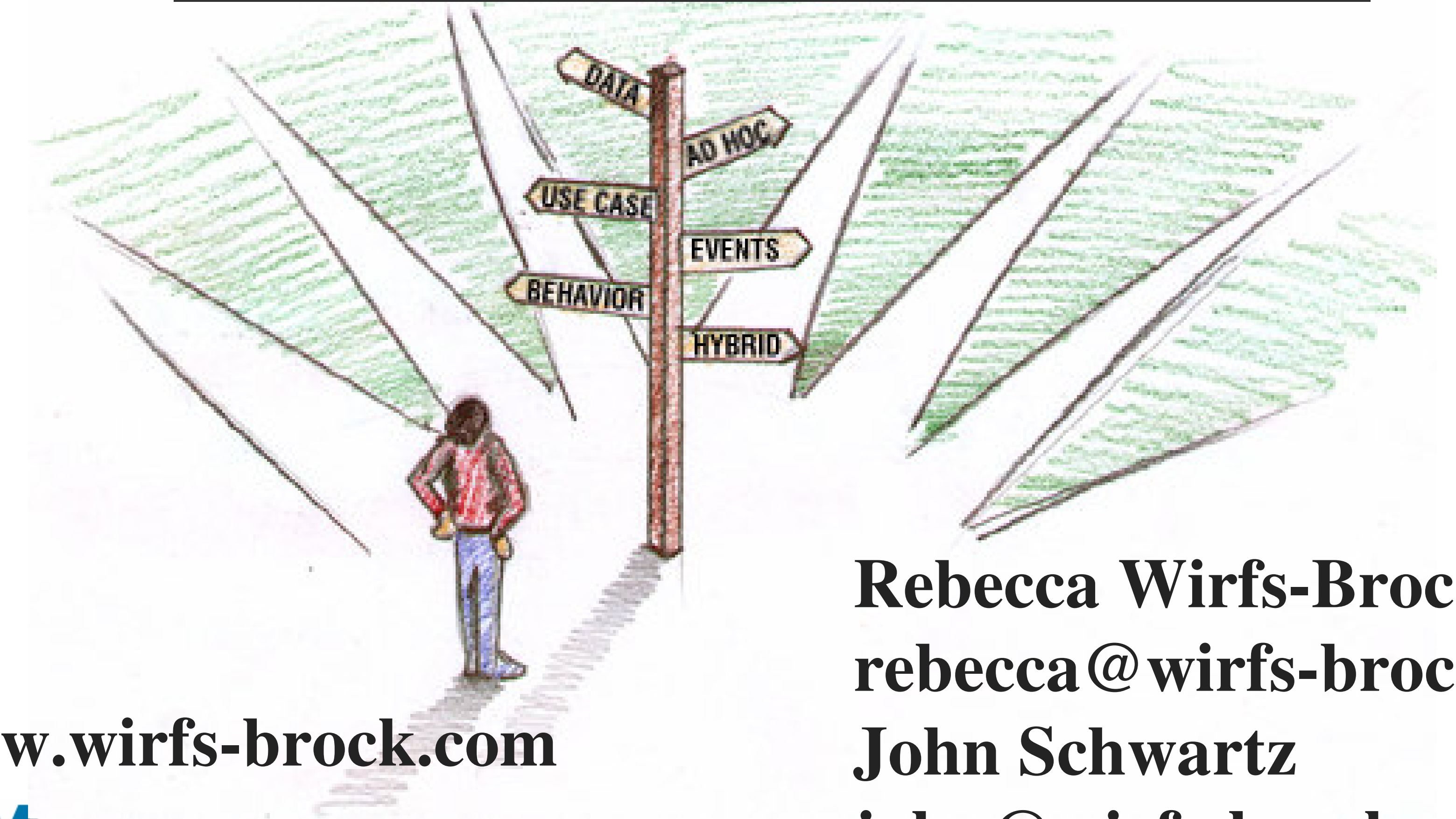
The Art of Writing Use Cases



Rebecca Wirfs-Brock
rebecca@wirfs-brock.com
John Schwartz
john@wirfs-brock.com

www.wirfs-brock.com

Goals

The goal of this course is to enable you to

– understand use case models: actors, use cases, glossaries and use case diagrams

– use three forms of use case descriptions

– write effective use case descriptions

– critique use case descriptions

– relate use cases to business policies, UI prototypes and other requirements

– add detail and precision to use case descriptions

Agenda

Use Cases, Actors and Glossaries

Exercise 1: Find Use Cases and Actors

Let’s Tell a Story

Exercise 2: Write Use Case Narratives

Scenarios and Conversations: Tips and Guidelines

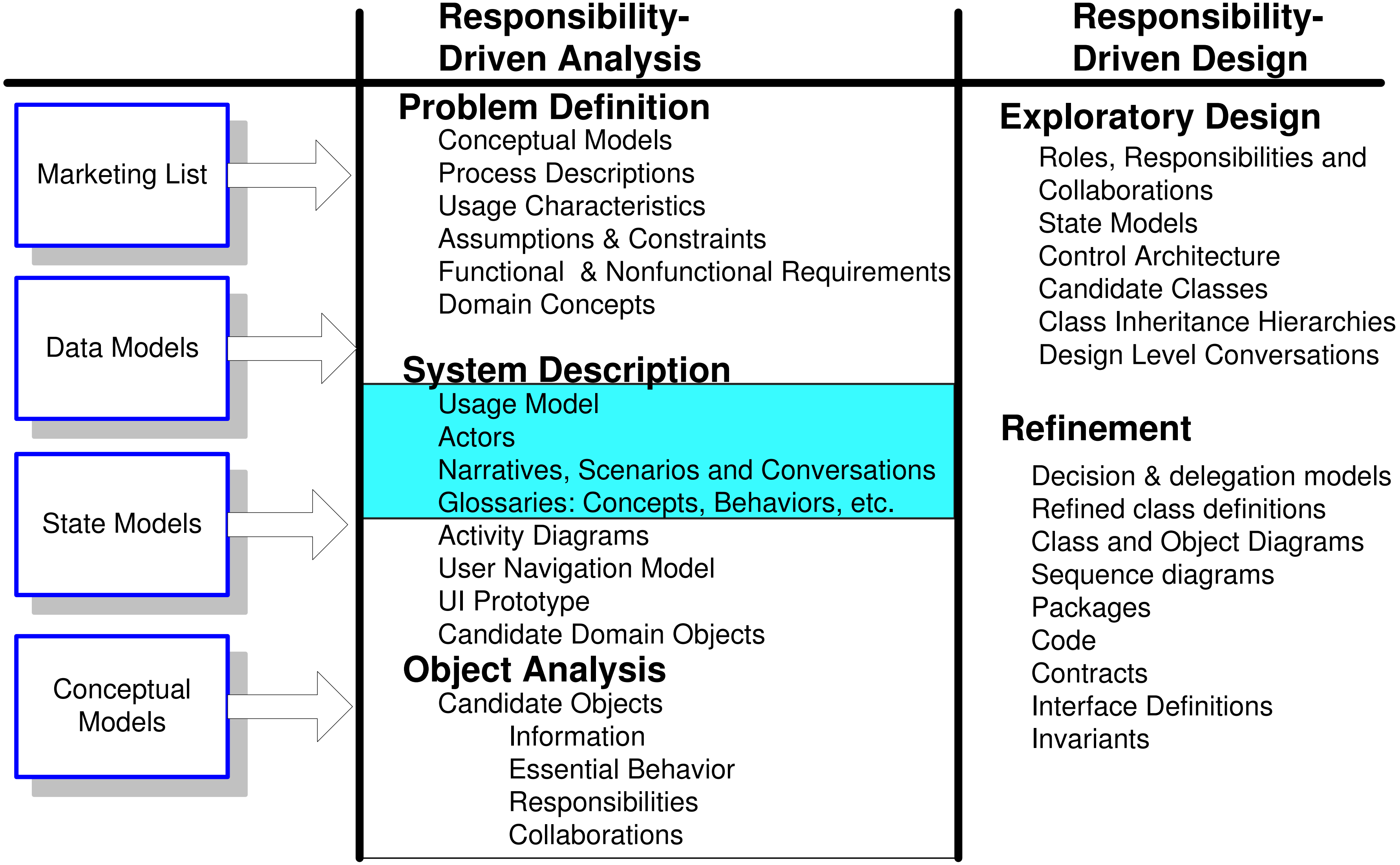
Exercise 3: “Clinic” a Scenario

Exercise 4: Write a Conversation

Alternatives: Exceptions and Variations

Exercise 5: Describing Alternatives

Scope of Tutorial



Marketing List

Data Models

State Models

Conceptual Models

Responsibility-Driven Analysis

Problem Definition

Conceptual Models
Process Descriptions
Usage Characteristics
Assumptions & Constraints
Functional & Nonfunctional Requirements
Domain Concepts

System Description

Usage Model
Actors
Narratives, Scenarios and Conversations
Glossaries: Concepts, Behaviors, etc.
Activity Diagrams
User Navigation Model
UI Prototype
Candidate Domain Objects

Object Analysis

Candidate Objects
- Information
- Essential Behavior
- Responsibilities
- Collaborations

Responsibility-Driven Design

Exploratory Design

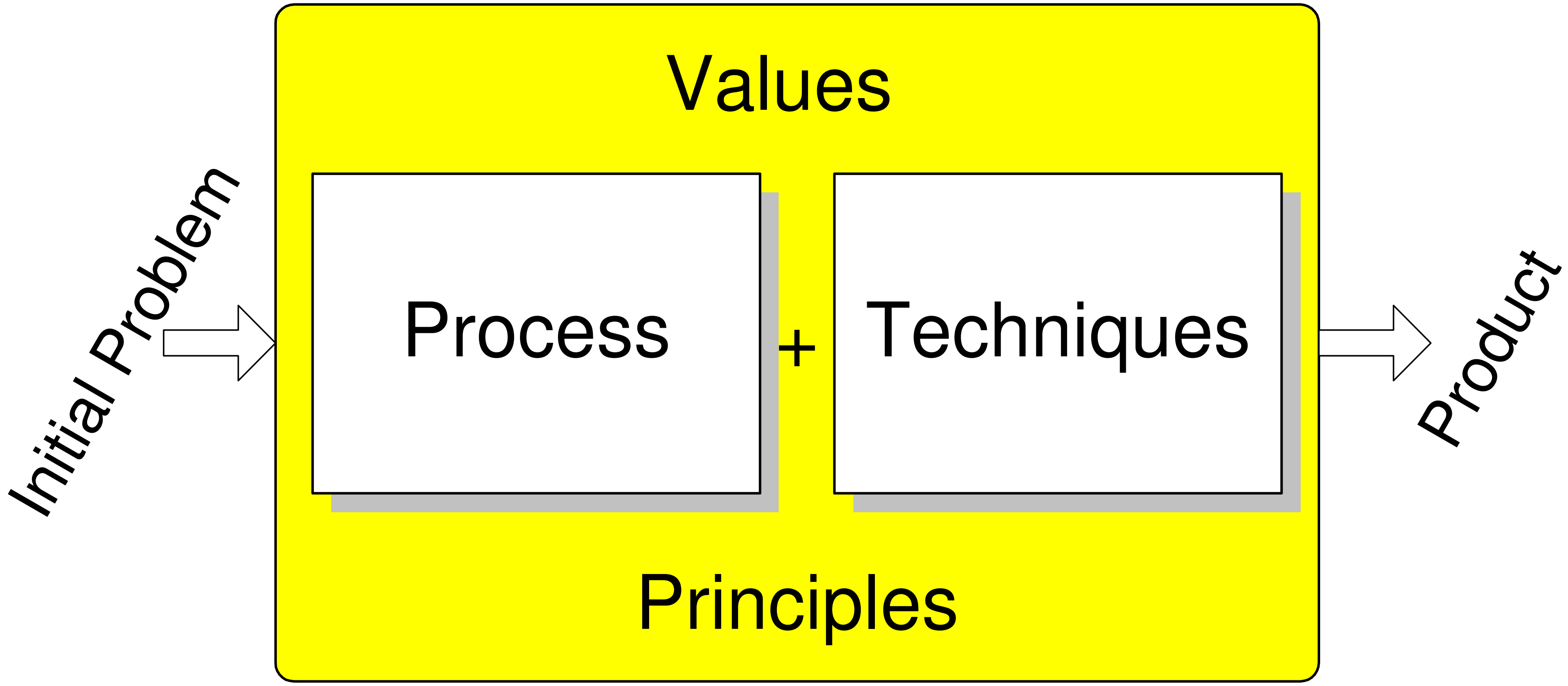
Roles, Responsibilities and Collaborations
State Models
Control Architecture
Candidate Classes
Class Inheritance Hierarchies
Design Level Conversations

Refinement

Decision & delegation models
Refined class definitions
Class and Object Diagrams
Sequence diagrams
Packages
Code
Contracts
Interface Definitions
Invariants

Use Cases, Actors and Glossaries

A Context


Values

Process

+

Techniques

Principles

Initial Problem

Product

Tell a Story

Cover the basics

- Key requirements for your application: use cases, scalability, etc.
- Glossary can assist

Unfold your story

- Choose the right form
- Choose a level of detail appropriate to your audience
- Don’t tell everything at once. Reveal details as needed
- Consider different actors’ perspectives

Use Case

Functionality from a particular point-of-view

A collection of task-related activities...

