Time, Video tape etc..

Contextual Inquiry Tips

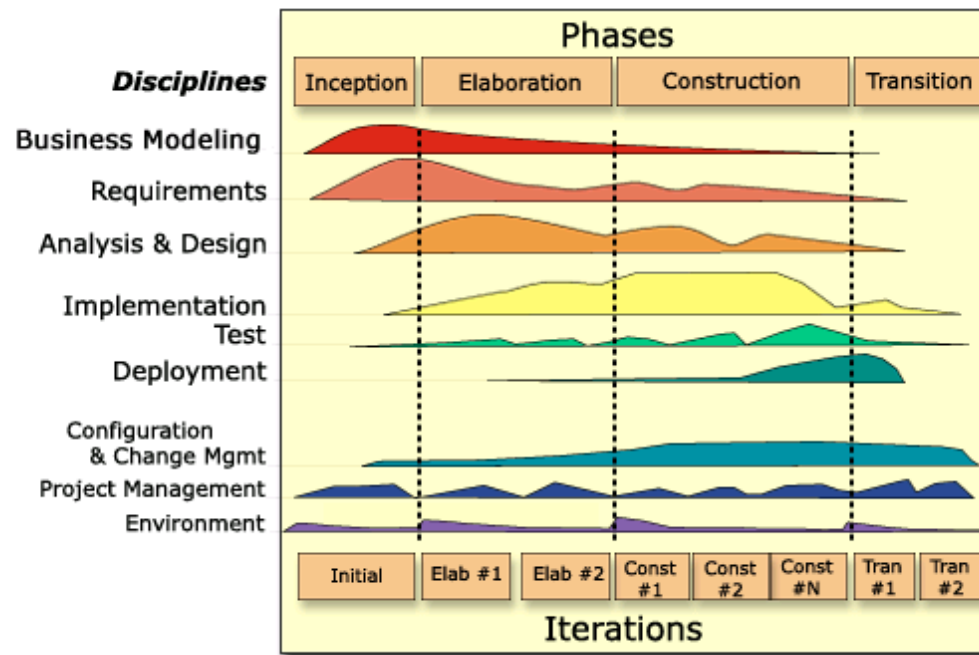
- Follow predefined role model
 - ~~Expert – Novice~~
 - ~~Interviewer – Interviewee~~
 - ~~Guest – Host~~
 - Apprentice – Master 
- Master / Apprentice Model
 - User is master craftsman who explains his work to the apprentice (interviewer)
 - Avoid other relationship models
- Avoid bias:
 - Say what users should do, but not how they should do it

Requirements Engineering Steps

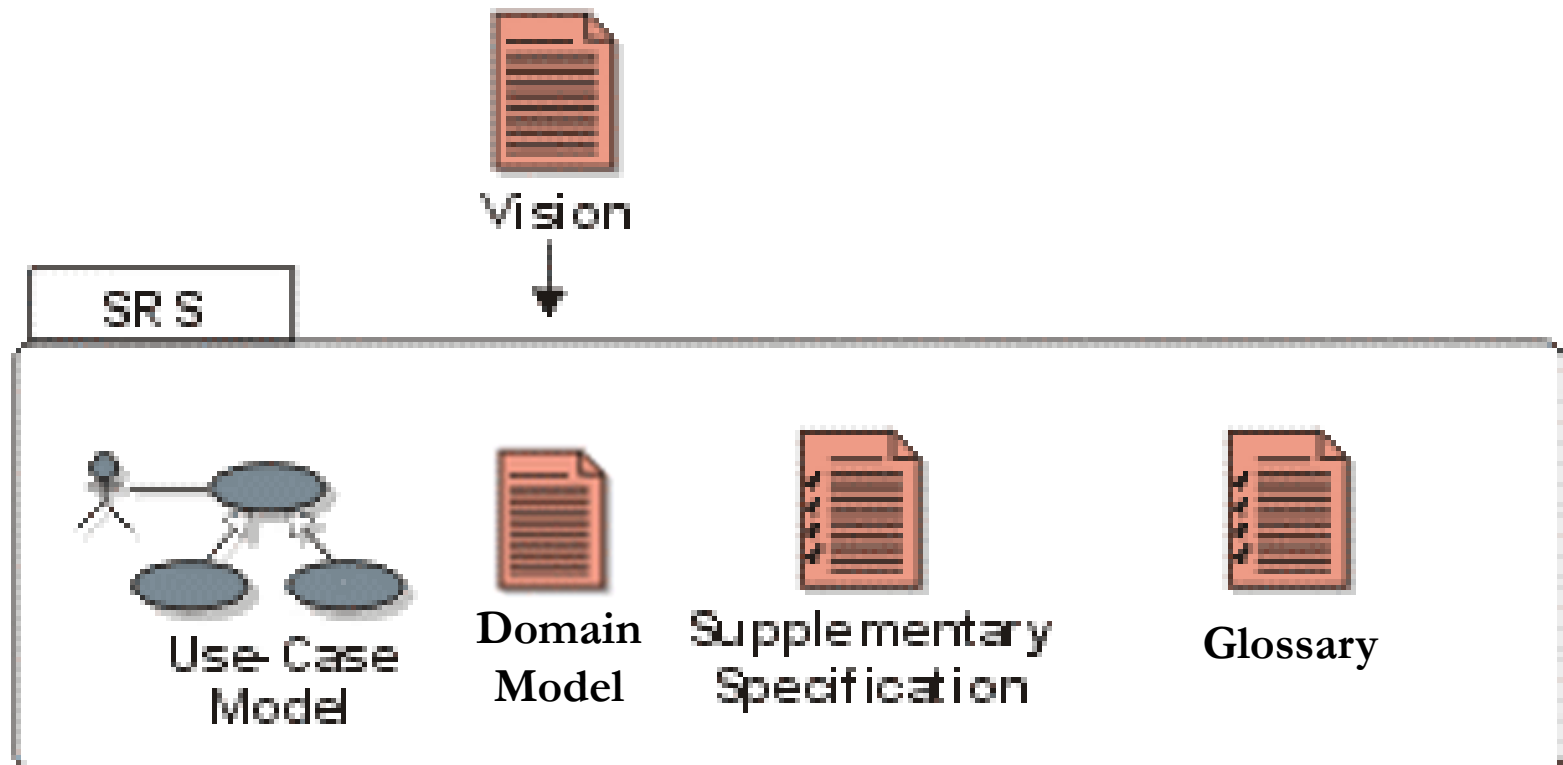
1. Analyzing the problem (and its root causes)
2. Requirements elicitation (understanding user and stakeholder needs)
3. **Requirements specification (defining the system)**

Requirements Specification

- Most important is not how you package your requirements information, but that you document them
- But we need to use some packaging, we will follow RUP.