Online Banking Use Cases

making a payment

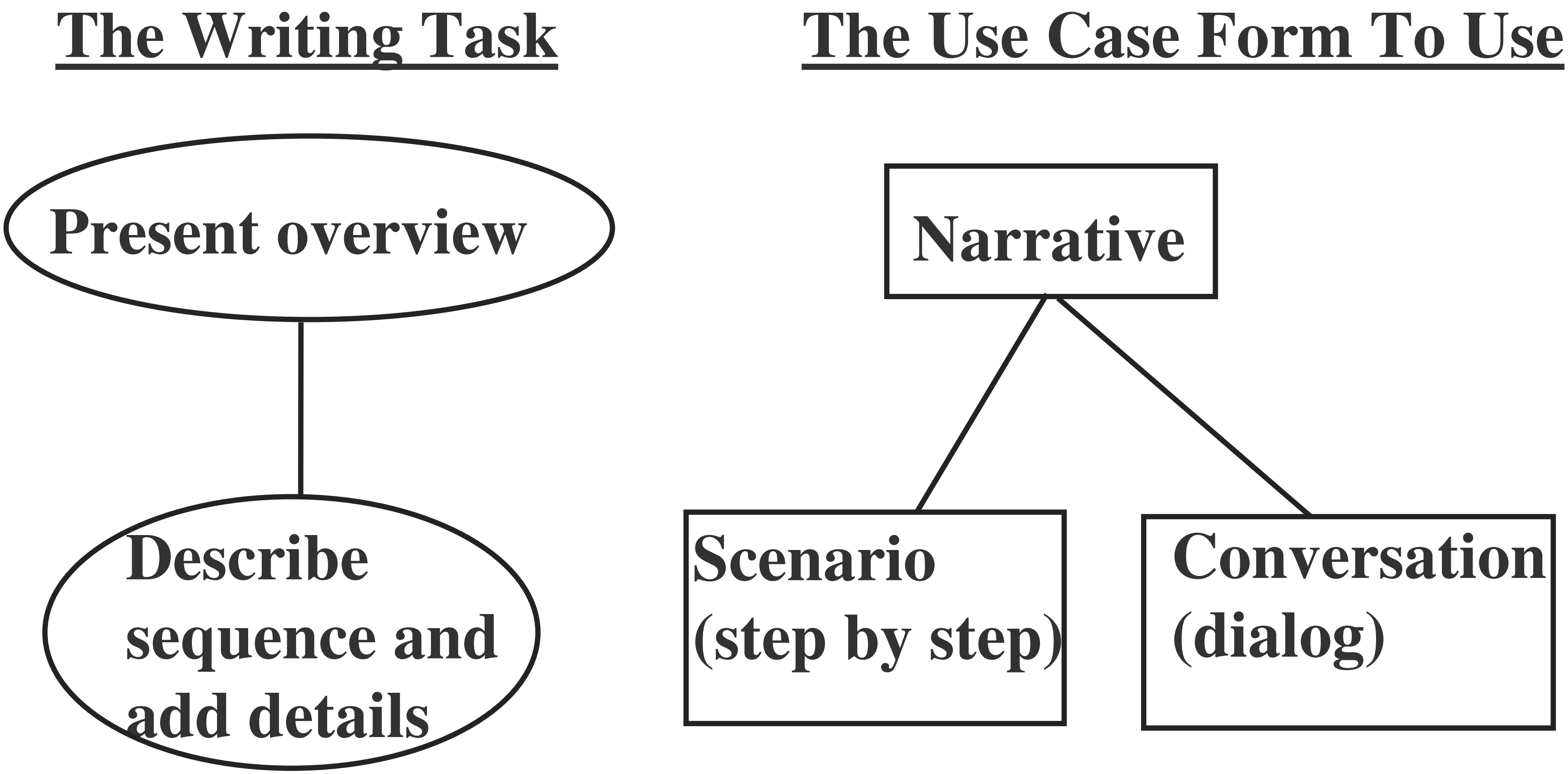
transferring funds between accounts

reviewing account balances

… describing a discrete "chunk" of the system

Use cases describe a system from an external usage viewpoint

Function and Form



First Form: A Narrative

Make a Payment

The user can make online payments to vendors and companies known to the bank. Users can apply payments to specific vendor accounts they have. There are two typical ways to make payments: the user can specify a one-time payment for a specific amount, or establish regular payments to made on a specific interval such as monthly, bi-weekly, or semi-annually.

Narrative Form

Free-form text in paragraph format

Describes the intent of the user in performing the use case

Describes high-level actions of the user during the use case

Refers to key concepts from the problem domain that are involved in the use case.

Second Form: A Scenario

Register Customer With Automatic Activation

1 User enters registration information:

Required information: user name, email address, desired login ID and password, and confirmation password

One of: account number and challenge data, or ATM # and PIN

Optional: language choice and company

2 System checks that password matches confirmation password.

3 System validates required fields and verifies uniqueness of login ID

4 System verifies customer activation information.

5 System creates and activates customer online account.

6 System displays registration notification.

Scenario Form

One particular path through a use case written from the actor’s point of view

Describes a sequence of events or list of steps to accomplish

Each step is a simple declarative statement with no branching

May describe:

- Actors and their intentions
- System responsibilities and actions

Third Form: A Conversation

Make A Payment

General Flow

Actor: User	System: Application
	Present list of payment templates to user organized by payee category
Select a payment template	
	Present details of selected Payment Template and recent payment history to payee
Enter payee notes, amount and account Submit payment information	
	Apply payment to payee Add new payment to recent payment list Redisplay the payment list
Optionally, request Setup Payment	Goto **Edit Payment Template Information**
Select next function	Goto **selected use case**

Optional Action

Multiple Actions

Invoking Another Use Case

Conversation Form

One path through a use case that emphasizes interactions between an actor and the system

Can show optional and repeated actions

Each action can be described by one or more substeps

May describe:

- Actor actions
- System responsibilities and actions

Comparing the Three Forms

Form	Strengths	Weaknesses
Narrative	• Good for high-level summaries and intentions • Can be implementation-independent	• Easy to write at too high or too low a level • Not suitable for complex descriptions • Can be ambiguous about who does what
Scenario	• Good for step-by-step sequences	• Hard to show parallelism, arbitrary ordering or optionality • Can be monotonous
Conversation	• Good for seeing actor-system interactions • Can show parallel and optional actions	• Easy to write to pseudo-code • Difficult to show repetition
All Forms	• Informal	• Informal

The Benefits of Use Cases

Use cases describe a system from an external usage perspective

They can be organized according to their relevance, frequency of use, and perceived value to the system’s users

System features can be correlated with how they are used within specific use cases

Impacts of adding and/or removing features on system usability can be analyzed

Use Cases Aid Understanding

Capture information in a natural way

Users: *“You mean we’ll have to …????”*

Discover “holes” in the understanding of a system

Sponsors: *“You left out one thing here ...”*

Organize work supported by the system

Developers: *“Hmm, these aren’t just a bulleted list of functions!”*

Use Cases Vary by Abstraction Level

Steve Registers for English 101, or

Student Registers for Course, or

User Uses System, or

Student Registers for Variable Credit Course, or

Student Registers for Music Course

Use Cases Vary in Scope

Which system boundary do we mean?

- component: describing the web applet
- application: online banking
- organization: the bank

We typically start by describing application level scope

Use Cases Vary in Detail

Do we describe general actions?

Enter deposit amount

or specific details?

Press number keys followed by enter key

Write at the level that seems appropriate to your readers

This typically means describing actor actions and system responses that match the goal for the use case

What Use Cases Cannot Do

Use Cases are best used to describe system functionality from a task-oriented perspective

They do not describe:

- user interfaces
- performance goals
- application architecture
- non-functional requirements

Finding Use Cases

Describe end user goals supported by the system…

“Transfer money between accounts...”

“Get money...”

“Make payments...”

“Set up vendors for automatic payments…”

Finding Use Cases

Describe the functions that the user will want from the system

Describe the operations that create, read, update, and delete information

Describe how actors are notified of changes to the internal state of the system

Describe how actors communicate information about events that the system must know about

Naming Use Cases

Name a use case with a verb-noun phrase that states the actor’s goal

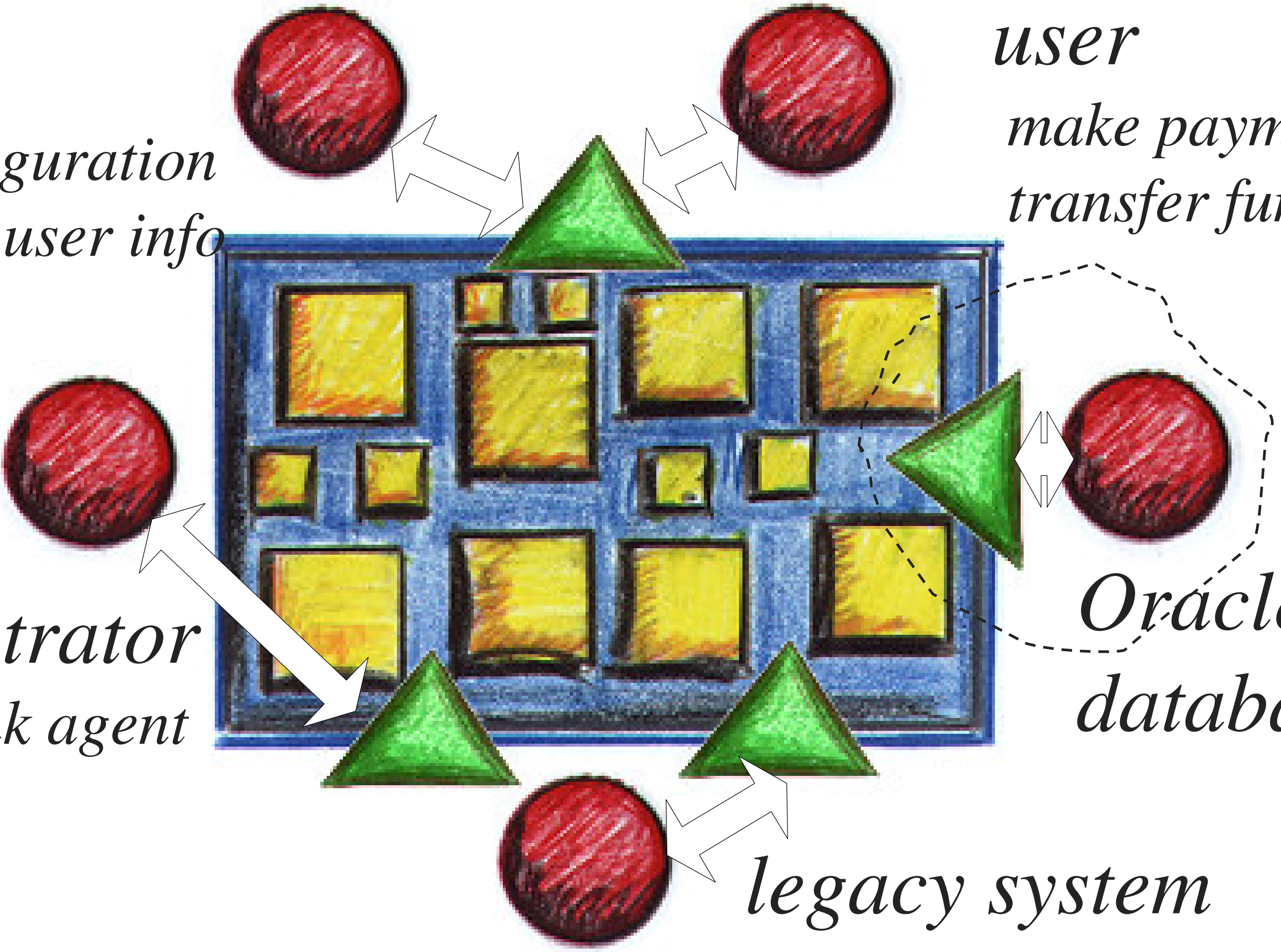
Use concrete, “strong” verbs instead of generalized, weaker ones. Weak verbs may indicate uncertainty

- Strong Verbs: create, merge, calculate, migrate, receive, archive, register, activate
- Weaker Verbs: make, report, use, copy, organize, record, find, process, maintain, list

Be explicit. Use specific terms. They are stronger

- Strong Nouns: property, payment, transcript, account
- Weaker Nouns: data, paper, report, system, form

Different Perspectives



operator

edit configuration

maintain user info

user

make payment

transfer funds

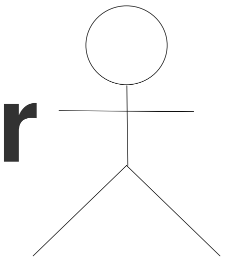
administrator

add bank agent

Oracle database

legacy system

Actor



Any one or thing that interacts with the system causing it to respond to business events

Something that

- stimulates the system to react (primary actor), or
- responds to the system's requests (secondary actor)

Something we don't have control over

Primary and Secondary Actors

Primary Actor— Any one or thing that interacts with the system causing it to respond to business events

Something we don't have control over

Secondary Actor— Something or someone that responds to system requests

Something the system uses to get its job done

Naming Actors

Group individuals according to their common use of the system. Identify the roles they take on when they use or are used by the system

Each role is a potential actor

Name each role and define its distinguishing characteristics. Add these definitions to your glossary

Don’t equate job title with role name. Roles cut across job titles

Use the common name for an existing system; don’t invent a new name to match its role

Don’t waste time debating actor names

Places to Look for Actors

Who uses the system?

Who gets information from this system?

Who provides information to the system?

What other systems use this system?

Who installs, starts up, or maintains the system?

Finding Actors

Focus initially on human and other primary actors

Group individuals according to their common tasks and system use

Name and define their common role

Identify systems that initiate interactions with the system

Identify other systems used to accomplish the system’s tasks

Use common names for these other “system” actors

Actor and Use Case Checklist

What system requirements are not represented by use cases?

- Document those that are internal to the system (can’t be seen by actors) elsewhere

Do all actors and use cases have descriptive names?

- Do those that need explanation have short descriptions?

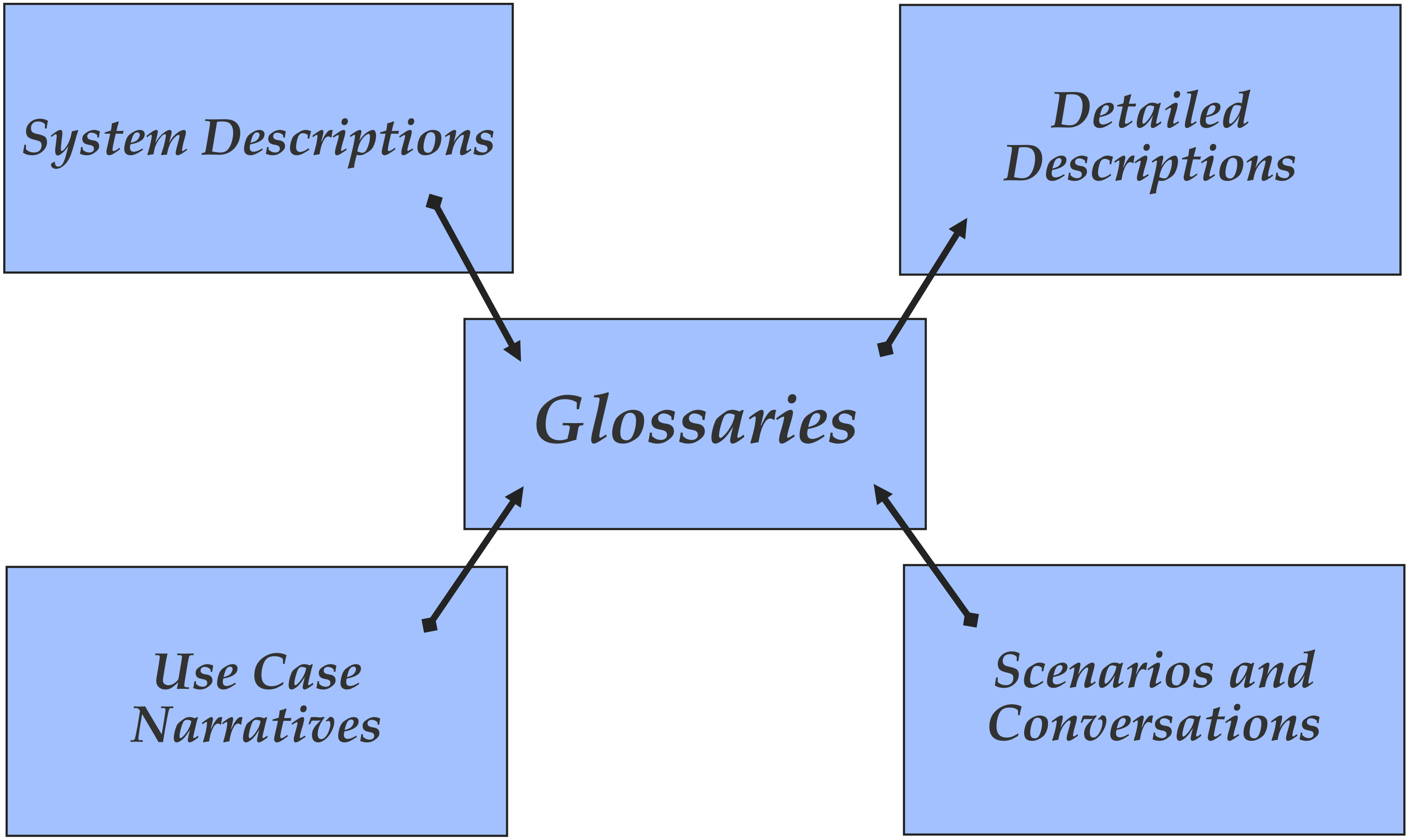
Are system boundaries and scope clear?

Are areas of uncertainty documented as assumptions and issues?

Exercise

Find Actors and Use Cases

Glossaries


System Descriptions
Detailed Descriptions
Glossaries
Use Case Narratives
Scenarios and Conversations

Glossary

A glossary is a central place for:

- Definitions for key concepts
- Clarification of ambiguous terms and concepts
- Explanations of jargon
- Definitions of business events
- Descriptions of software actions

The glossary is built incrementally

Build Consensus

Agree on the problem to be solved!

Define terms in a glossary

- Identify similar behaviors that have different names
- Identify different behaviors that have the same name
- Choose ONE definition!

Use team development and review

Defining Concepts

Identify a concept and its distinguishing characteristics

More than a synonym for a word

Identifies a way of mentally dividing reality for purpose of talking or thinking

Writing Glossary Entries

Why this concept is important

Typical sizes or values

Clarify likely misunderstandings

Show an example

Explain graphical symbols

Relate entries

A Good Form for Definitions

Name of Concept related to a **Broader Concept + Characteristics**

Contrast: A compiler is a program that translates source code into machine language

With a definition that leaves out context: A compiler translates source code into machine language

What performs this translation? A computer? A person?

Improving Glossary Definitions

Contrast the original:

Account In the online banking system there are accounts within the bank which customer-users can access in order to transfer funds, view account balances and transaction historical data, or make payments. A customer has one or more accounts which, once approved by the bank can be accessed. The application supports the ability for customers to inform the system of new accounts, and for the customer to edit information maintained about the accounts (such as name and address information).

With a definition that says what an account is and how it is used:

Account An account is ***a record of money*** **deposited at the bank for checking, savings or other uses.** A customer may have several bank accounts. Once a customer's account is activated for online access, account information can be reviewed and transactions can be performed via the internet.

Another Revision

Automatic activation. Automatic activation is ***an optional function of the online banking software*** **that enables immediate access to bank accounts.** *To automatically activate an account, a customer provides information that associates him with an account, called challenge data, such as mother's maiden name. Online access is granted once the challenge data is validated against bank records. Alternatively, the customer can supply a valid ATM bankcard number and PIN. All accounts associated with that ATM card would be activated.*

Characteristics:

- Optional feature
- Details of how the automatic activation function works

Relating Definitions

Customer-user. A customer-user is ***a person who has online access to banking accounts***. **One or more customer-users are associated with a customer.** Each customer-user can have different access privileges to and visibility of a customer's accounts. For example, in a small business, the accounting customer-user might make vendor payments from an account, while a business manager may simply view an account's transaction history.

Examples add to but don't replace definitions

Customer. A customer is ***a person or organization with one or more bank accounts***. Customers do not use the online banking system, their customer-users do.

Characteristics:

- How a customer-user relates to a customer
- What distinguishes one from another

Use Pictures to Relate Concepts

wire center— the geographical area served by a central office

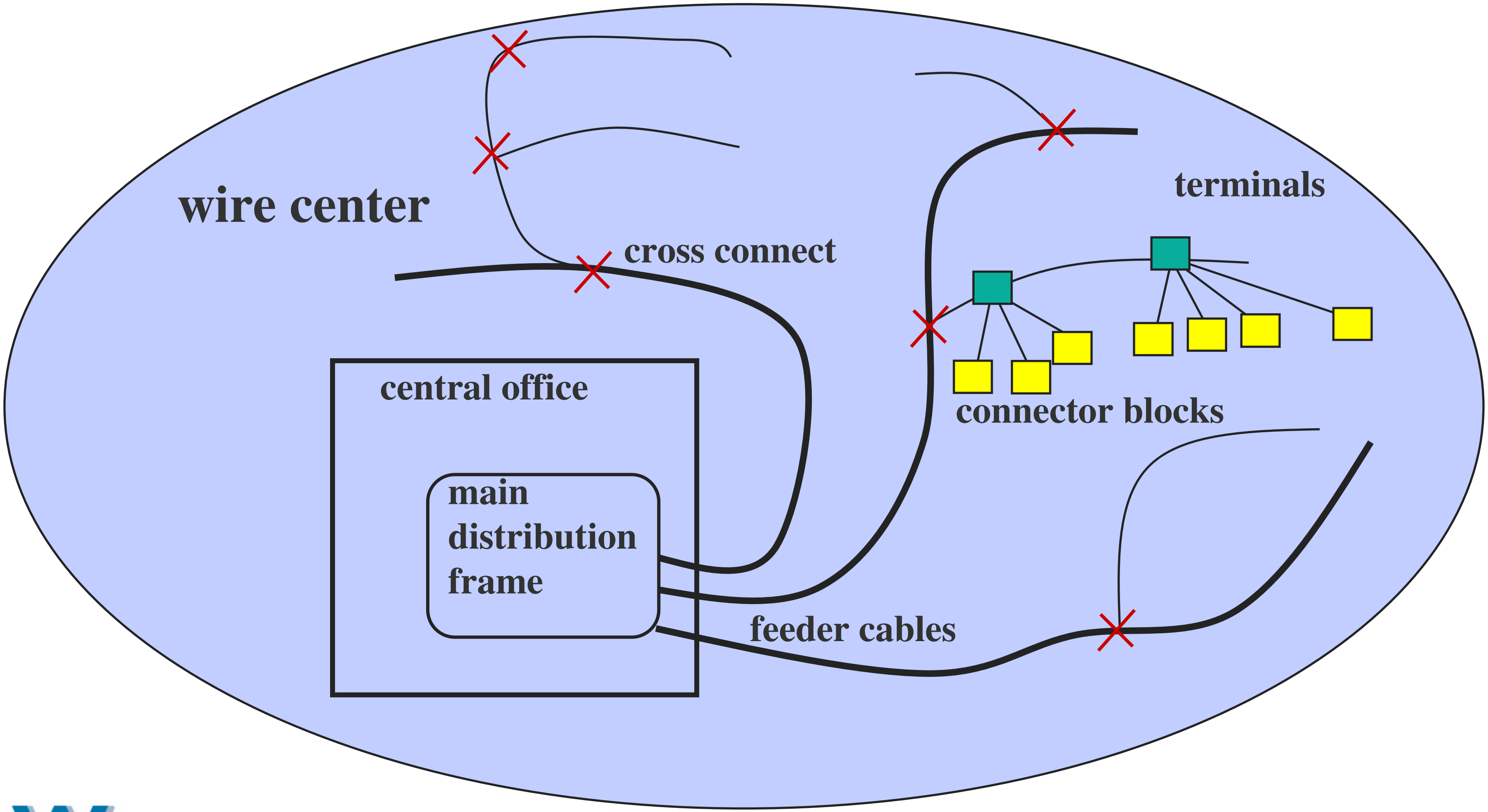
central office— a building where local call switching takes place

main distribution frame— a large connector at a central office, which connects switching equipment to feeder cables

feeder cable— a large cable that connects to the main distribution frame at a central office and feeds into distribution cables

distribution cable— a cable that connects between a feeder cable and one or more terminals

A Picture Relating Hierarchical Concepts



Define Acronyms
and
Their Concepts

Example:

OSS— Operations Support System: As defined by the FCC, a computer system and/or database used at a telephone company for pre-ordering, ordering, provisioning, maintenance and repair, or billing

Avoid Using “Is When” or “Is Where”

Definitions using these words are often missing the broader concept

Contrast: An ***overplot*** is an overlap between two or more graphic entities drawn at the same place on a page

With:An ***overplot*** is when two things overlap

Explain What Is Unclear

A Next Day Air Cautionary Note

Upgrading to Next Day Air does not mean you will get your order the next day.

Once shipped, Next Day Air packages are guaranteed to arrive at the end of the next business day. Note that upgrading method of shipment to Next Day or 2nd Day Air does not change how long it takes to assemble and ship an order – it only reduces the travel time after an order leaves the warehouse. For example, an item marked as “Usually ships within 2 to 3 days” and upgraded to Next Day Air will usually leave our warehouse on the 2nd or 3rd business day and reach you on the 3rd or 4th business day.

Once upon a time...

Let’s Tell a Story

Setting the Stage

Level— summary, core, supporting, or internal use case?

Actor(s)— role names of people or external systems initiating this use case

Context— the current state of the system and actor

Preconditions— what must be true before a use case can begin

Screens— references to windows or web pages displayed in this use case

Completing The Picture

Variations— different ways to accomplish use case steps

Exceptions— errors that occur during the execution of a step

Policies— specific rules that must be enforced by the use case

Issues— questions about the use case

Design notes— hints to implementers

Post-conditions— what must be true about the system after a use case completes

Other requirements— what constraints must this use case conform to

Priority— how important is this use case?

Frequency— how often is this performed?

A Use Case Template

Use case name

Preamble

Use case body (narrative, scenario or conversation)

Supplementary details and constraints

Narrative Form

Free-form text in paragraph format

Describes the intent of the user in performing the use case

Describes high-level actions of the user during the use case

Refers to key concepts from the problem domain that are involved in the use case.

Make Clear What You Don’t Know

Write questions about unsolved issues

Put them with the appropriate use case description to show you’re not done

Example:

Should the credit check be performed after the Order is submitted or before?

What happens if credit is denied?