Main UP Requirements Artifacts



Vision Document

- Captures:
 - Problem statement
 - Competitor analysis
 - Stakeholder description
 - High-level business requirements
 - User needs and (prioritized) features
 - In / Out list
 - (Domain Model)
 - UML Class Diagram capturing main concepts of problem domain

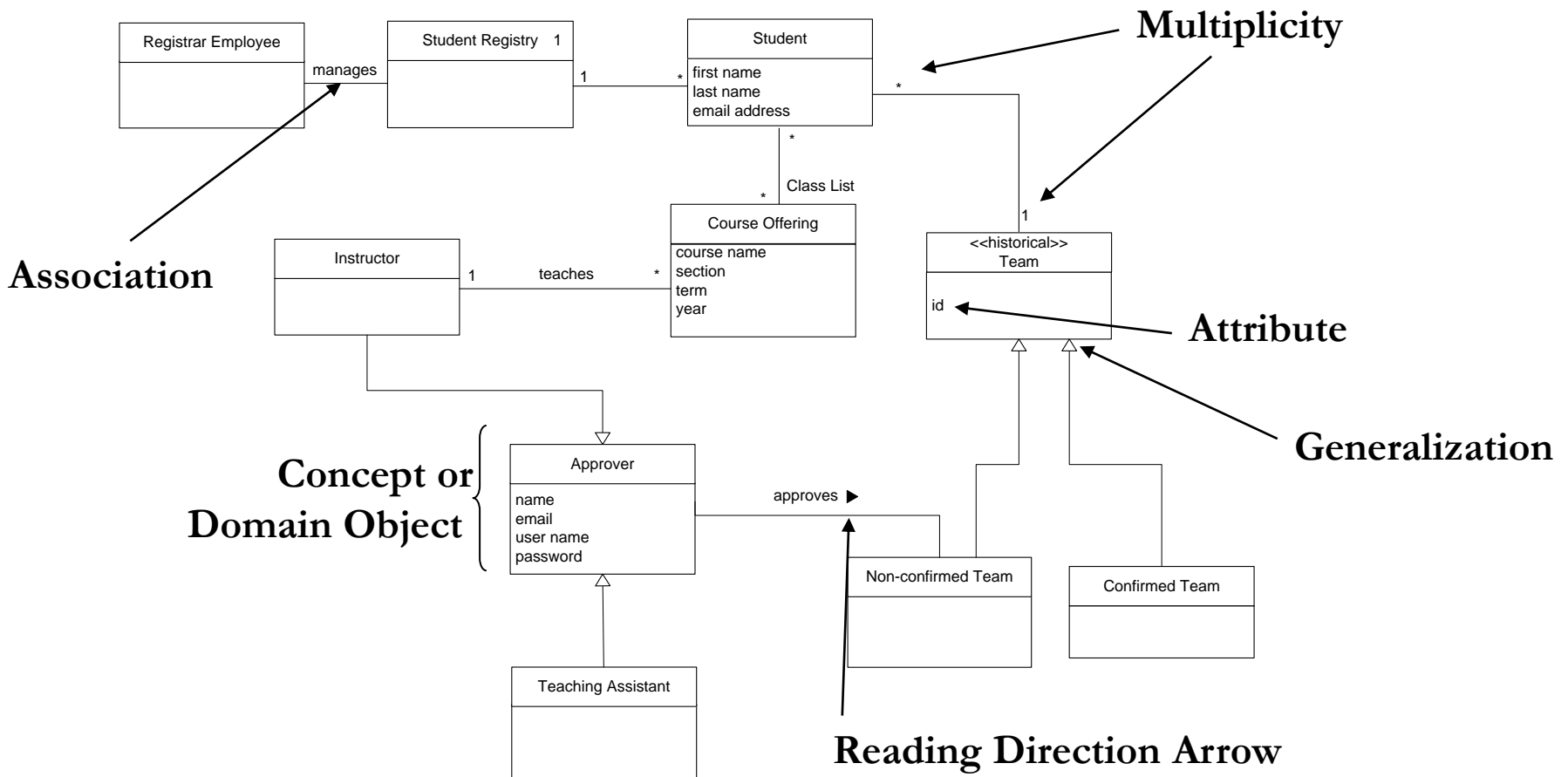
Software Requirements Specification

- Capture all S/W system requirements
- Consists of:
 - **Domain Model**
 - UML Class Diagram capturing main concepts of problem domain
 - **Use Case Model**
 - Captures functional requirements in terms of
 - Set of (textual) use cases
 - Use case diagram (optional)
 - System sequence diagram (optional)
 - **Supplementary Specification**
 - Captures non-functional requirements and design constraints
 - Business Rules
 - Additional information (references, licensing, legal requirements)
 - **Glossary**
 - Definitions, Acronyms and Abbreviations

The Domain Model

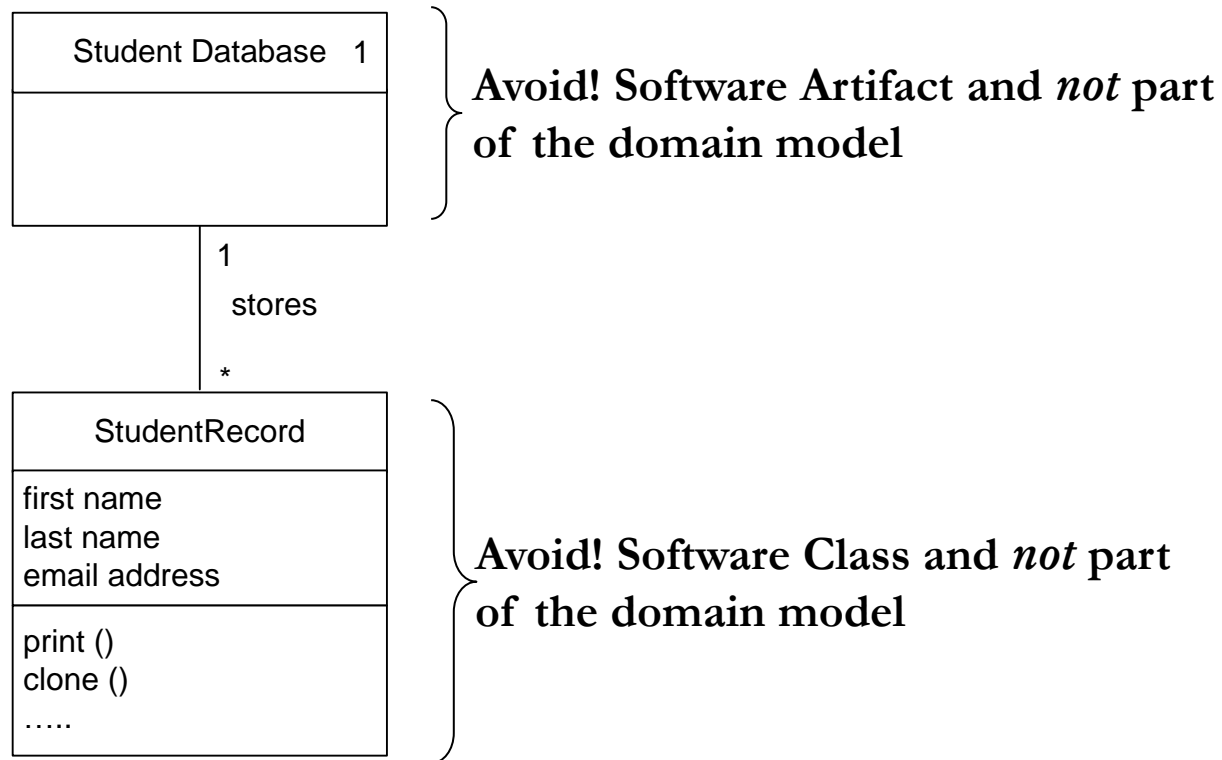
- “A **domain model** is a visual dictionary of noteworthy abstractions, domain vocabulary ... of the domain”. [Larman, p.128-9]
- Representation of the **problem domain** in terms of
 - Real-world *conceptual classes*
 - *Not* of software components
- Typically represented by UML Class Diagrams
- *Let's imagine the problem of Course Management. Which domain concepts exist?*