If you are unclear about a detail, don’t write fiction; it could become fixed

Avoid Vague Words

“Depends on,” in writing, is ambiguous

Example:

XYZ depends on the following software might mean:

- The following software must be complete before programmers at ABC can begin developing XYZ
- The following software produces data processed by XYZ
- The following software must be installed on any computer on which XYZ is to run

Writing a Use Case Narrative

Name the use case with an active verb phrase describing the user's goal

Write a paragraph explaining the user's intent, what should happen to achieve the goal, and some key facts about the process

Identify terms that should be defined

Annotate and reference other requirements that the use case satisfies

Tell this "story" from a single point of view (the user's)

Exercise

Write Use Case Narratives

Scenarios and Conversations: Tips and Guidelines

Write General and Specific Cases

Choose this option when your audience needs both general and specific usage descriptions

High-level use case names state a general goal. Write one narrative use case for each general goal:

Narrative: Make a payment

Describe what online payment means and typical ways of making them

Write scenarios or conversations that describe more specific goals:

Scenario 1: Make a **recurring** payment

All the steps in paying my monthly phone bill …

Scenario 2: Make a **non-recurring** payment

All the steps in paying a fixed amount …

Scenario 3: Make a **regular** payment

All the steps in paying a monthly loan …

Write Two “Versions” of the Same Use Case

Choose this option when some want a quick idea, while others want to see the details

First, write a narrative

Then, choose an appropriate form. Rewrite the use case body at this lower-level of detail

Leave the narrative as an overview

Consider adding an “overview” section to your template if you have always have diverse readers for your use case descriptions

Writing Scenarios and Conversations

Start by writing the success story, the “happy path”

Capture the actor’s intentions and responsibilities, from beginning to end goal

Define what information passes between the system and actor but don’t describe its format or details

Writing Scenarios and Conversations

All steps should be visible to or easily surmised by the actor

Resist the temptation to get too detailed

Convey how the system will work

Be clear on where to start

Describe how the goal is achieved

End there

Scenario

One particular path through a use case written from the actor’s point of view

Describes a sequence of events or list of steps to accomplish

Each step is a simple declarative statement with no branching

May describe:

- Actors and their intentions
- System responsibilities and actions

Record Issues

Clearly distinguish what you know from what you need to find out

Assign responsibility to a stakeholder for resolving an issue

Write and attach these to a specific description

Online Banking Scenario

Scenario: Register Customer with Auto-Activation

1 User enters registration information:

Required information: user name, email address, desired login ID and password, and confirmation password

One of: account number and challenge data, or ATM # and PIN

Optional: language choice and company

2 System checks that password matches confirmation password.

3 System validates required fields and verifies uniqueness of login ID

4 System verifies customer activation information.

5 System creates and activates customer online account.

6 System displays registration notification.

Include Actor Actions

Be explicit about what the actor does. Don’t disguise them as “system collects” or “system captures”actions

Actor actions disguised as system activities:

Scenario: Withdraw Fixed Cash Amount (Fast Cash)

1. Present transaction screen
2. Capture fast cash withdrawal request
3. Post transaction to bank and receive confirmation
4. Dispense money, card and transaction receipt

Fixed:

Scenario: Withdraw Fixed Cash Amount (Fast Cash)

1. ATM presents transaction screen
2. Customer selects “Fast Cash” option
3. ATM posts fast cash amount withdrawal transaction to bank and receives confirmation
4. ATM dispenses money, card and transaction receipt

Include System Actions

Be explicit about what the system does

No system behavior described:

Scenario: Withdraw Fixed Cash Amount (Fast Cash)

1. Customer selects “Fast Cash” option
2. Customer takes cash, card and receipt

Fixed:

Scenario: Withdraw Fixed Cash Amount (Fast Cash)

1. ATM presents transaction screen
2. Customer selects “Fast Cash” option
3. ATM posts fast cash amount withdrawal transaction to bank and receives confirmation
4. ATM dispenses money, card and transaction receipt

Describing Actions

Show actor intent, not precise movements

- Intention: User enters name and address
- Movements:
  - System asks for name
  - User enters name
  - System prompts for address
  - User enters address

Use simple grammar

- Subject…verb…direct object…prepositional phrase
- The system…deducts…the amount…from the account balance

Write actions that move the process forward

- “Validate that…,” don’t “Check whether”

Condense Information Entry and/or Validation Actions

List of Seemingly Unrelated Items:

Enter name

Optionally, enter address

Optionally, enter telephone number

Fixed:

Enter personal information (required: name; optional: address and phone number)

State System Actions At a Reasonably High Level

Includes Too Many Low Level Details and Substeps:

System opens connection to the bank
System requests authorization of bankcard number and PIN #
Bank confirms bankcard and PIN are valid
System requests active accounts for bankcard
Bank returns account list
System creates active online account entries for each account

Fixed:

System validates bankcard and PIN #s
System activates accounts associated with bankcard

Make sure what is going on, and why is it is being done is obvious to the typical reader. Know your audience

Showing Optional and Repeated Actions

Make clear whether a step is optional

Indent several optional steps

Optionally, select an available course section

In any order, do one or more of the following:

- eat
- drink
- make merry

Next ….

Merge cells to indicate the beginning and end of a block of repeated or optional actions

<table>
<tr><td colspan="2">Repeat</td></tr>
<tr><td colspan="2">actions go here</td></tr>
<tr><td colspan="2">Until proposed schedule is built</td></tr>
</table>

Writing a Scenario

Use a list

Record action steps

Record actor and system actions, identifying each

Scenario 1 Name

1. System does this first
2. Actor first does this
3. Actor next does this

Scenario 2 Name

1. System does this first

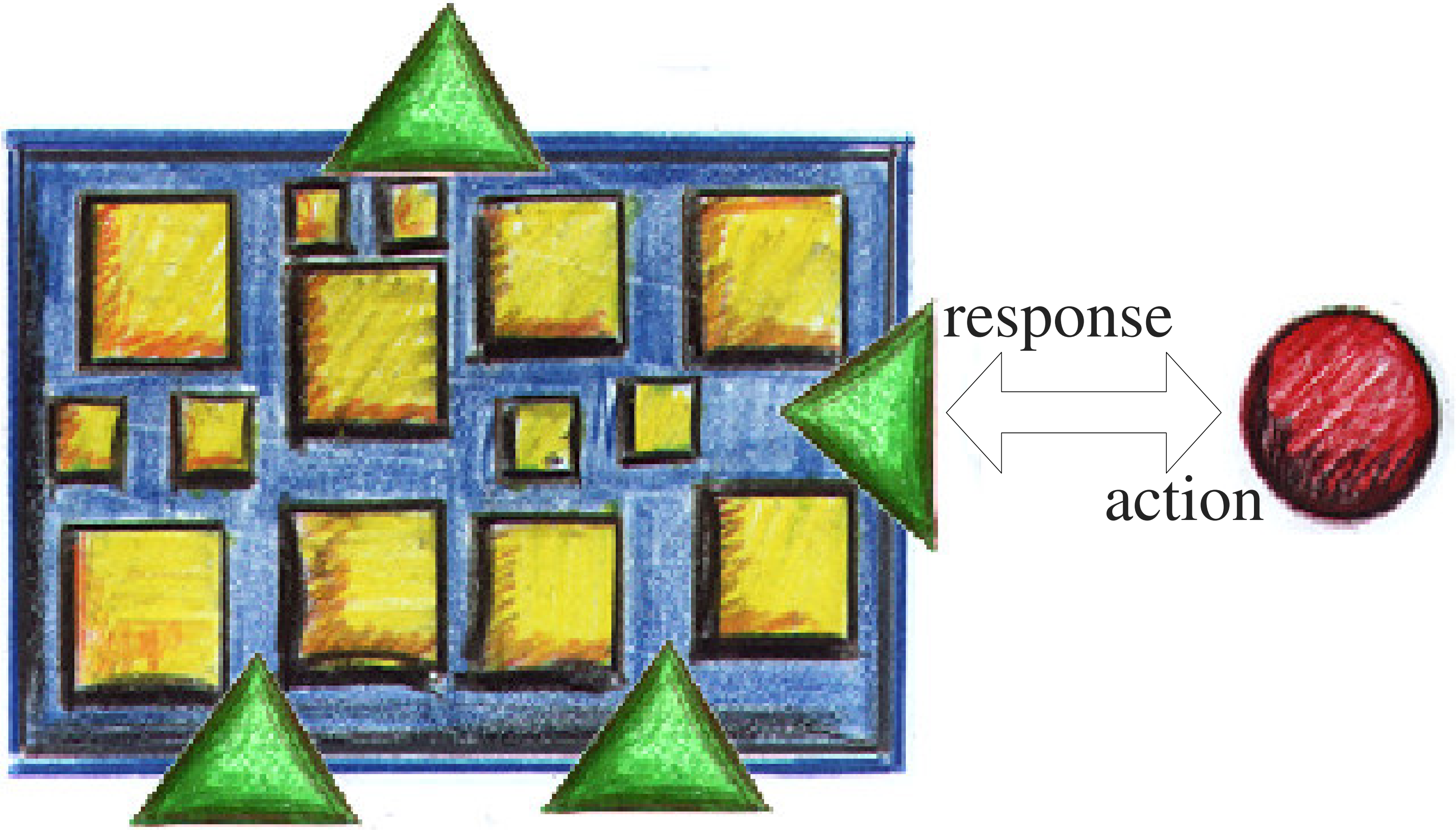
Actor:

  2. First does this
  3. And then does this

Exercise

“Clinic” a Scenario

Conversation



Conversation Form

One path through a use case that emphasizes interactions between an actor and the system

Can show optional and repeated actions

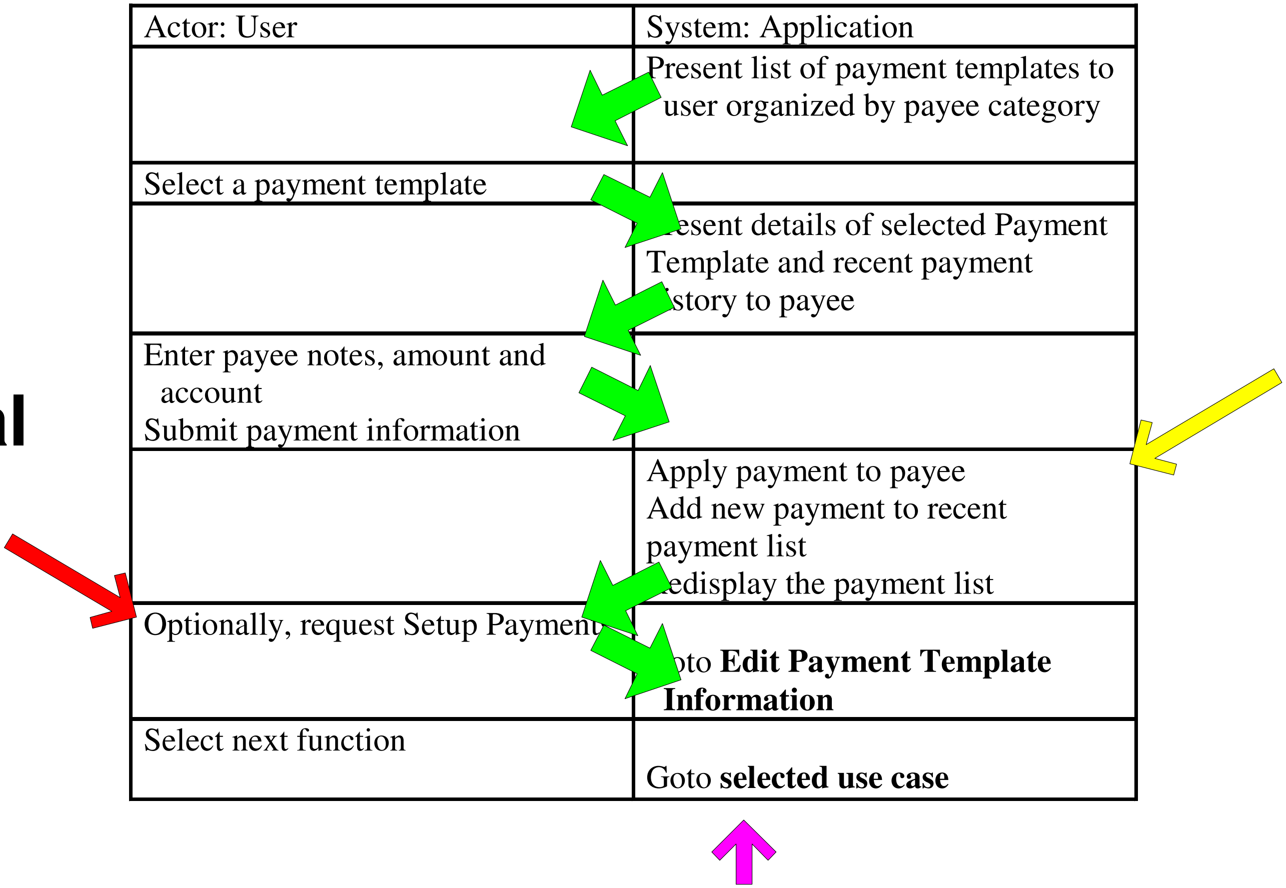
Each action can be described by one or more substeps

May describe:

- Actor actions
- System responsibilities and actions

Make a Payment Conversation

General Flow



Actor: User	System: Application
	Present list of payment templates to user organized by payee category
Select a payment template	
	Present details of selected Payment Template and recent payment history to payee
Enter payee notes, amount and account Submit payment information	
	Apply payment to payee Add new payment to recent payment list Redisplay the payment list
Optionally, request Setup Payment	Goto **Edit Payment Template Information**
Select next function	Goto **selected use case**

Optional Action

Multiple Actions

Invoking Another Use Case

Maintain a Consistent Level of Detail

Do not mix intent, action and detail in the same use case

Write at a level that seems appropriate to your readers

This typically means describing actions, not minute details

Description within a use case should be at the same level of abstraction (± one)

Maintain a Consistent Level of Detail

Mixed level of detail:

1 Check for required fields

2 Capture user ID and password
Ask security component for validation
Issue SQL statements to security database for logon authorization…

3 Open connection to bank server
Read account summaries…

Fixed:

1 Check for required fields

2 Login user to domain

3 Display account summaries and bulletin

Choosing Between Conversations and Scenarios

Use a scenario when:

- a simple list of actions is sufficient
- actor-system interactions aren't interesting

Use a conversation when:

- there are many interactions *and* you want to describe them
- you want to show more detailed system responses
- you want to separate the roles of actor and system

Don’t Embed Alternatives

Conversation: Registration with Automatic-Activation

	10. If bank supports automatic activation with ATM and PIN then... If ATM and PIN #s are valid then....

Fixed:

Conversation: Registration with Automatic-Activation

	10. Validate ATM and PIN #

Exception – Step 10: ATM and PIN #s are invalid— Report error to user

Leave Out Information Formats and Validation Rules

User Name: First name, last name (24 characters max, space delimited)

email address with embedded @ sign signifying break between user identification and domain name which includes domain and sub-domain names delimited by periods and ending in one of: gov, com, edu...