Example Domain Model: Course Management



Domain Modeling Principles

- Model the Problem Domain and NOT the (envisioned) Software



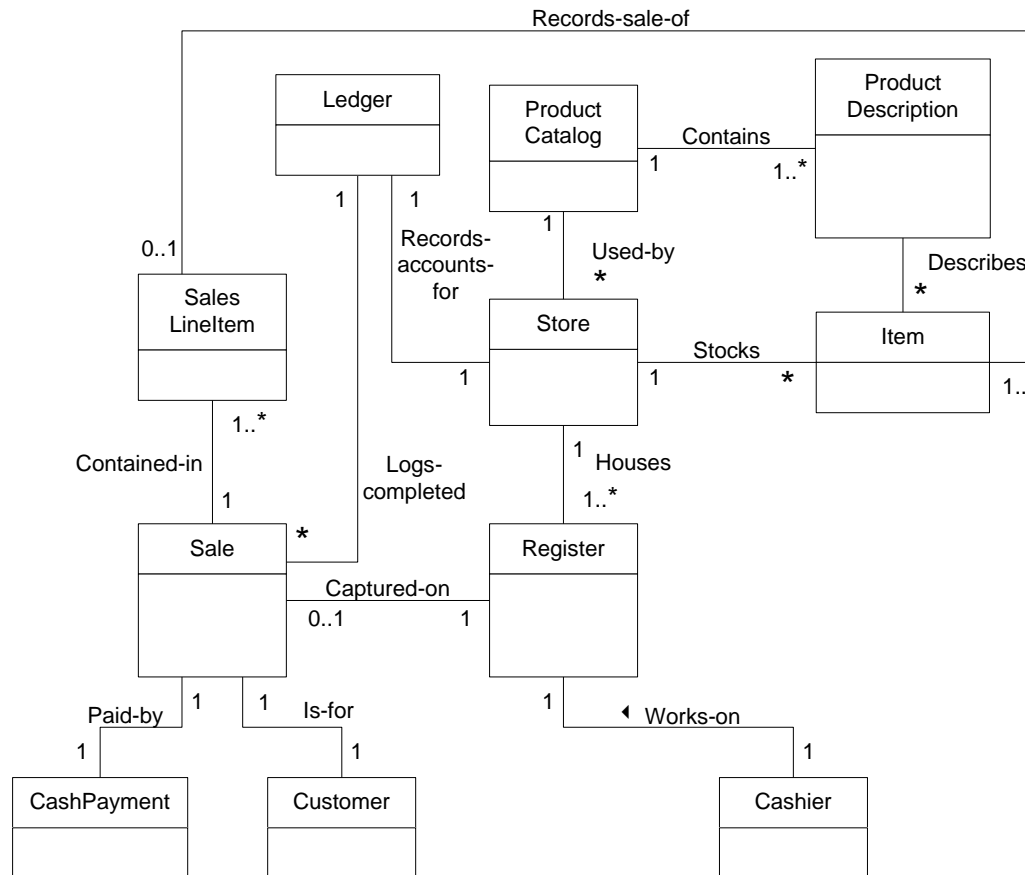
Domain Modeling Principles (cont.)

- Domain model is the result of refinement through **several iterations** of
 1. Find conceptual classes
 2. Create corresponding class in UML class diagram
 3. Add associations and attributes
- Mapmaker Metaphor (Larman, 2005)
 - Use existing names in the territory (domain)
 - Exclude irrelevant features
 - Do not add things that are not there

Domain Modeling Principles (cont.)

- Guidelines:
 - **Avoid:** Methods, data types, foreign keys!
 - Attribute vs. Association
 - Attribute: If you think of it as a number or text
 - Association: otherwise

Example Domain Model: Point of Sale System



The Use Case Model

- Medium of choice for capturing **functional requirements**
- Grouped into **use case packages**
- Defines several use cases, which capture interaction between actors and system under development as **scenarios**
- Use cases are organized into three parts:
 - Properties
 - Main Success Scenario (Normal Flow)
 - Extensions (Alternative Flow)

History

- 1986: Ivar Jacobson coined the term **use case** (result of his PhD thesis “Concepts for Modeling Large Real Time Systems”)
- 1992: Ivar Jacobson defined the **Use Case Driven Approach** in his book “Object-oriented Software Engineering: A Use-Case Driven Approach”
- 2003: Use cases have been adopted by the Rational Unified Process
- Have become a key activity in mainstream software development

Common Definitions of Use Cases

“A use case as a specific way of using the system by using some part of the functionality.”
(Ivar Jacobson)

“A use case is a collection of possible sequences of interactions between the system under discussion and its Users (or Actors), relating to a particular goal. “ (Alistair Cockburn)

“A use case represents a series of interactions between an outside entity and the system, which ends by providing a business value.” (Kulak and Guiney)

“Use cases represent the things of value that the system performs for its actors. [...] Use cases have a name and [...] detailed descriptions, about how the actors use the system to do something they consider important.” (Bittner and Spence)

Main Success Scenario (Normal Flow)

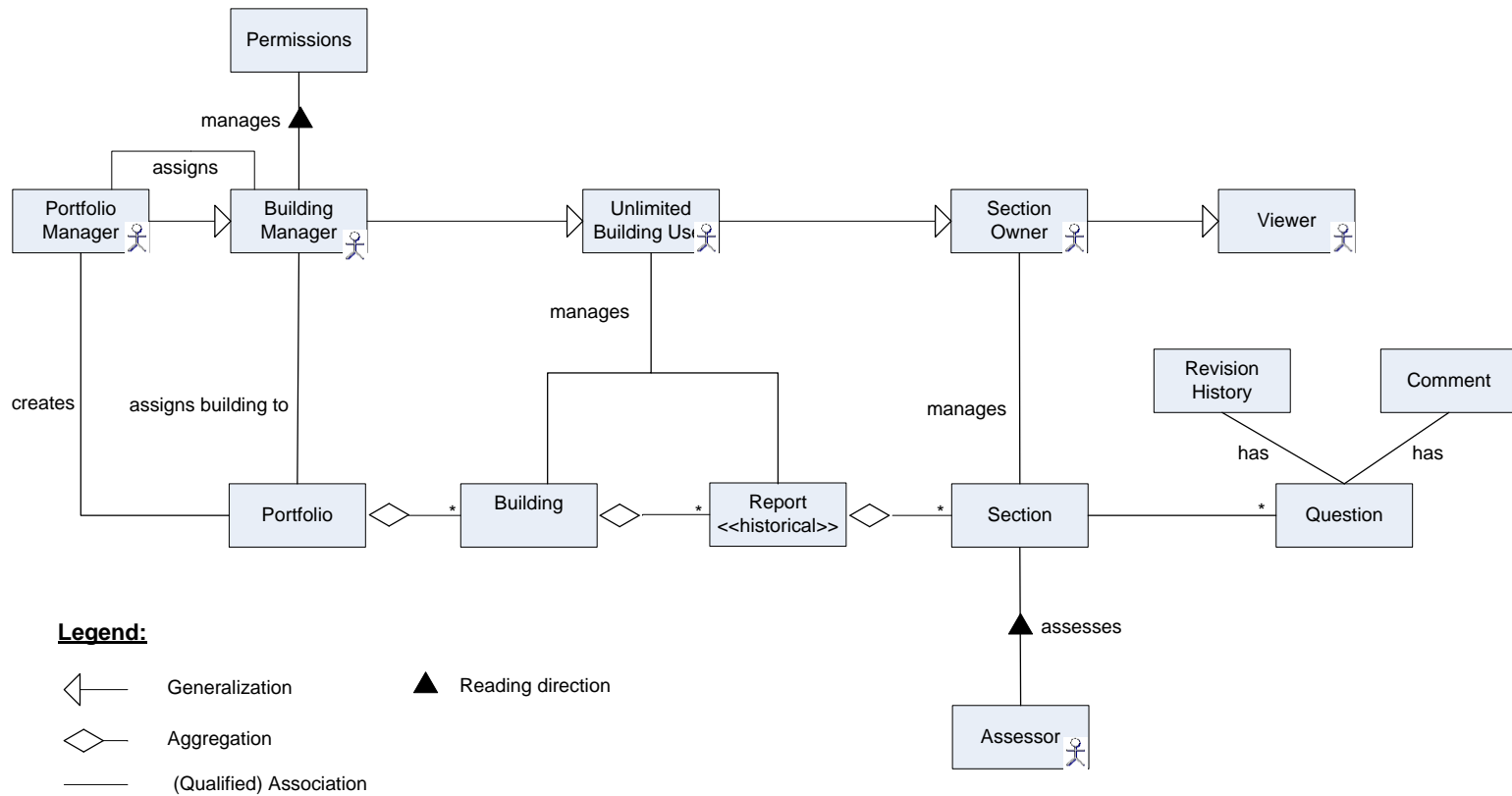
- Denotes most common way that primary actor achieves use case goal
- First step typically initiated by primary actor
- Last step typically performed by the system
 - leads to the fulfillment of use case goal