Fixed:

Required: user name, email address, desired login ID and password

One of: account number and challenge data, or ATM # and PIN

Optional: Company Name

Document information model details in a separate place!

Don’t Mention Objects in System Actions

Objects mentioned:

Create customer and account objects

Fixed:

Record customer account information

Remember who the readers are!

Leave Out Presentation Details

Widget details described:

- Display note in a read/write text field
- From account in a drop-down list box
- Amount in a currency field

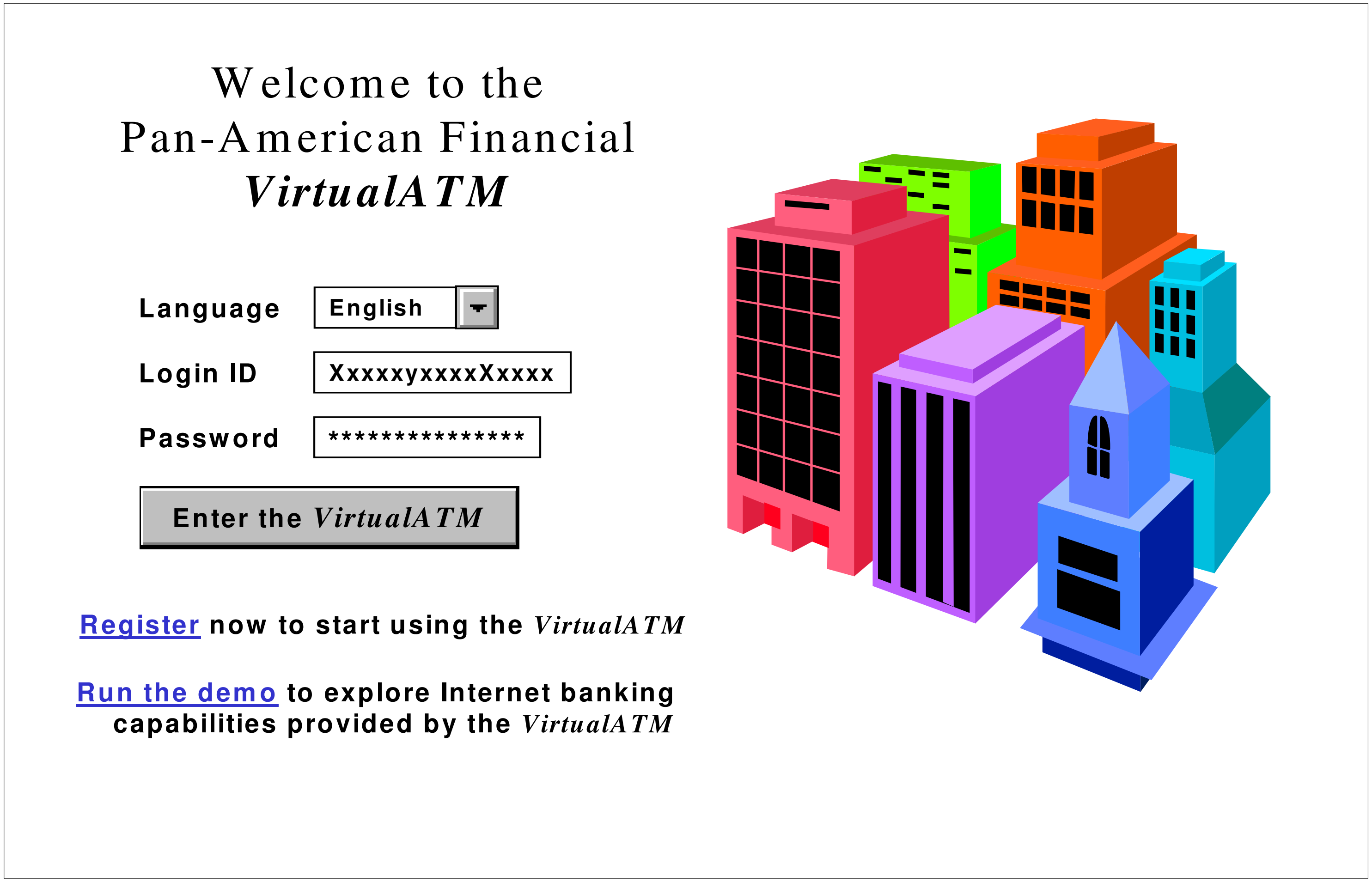
Fixed:

- Display payment template editable fields (note, from account, amount)

Reference screens used by a conversation

- Screens: See Login Page

User Interfaces Show a Different View



Welcome to the
Pan-American Financial
VirtualATM

Language: English

Login ID: XxxxxyxxxxXxxxx

Password: ***************

Register now to start using the *VirtualATM*

Run the demo to explore Internet banking capabilities provided by the *VirtualATM*

Writing a Conversation

Use a table

Separate actor actions from system responses

Record rounds between the actor and system

	Actor Actions	System Responses
Batch round		I do this
	And I respond by ..	
Interactive Round	I tell you this… …and this, too…	I am responding to what you are telling me and giving you feedback while you are talking

Showing More Detail

Describe what is done to accomplish the use case

- Basic functionality
- Variations
- Exceptional conditions
- Things that must be true before starting the use case
- Things that must be true on exiting the use case

Keep Rules in a “Policies” Section

Use Case: Register Customer

A new user must request access and gain approval in order to perform online banking functions. Registration can be done instantly, if the bank supports automatic activation, or the user can enter a request which will be approved by a bank agent.

Policies

Customer challenge data must be validated against customer account records before activating on-line access.

Use a Table for Complex Rules

Shipping Method	Shipping Time	Total price: add both columns	
		Per Shipment	Per Item
Standard Shipping	3 to 7 business days	$3.00 per shipment **plus**	$0.95 per book
2nd Day Air ***Note:*** *Not available to P.O. Boxes, the U.S. Virgin Islands, Guam, or APO/FPO addresses.*	2 business days	$6.00 per shipment **plus** add an additional $10 for AK, HI, PR, or American Samoa	$1.95 per book
Next Day Air ***Note:*** *Not available to P.O. Boxes, the U.S. Virgin Islands, Guam, or APO/FPO addresses.*	1 business day	$8.00 per shipment **plus** add an additional $15 for AK, HI, PR, or American Samoa	$2.95 per book

Document Global Requirements in a Central Place

Distinguish between system-wide requirements and those than span several use cases

- Example: System must run 7 by 24
- Example: Account information should be encrypted and transmitted over a secure connection

Reference those requirements that are satisfied by the use case below the use case body

Document Hints and Ideas

Design Notes

Errors and warnings about registration information contents should be collected and returned to the user in a detailed message rather than stopping at the first detectable error.

Payments should be shown in time order, with the current date first.

The user should not see payments that he should have visibility of. Prevent a user from seeing a payments from secret accounts that he should be unaware of.

Add design notes as they occur to you

Remove Clutter

Metatext— text that describes text that follows

The purpose of this use case is to describe how customers make payments.

Vague Generalities— well known principles

Each input screen shall fit entirely within the window and use as little scrolling space as possible.

Piling On— extra meaningless empty words, paragraphs, charts, sections, overbearing templates

Before piling on	**After**
Use Case	Business Use Case
Requirements	Requirements Specification Document

Exercise

Write a Conversation

Alternatives:
Exceptions and Variations

Alternative Paths

For each significant action:

Is there another significant way to accomplish it that could be taken at this point? **(Variation)**

Is there something that could go wrong? **(Exception)**

Choices for Describing Variations

Add textual descriptions of variations in the variations section of the use case template, which may reference an additional use case

or

Modify the body of the use case to show the variation, especially when you want to emphasis the variation, which may reference an additional use case

or

Draw an activity diagram that shows decision points, alternate paths, and parallel activities

Choices for Describing Exceptions

Add textual descriptions of exception in the exceptions section of the use case template, which may reference an additional use case

or

Draw an activity diagram that shows decision points, alternate paths, and parallel activities

Describing Exceptions Makes Requirements More Complete

Possibilities in *Place An Order*

Ideal situation (primary use case):

- Good credit, items in stock → accept order

Recoverable situations:

- Low credit and preferred customer → accept order
- Low stock, and OK to reduce quantity → accept reduced quantity order

Unrecoverable situations:

- Bad credit and not a preferred customer → decline order
- Out of stock → decline order

Exceptions Added to *Place An Order*

Scenario: Place An Order

1. Identify the customer
2. Identify each order item and quantity
3. System accepts and queues the order

Exceptions:

1a. Low credit and Preferred Customer:...

1b. Low credit and not Preferred Customer:...

2a. Low on stock and Customer accepts reduced amount:..

When to Create a New Use Case to Describe An Alternative

Write another...

- when an alternative appears complex
- when an alternative is important and you want to emphasize it

Document simpler alternatives in the supplementary part

Document more complex ones as separate use cases

Rewrite and reorganize for clarity!

Give new use cases specific names that identify specific conditions

Alternatives in *Registration w/ Auto Activation*

1. User enters registration information
2. System checks passwords match
3. System verifies login ID uniqueness

Variations :

1a. User enters ATM card # and PIN – see **Validate ATM card and PIN**

1b. User enters challenge data and account – see **Validate Challenge Data**

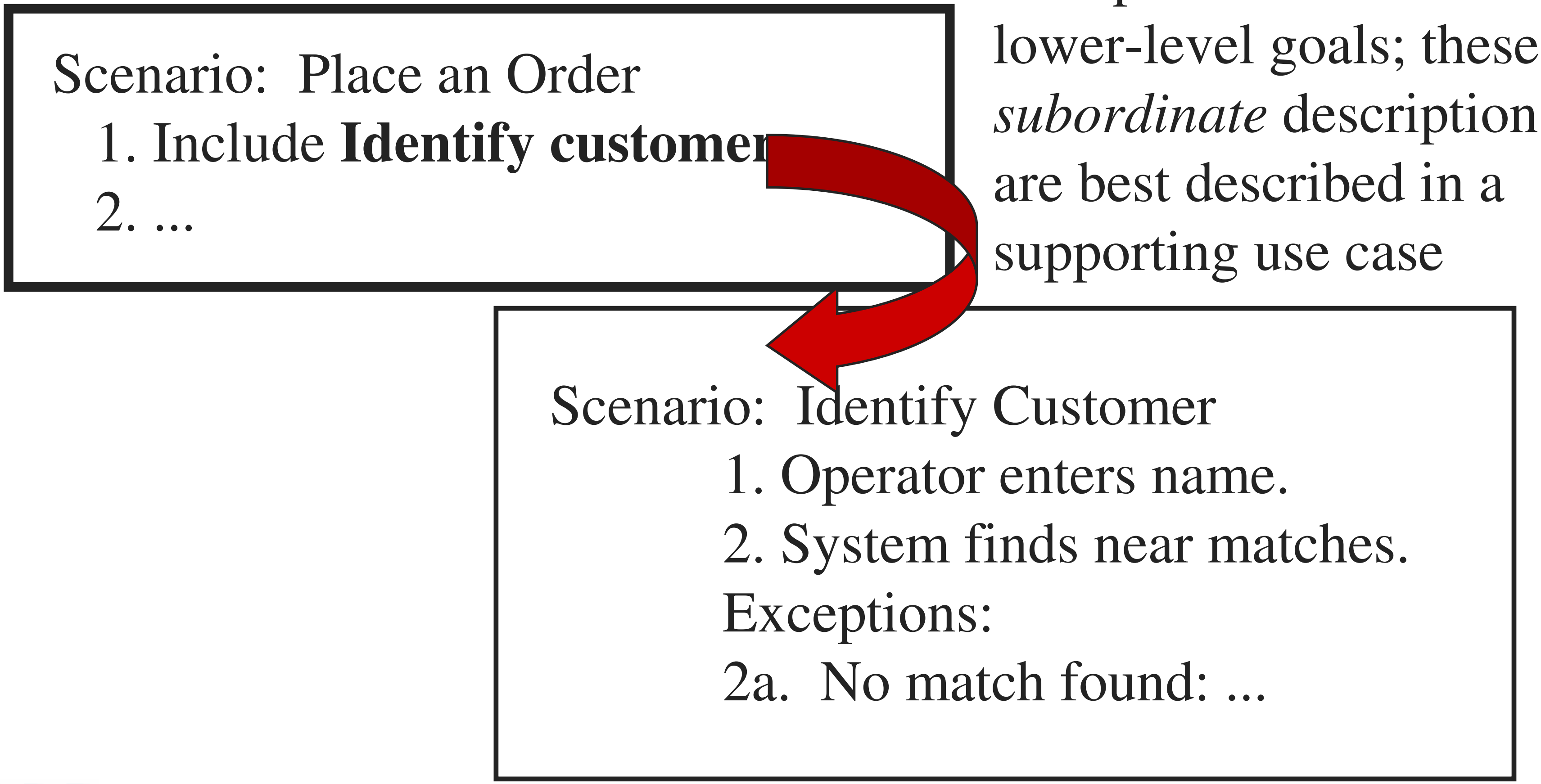
Exceptions:

2a. Report password mismatch and ask user to try again

2b. Third try – exit use case and report failure (unrecoverable)

3. Suggest unique alternative that user can accept or enter new choice

Keep Steps at Roughly the Same Level of Detail



Scenario: Place an Order

1. Include **Identify customer**
2. ...

Scenario: Identify Customer

1. Operator enters name.
2. System finds near matches.

Exceptions:

2a. No match found: ...

A step can refer to lower-level goals; these *subordinate* descriptions are best described in a supporting use case

Describe Exceptions at a High-Level

Write higher-level steps as if the supporting use case succeeds. Describe failure/recovery actions in an exception.