Extensions (Alternative Flows)

- Specify alternative scenarios which **may** or **may not** lead to the fulfillment of the use case goal
- Each extension starts with a condition and a reference to a use case step
- **Exhaustive modeling** of Extensions is indispensable to capturing full system behavior

“Bugs lurk in corners and congregate at boundaries.” — Boris Beizer

Domain: Property Management



Example Use Case: Delete Portfolio

Use Case: Delete Portfolio

The portfolio owner deletes a selected Portfolio.

Use Case Properties:

| | |
|------------------|--|
| Use Case Package | Portfolio Management |
| ID | UC-PFM-002 |
| Use Case Goal | Primary actor successfully deletes a selected <u>Portfolio</u> |
| Actor(s) | Primary Actor: <u>GBI User</u> |
| Level | User-goal |
| Precondition | <u>GBI User</u> is the portfolio owner. |
| Domain Entities | <u>Portfolio</u> , <u>GBI User Account</u> |

Example Use Case: Delete Portfolio (cont.)

Normal Flow:

| Step | Action | Notes |
|------|--|---|
| 1 | Primary Actor indicates to delete a selected <u>Portfolio</u> . | |
| 2 | System verifies that the <u>Portfolio</u> is empty, that there are no Portfolio Administrators associated with the <u>Portfolio</u> and then prompts Primary Actor for confirmation. | Ken, Please confirm that the Portfolio should be empty and that no Administrators are associated with it. |
| 3 | Primary actor confirms. | |
| 4 | System removes the selected <u>Portfolio</u> , reimburses (BR4) the Primary Actor and confirms the deletion of the <u>Portfolio</u> (UM9). | |
| 5 | <i>Use case ends successfully</i> | |

Example Use Case: Delete Portfolio (cont.)

Alternative Flows:

2a. The Portfolio is not empty

| Step | Action | Notes |
|------|--|-------|
| 2a.1 | System informs Primary Actor that the <u>Portfolio</u> cannot be deleted because it is not empty (<u>UM10</u>) | |
| 2a.2 | <i>Use Case ends unsuccessfully.</i> | |

2b. There are currently Portfolio Administrators associated with the Portfolio

| Step | Action | Notes |
|------|---|-------|
| 2b.1 | The system informs the Primary Actor that the <u>Portfolio</u> cannot be deleted because there are currently Portfolio Administrators associated with the <u>Portfolio</u> (<u>UM11</u>). | |
| 2b.2 | <i>Use Case ends unsuccessfully.</i> | |

Example Glossary of Domain Concepts

Business Entities

| Business Entity | Description |
|-------------------------|--|
| <u>Portfolio</u> | Grouping mechanism for buildings. |
| <u>Portfolio Credit</u> | A sufficient number of Portfolio Credits is required to create a <u>Portfolio</u> . |
| <u>Portfolio Report</u> | Statistical overview of the <u>Buildings</u> in the <u>Portfolio</u> |
| <u>Building</u> | Primary business object of the application. Main purpose of application is to have building(s) certified by an assessor. |
| <u>GBI User Account</u> | Each registered user has a GBI User Account. |

Example Business Rules

| Number | Label | Business Rule | Notes |
|------------|--|--|--|
| <u>BR1</u> | Portfolio Credits required to Create a Portfolio | In order create a <u>Portfolio</u> , at least 1 paid, unused <u>Portfolio Credit</u> is required. Once the <u>Portfolio</u> has been successfully created 1 <u>Portfolio Credit</u> will be deducted. | |
| <u>BR2</u> | Valid Portfolio/Building Name | A valid name for a building or a portfolio satisfies that following three requirements: (1) It is unique in the system relative to a given user account and (2) it ranges between 3 and 30 characters in length. (3) Permissible characters include numbers ['0'..'9'], lowercase letters['a'..'z'], uppercase letters['A'..'Z'], dashes ['-'], and underscores ['_']. | Ken, please revise naming conventions. |
| <u>BR3</u> | Adding Portfolio Administrator Permissions | The portfolio owner may add portfolio administrator permissions to a <u>GBI User Account</u> , if the following requirements are satisfied: (1) The <u>GBI User Account</u> is a full user account and (2) the maximum number of 5 Portfolio Administrators has not yet been reached. | Ken, if needed remove or revise the maximum number of Portfolio Administrator for a Portfolio. |
| <u>BR4</u> | Reimbursement of Portfolio Credits | In case the portfolio owner decides to delete a <u>Portfolio</u> , 1 <u>Portfolio Credit</u> will be added to the total amount of <u>Portfolio Credits</u> . | |

Use Case Actors

- **Actors** represent users or entities that interact with the system. By definition, actors are outside of the system boundary.
- **Primary Actor:**
 - Typically a user
 - Initiates the use case in order to accomplish a pre-set goal
- **Secondary Actor:**
 - Plays the role of supporting the execution of the use case
 - Participates in the interaction later

Linking Use Cases

- A use case may **include** sub-use cases or may **extend** an existing use case
- Use Case «includes»:
 - Invokes a sub-use case
 - Often an extension is needed if the sub-use case terminates unsuccessfully
- Use Case «extends»:
 - Expands an existing base use case with additional interactions
 - This relationship is often used to model optional or seemingly unrelated behavior relative to the goal of the base use case

«includes» Relationship

Meaning:

- Base use case invokes a sub use case

Usages:

- Reuse of existing use cases
- Reduction of complexity
- Elimination of redundancies by grouping common sub flows

«includes» Relationship (cont.)

Examples:

- “Order Product” «includes» “Login”
- “Change Order Status” « includes » “Check Access Rights”
- “Withdraw Money” « includes » “Check PIN Code”

Important:

- To ensure reusability, the base use case is dependent on the (result of the execution of the) included use case, but *not* the other way around
- Included use case must *not* depend on terminology, actors or business rules stated in the base use case

«includes» Relationship: Example (Version 1)

Use Case: Order Product

Properties

...

Primary Actor: Customer

....

Main Success Scenario

1. Customer submits login coordinates.
2. System authenticates Customer.
3. System informs Customer that login was successful.
4. System grants access to Customer based on Access Levels.
5. Customer specifies desired Product Category.
6. System

...

Extensions

2a. Login coordinates are invalid (BR 1)

2a1. System informs *Customer* that login was not successful.

2a2. Use case *terminates unsuccessfully*

«includes» Relationship: Example (Version 1)

Use Case: Order Product

Properties

...

Primary Actor: Customer

....

Main Success Scenario

1. Customer submits login coordinates.
2. System authenticates Customer.
3. System informs Customer that login was successful.
4. System grants access to Customer based on Access Levels.
5. Customer specifies desired Product Category.
6. System

...

Extensions

2a. Login coordinates are invalid (BR 1)

- 2a1. System informs *Customer* that login was not successful.
- 2a2. Use case *terminates unsuccessfully*

«includes» Relationship: Example (Version 2)

Use Case: Order Product

Properties

...

Primary Actor: Customer

....

Main Success Scenario

1. Customer performs Login
2. System grants access to Customer based on Access Levels.
3. Customer specifies desired Product Category.
4. System

...

Extensions

1a. Login fails

1a1. Use case *terminates unsuccessfully*.

«includes» Relationship: Example (Version 2)

Use Case: Order Product

Properties

...

Primary Actor: Customer

....

Main Success Scenario

1. Customer performs Login
2. System grants access to Customer based on Access Levels.
3. Customer specifies desired Product Category.
4. System

...

Extensions

1a. Login fails

1a1. Use case *terminates unsuccessfully*.

«includes» Relationship: Example (Version 2 cont.)

Use Case: Login

Properties

...

Primary Actor: Customer

....

Main Success Scenario

1. Customer submits login coordinates.
2. System authenticates Customer.
3. System informs Customer that login was successful.
4. Use case *terminates successfully*.

...

Extensions

2a. Login coordinates are invalid (BR 1)

- 2a1. System informs Customer that login was not successful.
- 2a2. Use case terminates unsuccessfully.

«extends» Relationship

Base use case is extended (enriched) by the extending use case

Extension occurs at pre-defined extension point(s), which is defined in the extending use case.

Examples

- “Re-stock Item” extends “Order Item”
- “Log Transaction” extends “Withdraw Money”
- ~~“Product not in Stock” extends “Order Product”~~

«extends» Relationship (cont.)

Usages

- Expansion of an existing base use case (without changing the base use case)
- Introduction of new features after initial requirements has been signed off / reviewed
- Elimination of redundancies (grouping of common sub flows)
- Addition of features that conceptually are unrelated to use case goal (e.g., logging)

Important

- Extending use case is dependent on base use case and not the other way around
- As a result, an extending use case can be added to the model without affecting the base use case
- One use case may extend several other use cases

«extends» Relationship: Example

Use Case: Order Product

Properties

...

Primary Actor: Customer

....

Main Success Scenario

1. Customer performs Login
2. System grants access to Customer based on Access Levels.
3. Customer specifies desired Product Category.
4. ...
5. System invoices the Order and updates the Inventory.
6. System informs Customer that Order has been invoiced

...

Extensions

...

«extends» Relationship: Example

Use Case: *Restock Item* (abstract use case)

Properties:

Precondition: Number of items in stock is below threshold level (BR 2)

Extends:

Order Product @ Step 5

...

Main Success Scenario

1. System submits restock order to Product Inventory System
2. System notifies Product Manager that a restock has been submitted
3. Use case *terminates successfully*.

...

Extensions

...

«extends» Relationship: Example

Use Case: *Restock Item* (abstract use case)

Properties:

Precondition: Number of items in stock is below threshold level (BR 2)

Extends:

Order Product @ Step 5

...

Main Success Scenario

1. System submits restock order to Product Inventory System
2. System notifies Product Manager that a restock has been submitted
3. Use case *terminates successfully*.

...

Extensions

...

Basic Use Case Guidelines

- Use simple grammar:
 - Subject ... verb ... direct object
 - Use an expressive verb in present tense, for example: The system deducts the amount from the account
- Specify whether the system or actor performs the use case step
- Write from a commentator's view
- Show the process moving forward
- Keep UI details out! Show intent, not actions

Common Mistake #1

- Use Cases should **not** contain UI details
- This is problematic because the UI is more prone to change compared to core functionality

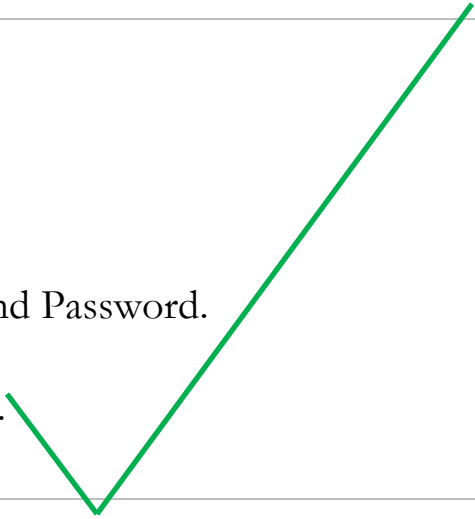
Use Case: Buy XYZ

- **Level:** user-goal
- **Primary Actor:** Customer
- **Main Flow:**
 1. System presents ID and Password screen with two input fields.
 2. Customer types ID and password into system and then clicks the “Okay” button at the bottom of the screen.
 3. System validates user.
 4. Customer types last name first in the “last name” field, then first name

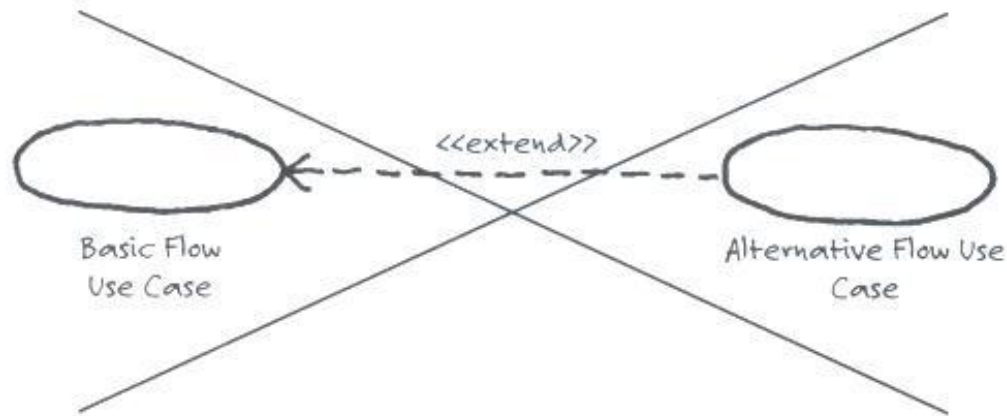
Solution

- Abstract the Use Case from UI Details
- Results in less clutter and redundancies, reduced maintenance overhead