Scenario: Place an Order

1. Include **Identify customer**
2. ...

Exceptions:

1a. Customer not found:. . .

assumes success

does not care why it failed, only describes recovery or failure actions

Documenting Exceptions

Name the exception below the use case body

Tell what step it relates to

Tag an exception when it is unrecoverable. Describe what happens after it is detected, or

When an exception is recoverable, describe the steps the actor or system takes to recover

Document what happens:

- Choose an appropriate form
- Briefly describe what happens, or
- Refer to another use case describing the exception handling

Documenting Variations

Decide whether the variation should be described within the use case body or if it should be referenced below the use case body (consider emphasis)

Decide whether it needs a separate description

Document what happens. Either:

- Briefly describe the variation, or
- Refer to a new scenario or conversation that describes the variation in detail

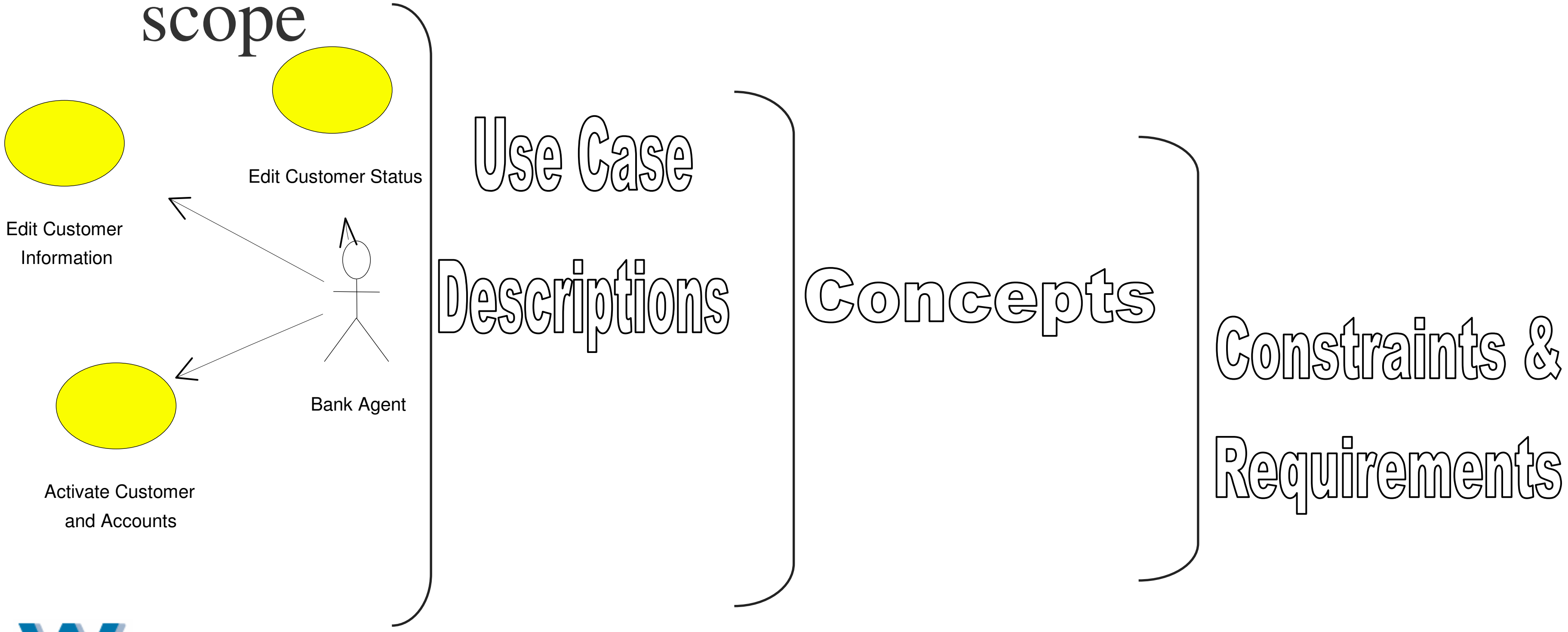
Exercise

Describing Alternatives

Use Case Models

Use Case Model

A Use Case Model includes structured use case descriptions that are grounded in well-defined concepts constrained by requirements and scope



Use Case Descriptions

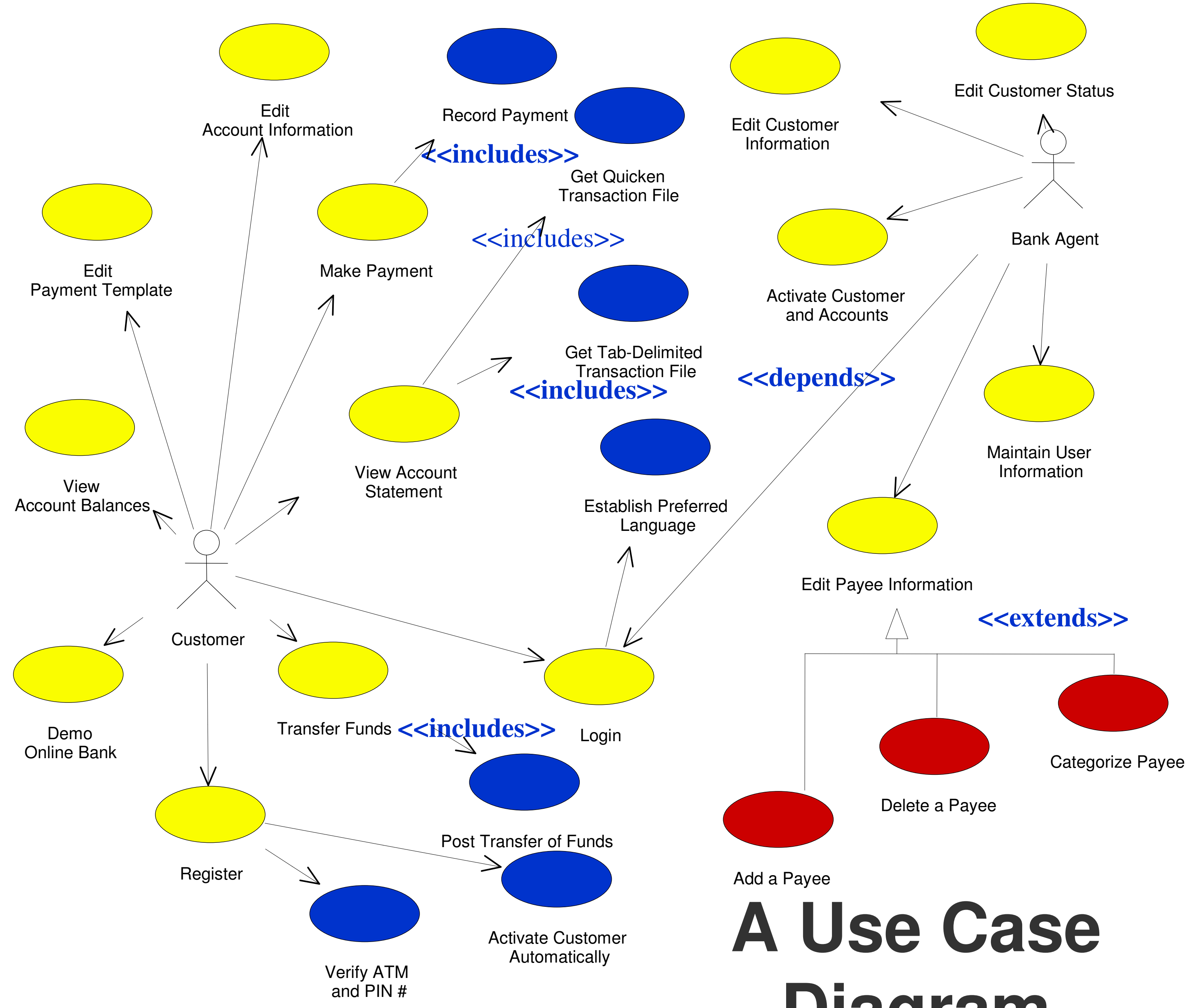
Concepts

Constraints & Requirements

Use Cases Can Be Related

UML defines these relationships between use cases:

- **Dependency—** The behavior of one use case is affected by another
  - Being logged into the system is a pre-condition to performing online transactions. **Make a Payment** depends on **Log In**
- **Includes—** One use case incorporates the behavior of another at a specific point
  - **Make a Payment** includes **Validate Funds Availability**
- **Extends—** One use case extends the behavior of another at a specified point
  - **Make a Recurring Payment** and **Make a Fixed Payment** both extend the **Make a Payment** use case
- **Generalize—** One use case inherits the behavior of another; it can be used interchangeably with its "parent" use case
  - **Check Password** and **Retinal Scan** generalize **Validate User**



A Use Case Diagram

Use Case Levels

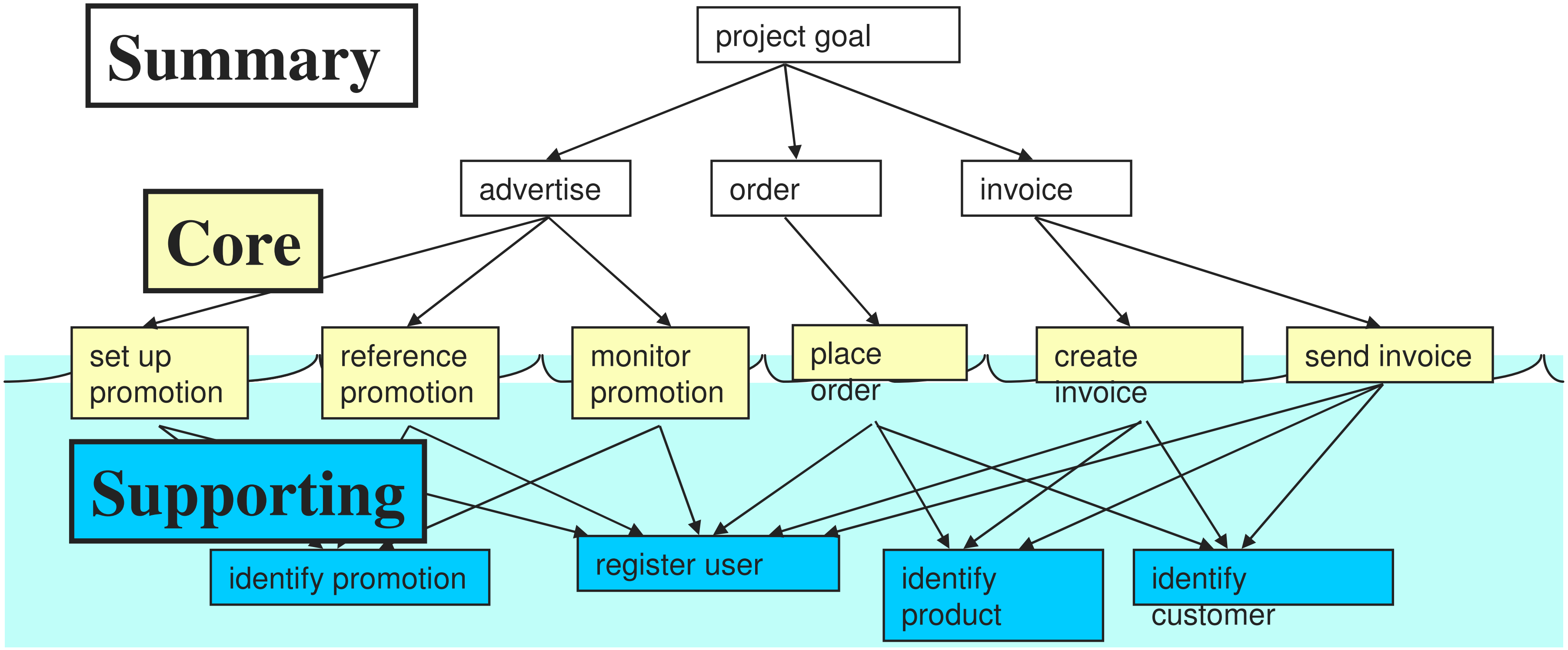
Use cases can be written at differing levels of abstraction and scope. Each serves a purpose:

- **Summary**— General descriptions and sweeping overviews of system functionality or business processes
- **Core**— Task-related descriptions of users and how they interact with the system; descriptions of a specific business process
- **Supporting**— Descriptions of lower-level activities used to complete subparts of a core use case
- **Internal**— Descriptions of the behaviors of and interactions between internal system components