Use Case: Buy XYZ

- **Level:** user-goal
 - **Primary Actor:** Customer
 - **Main Flow:**
 1. Customer accesses system with ID and Password.
 2. System validates user.
 3. Customer provides name and address.
 4. ...
- 

Common Mistake #2: Misuse of «extends»



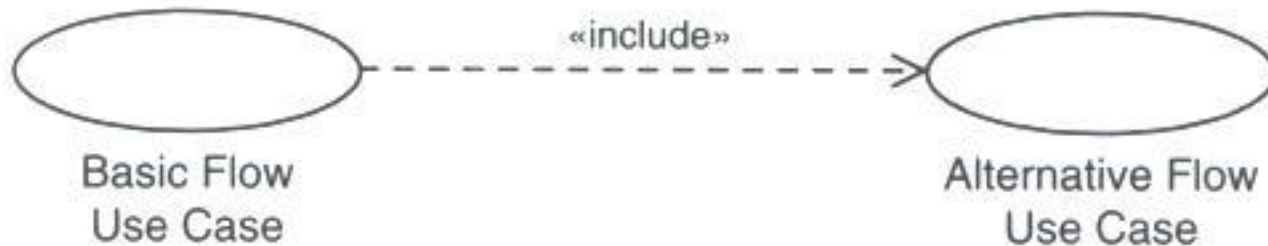
Detection:

- Base use case is incomplete without extending use case
- Flow of extending use case replaces part of the base use case flow

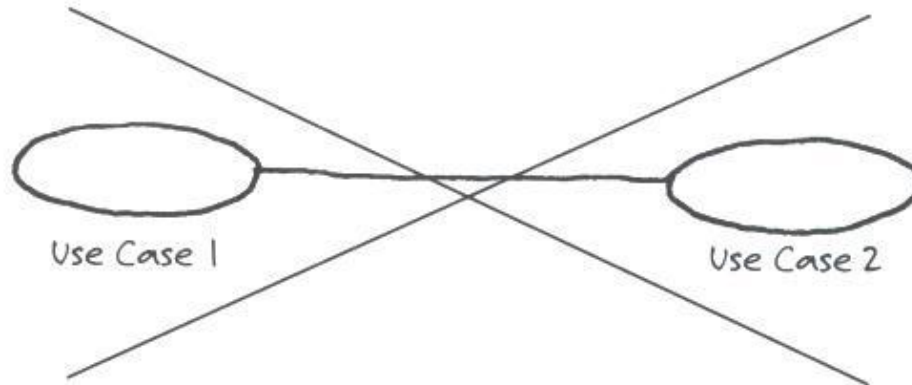
Common Mistake #2: Misuse of «extend»

Solution:

- Merge base use case with extending use case
- If appropriate, make use of «include» relationship



Common Mistake #3: Communicating Use Cases



Detection:

- Two use cases are related by an association which is not «includes» or «extends»

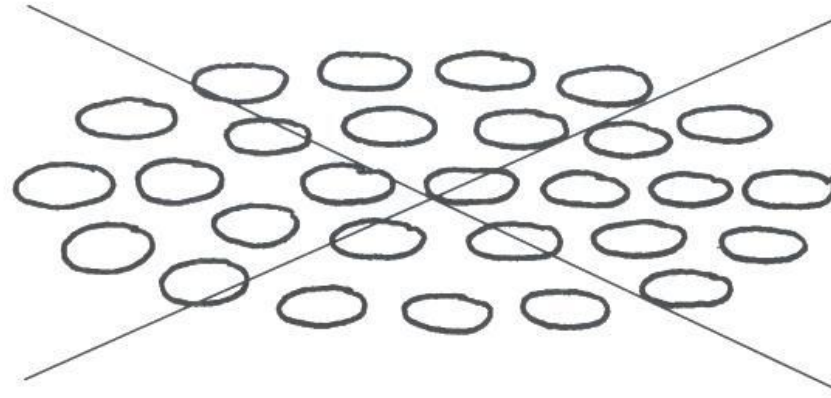
Rationale:

- Use cases represent self-contained functionalities and do not communicate with each other

Solution:

- Merge both use cases

Common Mistake #4: Micro Use Cases



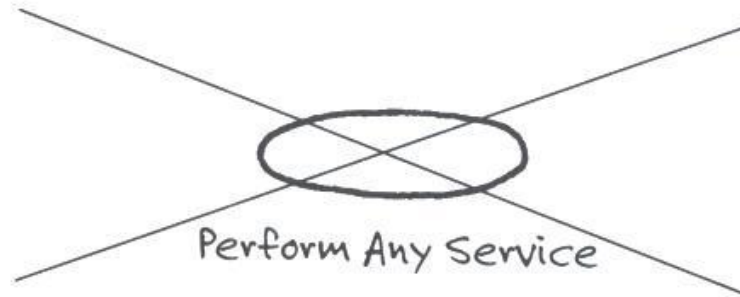
Detection:

- Large number of use cases named after single (system) operations
- Each use case contains very little value for stakeholders

Solution:

- Group use cases according to user goals
- Raise abstraction level

Common Mistake #5: Multiple Business Values



Detection:

- Use case addresses more than one use goal

Solution:

- Split use case into several user-goal use cases

Other Mistakes

| Mistakes | Implication | Solution |
|---|------------------------------|---------------------------------------|
| Use Case is not goal-oriented | Does not reflect user needs | Make use of an actor – goal list |
| Duplication of information | Results in inconsistencies | Use case refactoring |
| Incorrect abstraction level (too many details) | Maintenance overhead | Show process moving forward |
| Conditional Statements | Clutter and poor readability | Use alternative flows (extensions) |

Basic Use Case Refactorings

- A refactoring is a change made to the internal structure of the use case model without changing its meaning
- Examples are renaming, merging UCs, splitting UCs, etc...
- Leads to:
 - less redundancies
 - decreased maintenance overhead
 - improved readability
 - enhanced separation of concerns

Use Case Refactoring #1: Factor out business rules

- **Change:**

- Extract information originating from policies, rules, and regulations of the business from the flow
- Describe this information in a separate section as a collection of business rules referenced from the use case description (e.g., numbered list text document, or as a class model)

- **Gain:**

- Robustness towards change of business rules
- Business rules can be reviewed and updated separately, without changing use case
- Use case becomes more maintainable, shorter and easier to handle.
- Separation of concerns

Use Case Refactoring #2: Extract common step sequences

- **Change:**

- Use cases may overlap
- Extract a sequence of steps that appear in multiple places and express them separately in sub use cases
- Make use of «includes» and «extends» relationships

- **Gain:**

- Less redundancies
- Avoidance of inconsistencies
- Maintainability

Use Case Refactoring #3: Merging CRUD UCs

Change:

- Merge short, simple use cases, such as **C**reating, **R**eading, **U**dating, and **D**eleting into a single use case forming a conceptual unit

Gain:

- Coherence
- Conciseness

Attention:

Only feasible if all CRUD UCs are performed in a similar manner

Use Case Refactoring #4: Elimination of long and broad UCs

Change Long UCs (long scenarios):

- Introduction of sub-UCs to model parts of the base UC
- Raising level of abstraction

Change Broad UCs (many extensions):

- Introduction of sub-UCs to capture extensions

Gain:

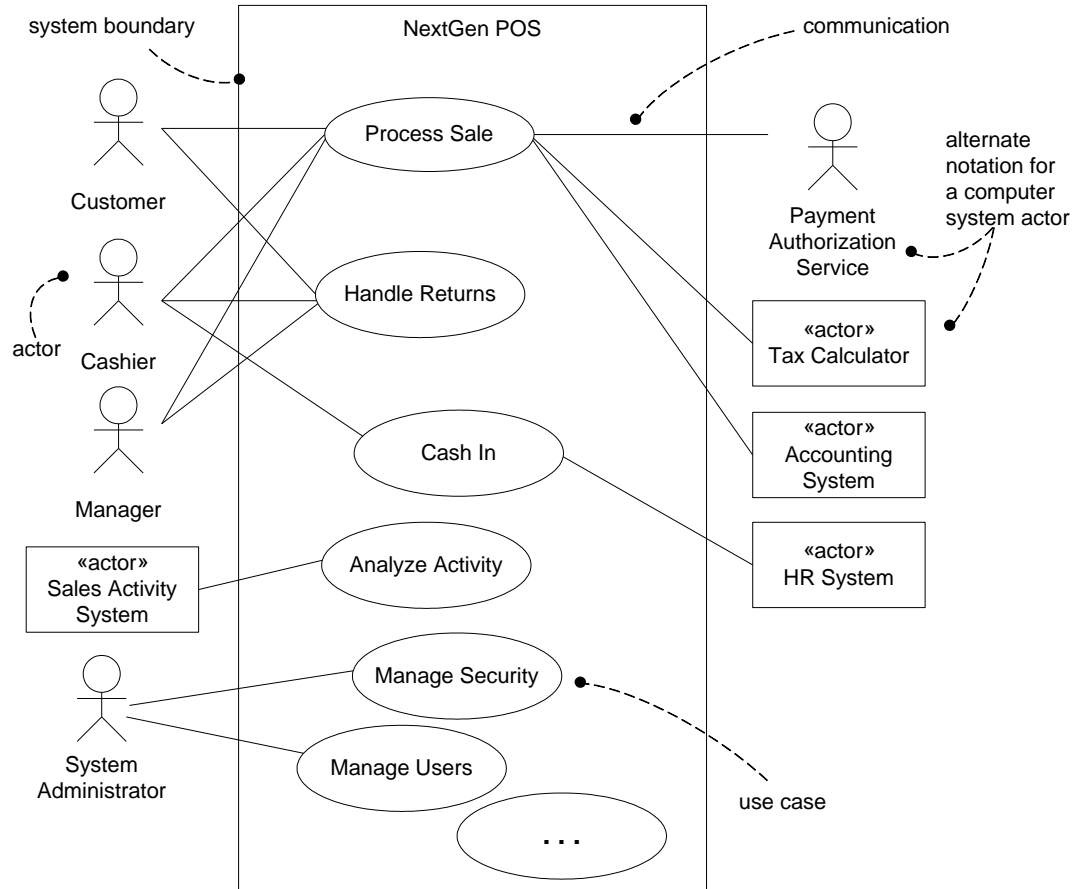
- Readability
- Decreased cognitive load

[Use Case Writing Guidelines]

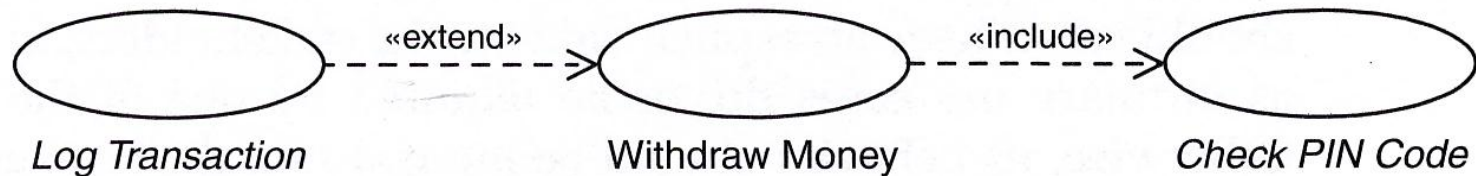
Use Case Diagrams

- Like a UC Model table of contents (ToC)
 - A ToC is no substitute for the content of a book.
 - Use Case Diagram(s) are no substitute for the content of a Use Case model.
- UC diagrams show:
 - Most important use cases (incl. UC inter-relationships)
 - Actors involved (with each use case).
 - Primary actors are displayed on the left hand side of the system boundary
 - Secondary actors are displayed on the right hand side of the system boundary
 - System name and System boundary

Use Case Diagram Example: POS System



Use Case Relationships in Use Case Diagrams



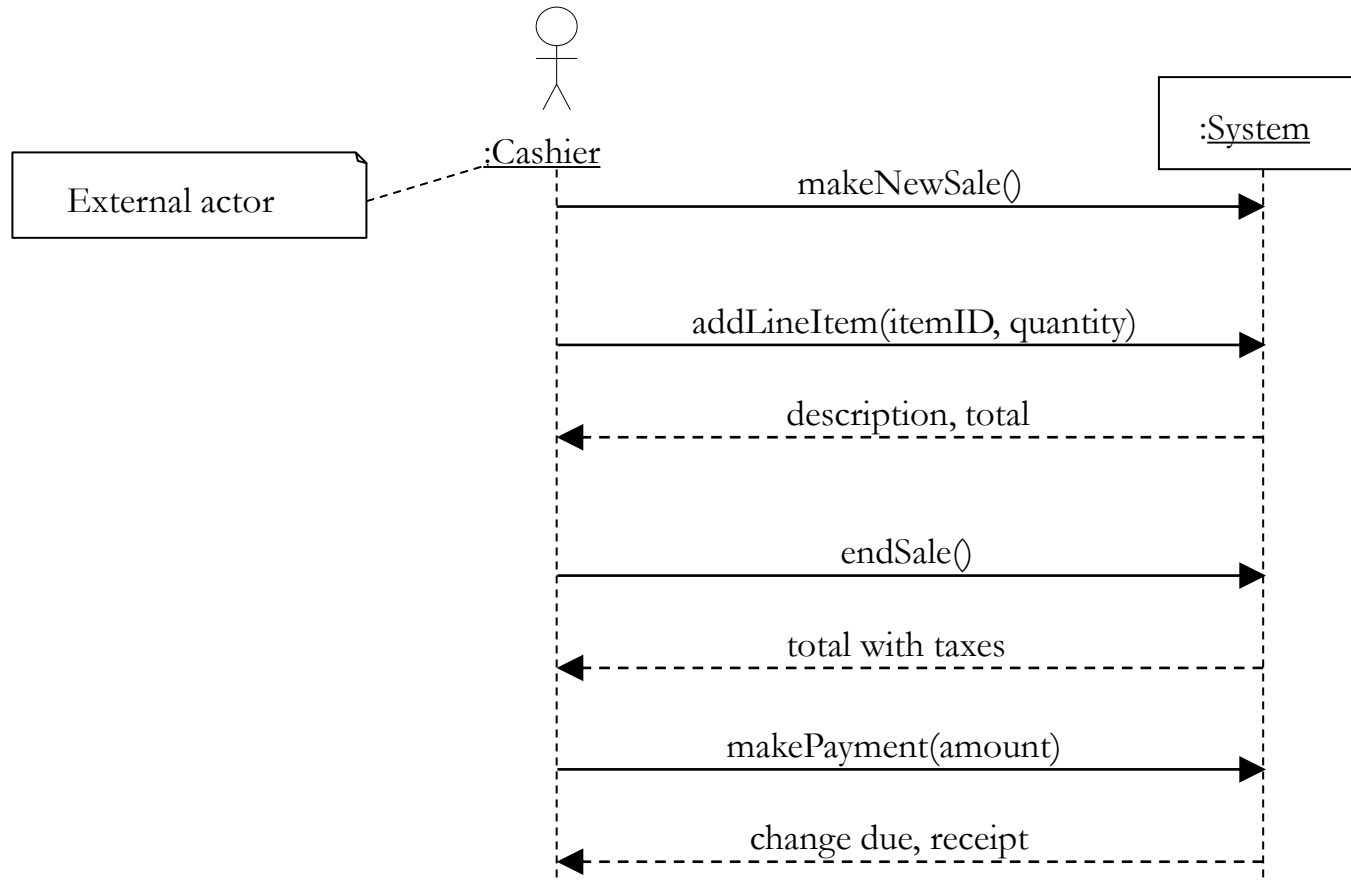
System Behavior and UML Sequence Diagrams