Use Case Models Vary in Shape

Sailboat – balanced use cases

Classical business functions

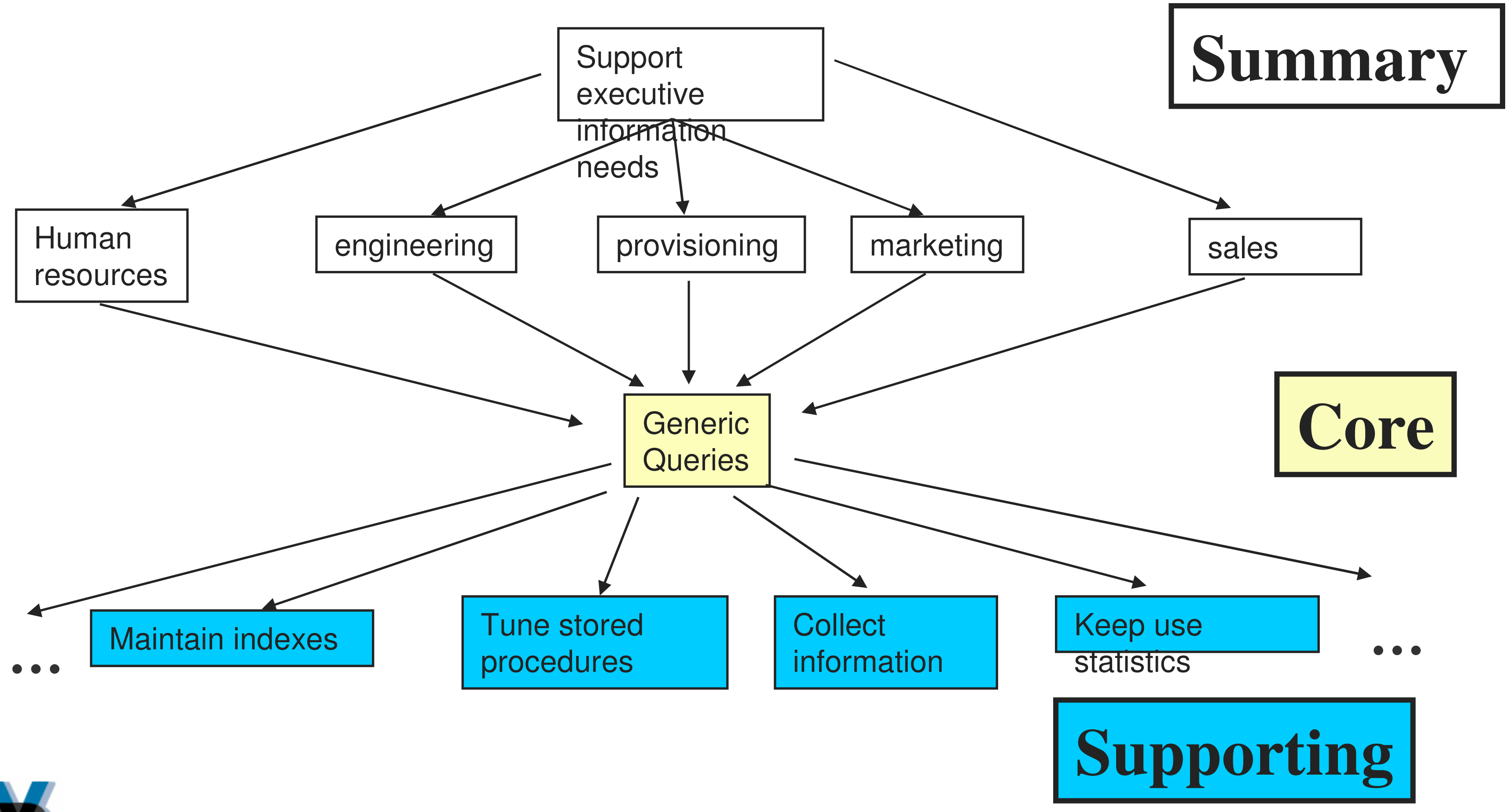


Alistair Cockburn, *Humans and Technology*

Use Case Models Vary in Shape

Hourglass—small core

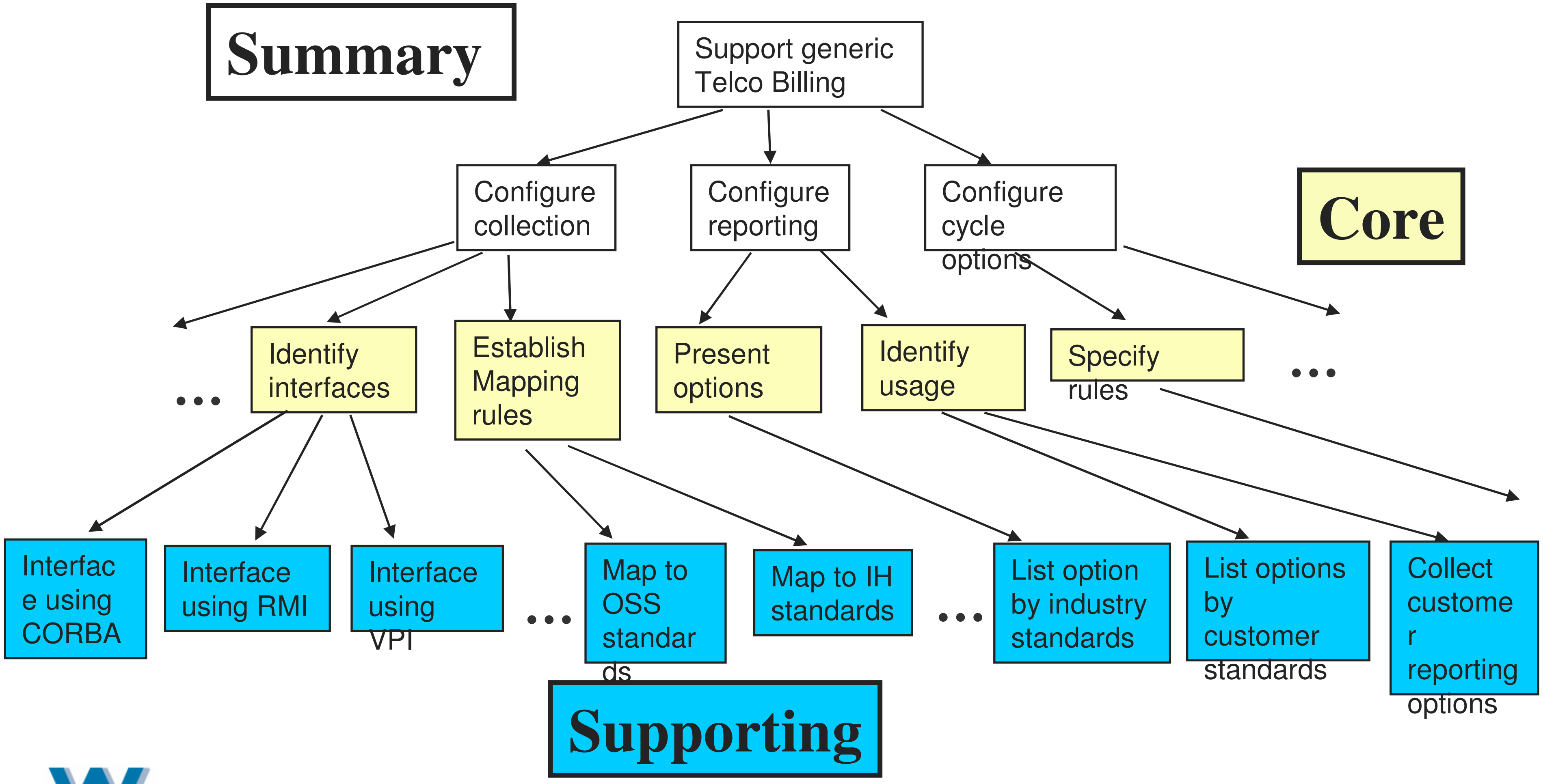
Ad hoc information query/data warehousing



Use Case Models Vary in Shape

Pyramid—supporting use case rich

Software application development environment



Emphasize What’s Important Within a Use Case

Things gain prominence by their position and appearance. To increase an item’s emphasis:

Put it first

Highlight it

Surround it by white space

- Put it in a bulleted list

Mention it in multiple places

Give it more room

Repeat or restate it in different forms

Say it another way

Mention it in multiple places

What’s Emphasized?

Template 1

Use Case: Make a Payment

Author: Rebecca

Last Revision Date: 9/11/01

Version: 0.4

Status: Preliminary Review

Level: Summary

Template 2

Use Case: **Make a Payment**

Actor: Bank Customer

Pre-condition: User has an active account and is authorized to transfer funds

What's Emphasized?

Choose course by optionally, in any sequence:

- Include **Browse Course Catalog**
- Include **Choose Next Course from Degree Plan**
- Enter course section

Emphasize What’s Important Within a Use Case Model

Place first those use cases you wish to emphasize

Choose the form of use case descriptions according to what you want to emphasize:

- A conversation emphasize the dialog between system and actor
- A narrative emphasizes the high points of a story, not the details

Repeat and restate things to make them stand out

Choose a template that doesn’t inadvertently emphasizes the wrong things

A Use Case Writing Process

Full Team	Small Teams or Individuals	The Products
Align on scope, level of abstraction, actors, goals, point-of-view		Actors, Candidate Summary Use Case Names
	Write summary descriptions	Narratives
Collect and clinic, brainstorm key use cases		Candidate Core Use Case Names
	Write detailed descriptions	Scenarios OR conversations
Collect and clinic, identify gaps and inconsistencies		Potential new Use Cases
	Revise and add precision	Revised Use Cases with Supplementary Details

Organize Your Use Case Descriptions

Choose an organization for your use cases

- by level (summary first, core next, supporting, then internal ones last)
- by actor
- by type of task
- arranged in a workflow

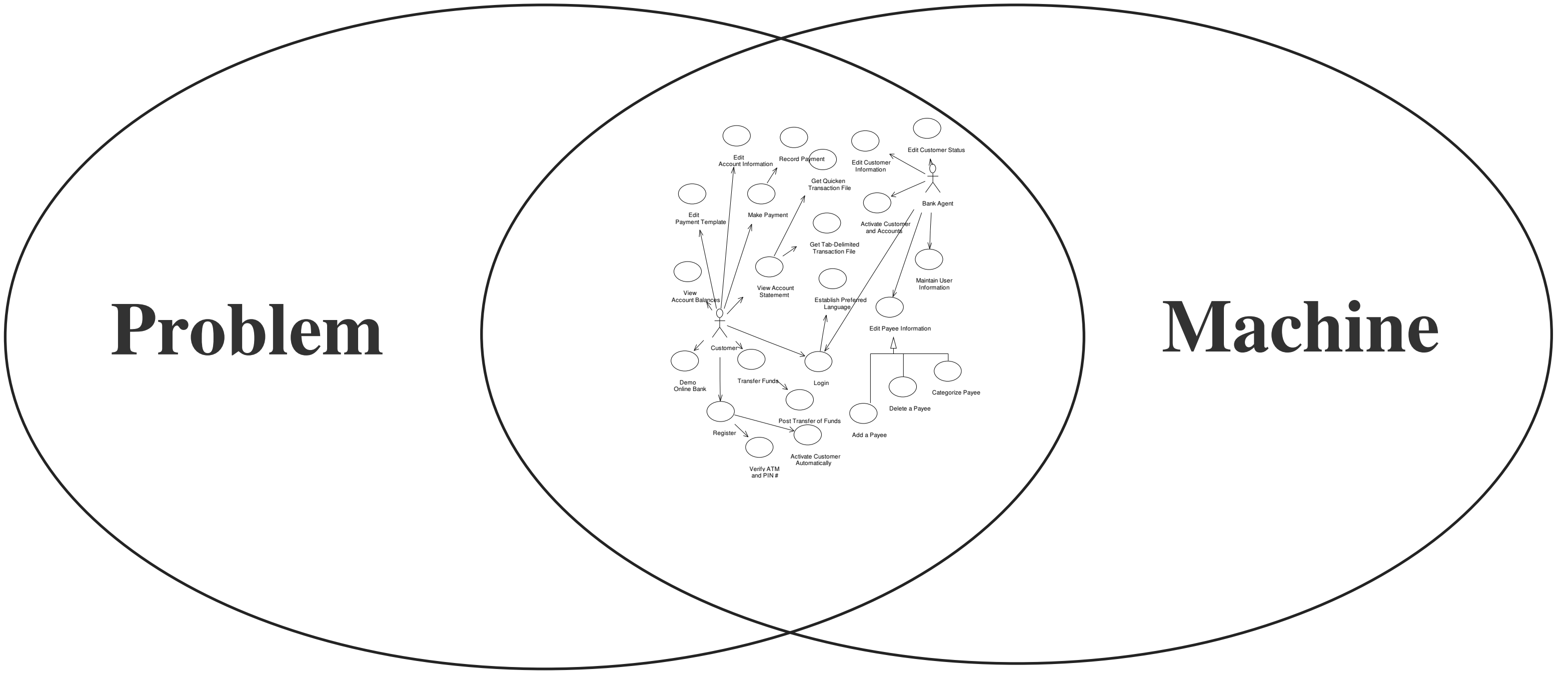
Be consistent. Keep various forms of a single use case together

Use Case Model Review Checklist

- Check for internal consistency between use cases
- Identify “central” use cases
- Identify unmet or externally satisfied preconditions
- Review the actor’s view for completeness
- Review the handling of exceptions
- Document use case dependencies, extensions and includes relationships
- Document external dependencies

Two Worlds, Three Descriptions

Solving a Problem With a Machine



Problem

Machine

Requirements

Specification

Product

THE ART OF WRITING USE CASES TUTORIAL NOTES

I. Description and Objectives

This is an introduction to use cases, a technique for structuring system usage descriptions, and the principles of a user-oriented development process. You will be able to apply the principles and techniques to your projects, writing appropriate usage descriptions.

The topics include:
III. The context for use cases
IV. Use case modeling constructs
V. System glossary
VI. A Use Case Template
VII. Narratives
VIII. Scenarios and Conversations
IX. Other Descriptions, Exceptions, and Variations
X. A Use Case Model Checklist
XI. The Writing Process
XII. More Tips and Techniques

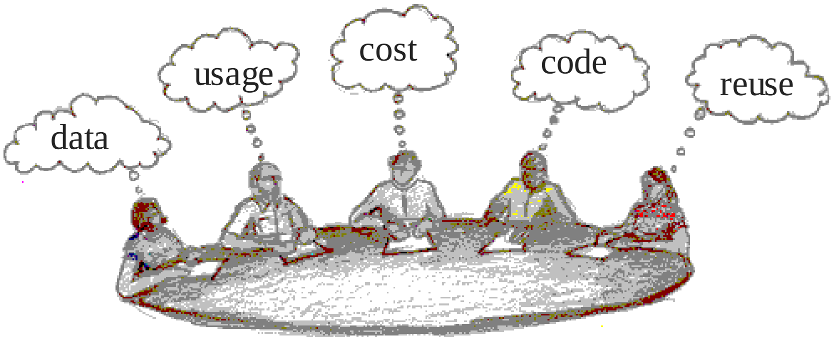
II. Further Resources

There are several good books about use cases. We recommend these three:
Writing Effective Use Case, Alistair Cockburn, Addison-Wesley, 2001, ISBN 0-201-70225-8
Use Cases Requirements in Context, Daryl Kulak and Eamonn Guiney, Addison-Wesley, 2000, ISBN 0-201-657678-8
Software for Use A Practical Guide to the Models and Methods of Usage-Centered Design, Larry Constantine and Lucy Lockwood, ACM Press,1999,ISBN 0-201-92478

Andy Pol’ s website, The Use Case Zone has many pointers to online articles, templates and use case discussions: http://www.pols.co.uk/usecasezone/

III. The Context for Use Cases: Team Development

Development is never done in a vacuum; there is always a context. Many of the stakeholders in our development efforts do not speak in our native object-oriented tongue. In our role of analyst we face two challenges: correctly interpreting stakeholders' knowledge of the problem, their concerns and requirements in our models, and presenting our design work in terms they can understand.