- It is useful to investigate and define the behavior of the software as a “black box”.
- System behavior is a description of *what the system does* (without an explanation of how it does it).
- Use cases describe how external actors interact with the software system. During this interaction, an actor generates events.
- A request event initiates an operation upon the system.

System Behavior and System Sequence Diagrams (SSDs)

- A sequence diagram is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events.
- All systems are treated as a black box; the diagram places emphasis on events that cross the system boundary from actors to systems.

SSDs for Process Sale Scenario

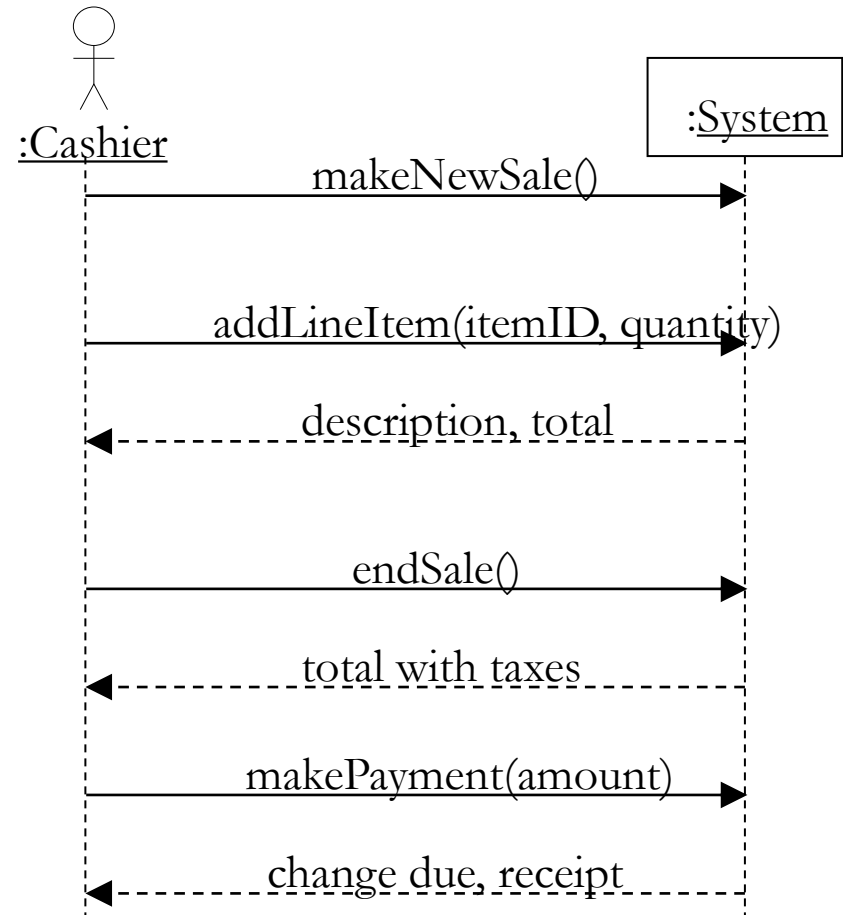


SSD and Use Cases

Simple Process Sale Use Case Scenario

1. Customer arrives at a POS checkout with goods to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier and quantity.
4. System records sale line item, and presents item description, price and running total.
5. System presents total with taxes calculated.

...



Book Metaphor

– Use Case Model

- Use case diagram(s)
- **Use case descriptions**
- System Sequence Diagrams

- UC diagrams are *optional*
- System Sequence Diagrams are *optional*

– Book

- Table of Contents (ToC)
- **Book content**
- Appendix

- ToC is *optional*.
- Appendix is optional

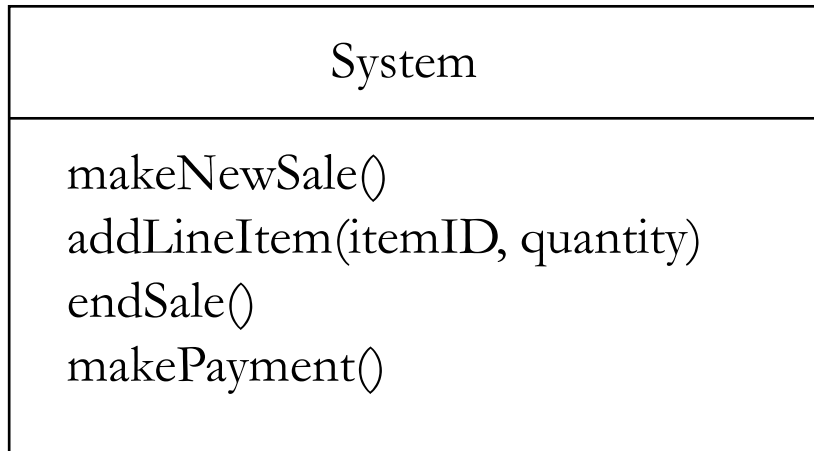
Naming System Events and Operations

- The set of all required system operations is determined by identifying the system events.
 - makeNewSale()
 - addLineItem(itemID, quantity)
 - endSale()
 - makePayment(amount)

Operations Contracts

- Contracts are documents that describe system behavior.
- Contracts may be defined for **system operations**.
 - Operations that the system (as a black box) offers in its public interface to handle incoming system events.
- The entire set of system operations across all use cases, defines the public system interface.

System Operations and the System Interface



- In the UML the system as a whole can be represented as a class.
- Contracts are written for each system operation to describe its behavior.

Example Contract: addLineItem

Contract: addLineItem

Operation: addLineItem (itemID: ItemID, quantity: integer)

Cross References: Use Cases: Process Sale.

Pre-conditions: There is a sale underway.

Post-conditions:

- ❑ A SalesLineItem instance *sli* was created. (instance creation)
- ❑ *sli* was associated with the Sale. (association formed)
- ❑ *sli.quantity* was set to quantity. (attribute modification)
- ❑ *sli* was associated with a ProductSpecification, based on itemID match (association formed)

Pre- and Postconditions

- Preconditions are assumptions about the state of the system before execution of the operation.
- A postcondition is an assumption that refers to the state of the system after completion of the operation.
 - The postconditions are not actions to be performed during the operation.
 - Describe changes in the state of the objects in the Domain Model (instances created, associations are being formed or broken, and attributes are changed)