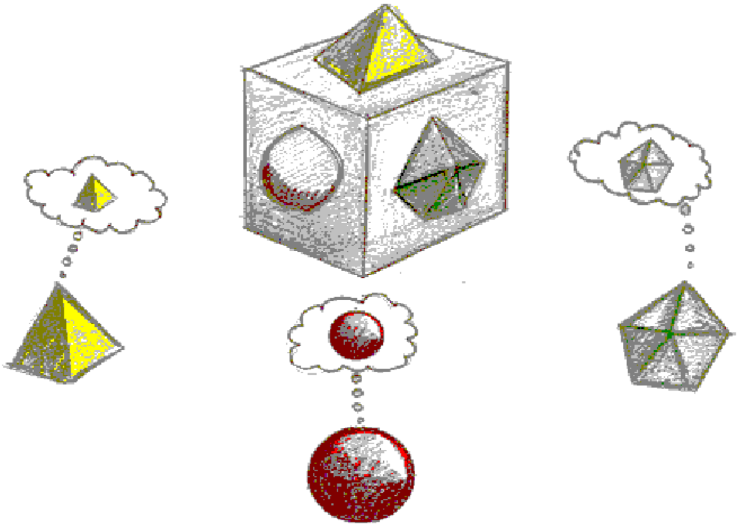


A good system never dies, it is adapted and improved upon.

A system takes form through a series of textual and graphical descriptions. At each time-slice of the project, the description should be less ambiguous, but each form should be describing the same thing. Each description, when viewed by a practitioner with experience in the corresponding natural, graphical, or programming language, can be evaluated according to a number of well-known criteria. Typically, the system came to be through a structured process known as design, often preceded by a form of requirements gathering and specification called analysis. During analysis, one of your tasks is to describe our system' s usage with use cases.

Each participant in the life of a software system has a unique set of criteria for evaluating its quality during its development. The target values that are used during such evaluation varies according to their point-of-view. To begin simply, let's imagine sets of criteria that are important from three points-of-view:

- user
- analyst/designer
- programmer



To bring together all of these perspectives, you need a systematic way to consider the problem. Once you can agree on the nature and requirements of the problem, you can make informed decisions about and document which parts of the problem you intend to automate with the computer. Finally, you will have solid ground for determining whether or not the program that you build for our machine has, in fact, accomplished your goals.

The User

The user is particularly concerned that the system be easy-to-use. Of course, this requires that the application controls and processing be transparent, consistent, correct and natural to the user. The system must also do the job, i.e., it must be complete, and it should be configurable to an individual user's specific needs.

The Analyst/Designer

From the analyst/designer’ s point-of-view, the requirements, the specification and design must be simple and easy to understand. It must be modular and traceable to the requirements. Due to an ever-evolving specification, it must be flexible and extensible. Specified portions must be reusable. Further, the system under development is constrained by business and user requirements. The functional characteristics of the design should be concise without losing the details of its execution behavior.

The Programmer

Programmers have all the issues of the designer. But when entering the implementation domain, they must be certain that the application is possible. Beyond that, they must live with the constraints imposed by the hardware that application performs on.

Building Consensus

System development has three areas of activity: understanding and documenting the problem and its requirements, specifying how the various users will be able to use the system to satisfy the requirements and how the system will fulfill all of the remaining non-usage requirements, and implementing the specification as software executing on appropriate hardware.

IV. Use Case Modeling Concepts

A specification is a statement of what the system is to do in the context of your problem. It describes how the requirements that you have elicited by asking the right probing questions will be fulfilled. Requirements that can be satisfied by interactions between a user and the program can be described by use cases.Use cases present a model of how your system is used and viewed by its users. This use case model is just one view developers need to understand as they proceed with design and implementation. It is also a view that other stakeholders in the specification of a product can readily understand and comment on. A usage model, expressed as use cases forms the basis for a behavioral description of a system.

Let’s introduce the core concepts of use case models:

Use Case

A *use case* is a description of system functionality from a particular point-of-view. Many use cases describe task-related activities. For example, in the Online Bank application, which we draw upon to illustrate concepts in this course, we wrote use cases to describe these activities, among others

- making a payment
- transferring funds between accounts
- reviewing account balances

Each use case describes a discrete “chunk” of the online banking system. These use cases were described from the users’ viewpoint.

Use cases don’ t dive into implementation details, but describe how something behaves from an external perspective. A use case may include more or less detail, depending on its intended audience and what level it of the system it describes.

Three Use Case Forms

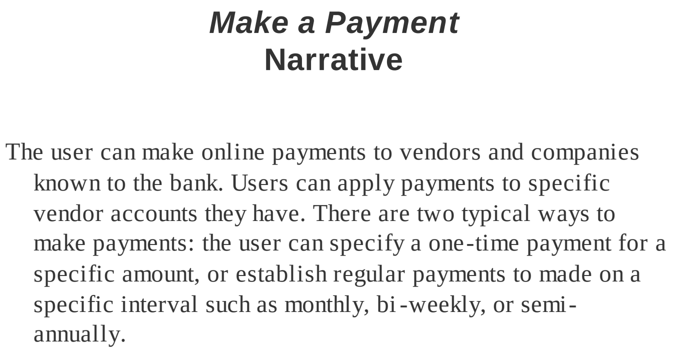
We recommend you consider three forms of use case descriptions. Each different form has its strengths and weaknesses. Depending on what you need to describe, and at what level of detail, you should pick the appropriate form to write a use case description. You might choose to first write high-level overviews, then add detail and describe the sequences of actions and interactions between the user and the program. The form you choose depends on what you are trying to express.

You may write one or more forms for each use case, depending on the needs of your audience. Write narratives to present a high-level overview. Then, if appropriate, write one or more scenarios or conversations that elaborate this high-level description.

The Writing Task	The Best Form To Use
Present Overview	**Narrative**
Describe simple sequence of events	**Scenario**
Emphasis actor-system interaction	**Conversation or Essential Use Case**

Here are examples of each of the three forms.

First, a use case narrative taken from an on-line banking project:

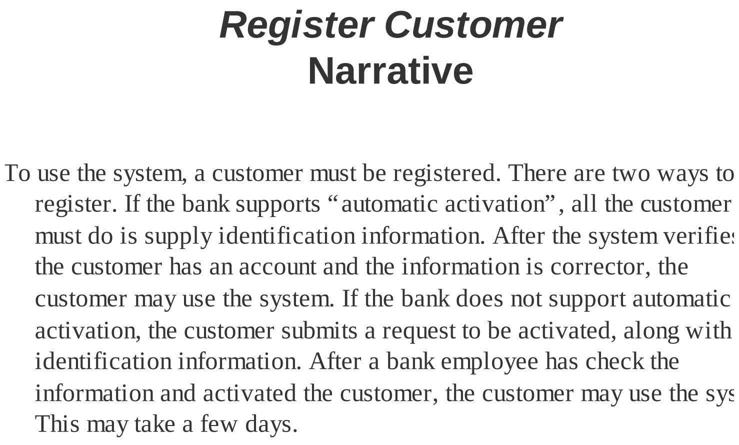


Make a Payment
Narrative

The user can make online payments to vendors and companies known to the bank. Users can apply payments to specific vendor accounts they have. There are two typical ways to make payments: the user can specify a one-time payment for a specific amount, or establish regular payments to made on a specific interval such as monthly, bi-weekly, or semi-annually.

It offers a high-level view of how the requirements of “Make a Payment” are satisfied.

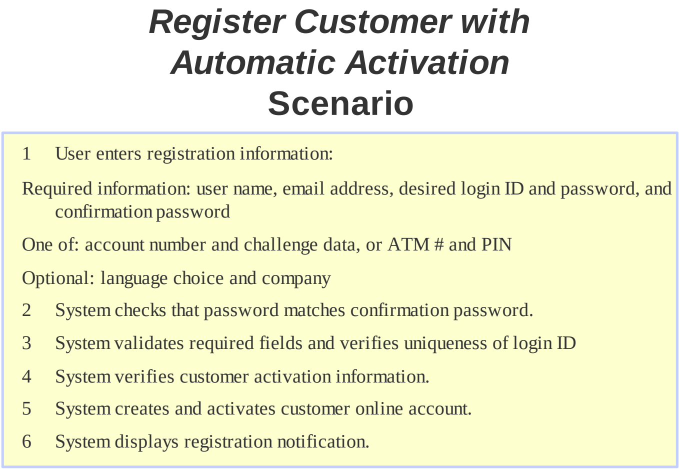
A use case narrative has a very simple format. It begins with the name of the use case, and is followed by a brief, textual description that explains at a high level how an actor interacts with our system in order to accomplish a task or goal. Here is another narrative:



Register Customer
Narrative

To use the system, a customer must be registered. There are two ways to register. If the bank supports “automatic activation”, all the customer must do is supply identification information. After the system verifie the customer has an account and the information is corrector, the customer may use the system. If the bank does not support automatic activation, the customer submits a request to be activated, along with identification information. After a bank employee has check the information and activated the customer, the customer may use the sys This may take a few days.

There are two scenarios outlined in the narrative: one for automatic activation, another with manual activation. We write a sequence of actions to describe each. Here is an example of the scenario for registering with automatic activation.



Register Customer with Automatic Activation Scenario

1 User enters registration information:

Required information: user name, email address, desired login ID and password, and confirmation password

One of: account number and challenge data, or ATM # and PIN

Optional: language choice and company

2 System checks that password matches confirmation password.

3 System validates required fields and verifies uniqueness of login ID

4 System verifies customer activation information.

5 System creates and activates customer online account.

6 System displays registration notification.

Notice that, along with the sequence of actions, we include some notion of the types of information that are used.

Finally, the more detailed *conversation* form allows us to clearly show the system' s responses to the actions of the user. Here we have many opportunities to demonstrate decision-making, iteration, and dependency among the parts of the problem.

Actor: User	System: Application
	Present list of payment templates to user organized by payee category
Select a payment template	
	Present details of selected Payment Template and Recent payment history to Payee
Enter payee notes, amount and account Submit Payment Information	
	Apply payment to payee Add new payment to recent payment list Redisplay the payment list
Optionally, request Setup Payments	
	Goto **Edit Payment Template Information**
Select next function	
	Goto **selected use case**

Each form has its strengths and weaknesses. Conversations show more detail, scenarios show step-by-step sequences, narratives are free-form text. The form you choose depends on what you want to convey to your reader, and how much detail you want to show.

Form	Strengths	Weaknesses
Narrative	• Good for high-level summaries • Can be written to be implementation independent	• Easy to write at too high or too low a level • Not suitable for complex descriptions • Can be ambiguous about who does what
Scenario	• Good for step-by-step sequences	• Hard to show parallelism or arbitrary ordering • Can be monotonous
Conversation	• Good for seeing actor-system interactions • Can show parallel and optional actions	• Easy to write at too detailed level: pseudo pseudo-code • Only two columns: What about multiple actors?
All Forms	• Informal	• Informal

Abstraction, Scope, and Detail

Use cases can be written very concretely, or they can generalize specific actions to cover broader situations. For example, we could write use cases that describe:

> Steve registers for English 101, or
> Student registers for Course, or
> User uses System, or
> Student registers for Variable Credit Course, or
> Student Registers for Music Course

In order to choose the right level of abstraction to write a use case, you need to understand how the behaviors of both the actor and the system might be expressed to cover the widest range of situations without losing any important details. Clearly, "User uses System" is too high-level, and "Steve registers for English 101" is too concrete. However, it may be important to write use case descriptions for "Student registers for Course" and, if the system's or user's actions are sufficiently different, to also describe "Student registers for Variable Credit Course." In fact, if registering for a music course means signing up for practice sessions in practice rooms in addition to classroom instructions, it too may need additional description. You can also express variations within a single use case description.

Use cases vary in scope and detail. You can use them to describe all or part of our "system". Which system boundary do we mean: At a particular component (describing the web applet)? across the application (on-line banking)? or across multiple applications within the organization (the bank)?

We typically start by describing application level scope. The amount of detail that we choose to put into use cases varies. We could describe general actions: *Enter deposit amount*. Or specific detail: *Press number keys followed by enter key*

Write at the level that seems appropriate to your readers. This typically means describing actor actions and system responses that match the goal for the use case. So, to follow that guideline, if the use case were named "Make Deposit," we'd describe the user general action of "enter deposit amount," not his or her gestures: "Press number keys followed by enter key."

Recipe: Finding the Use Cases

1. Describe the functions that the users will want from the system.
2. Describe the operations that create, read, update, or delete information that the system requires. Describe these operations.
3. Describe how actors are notified of changes to the internal state of the system.

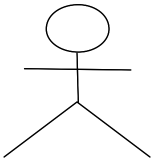
Identify actors that inform the system about events that the system must know about. Describe how the users will communicate the information about these events.

Actors

An *actor* is some one or some thing that interacts with our system. We divide actors in to two groups:

- those that stimulate the system to react (primary actors), and
- those that respond to the system' s requests (secondary actor)

We model actors so we can understand what behaviors they' ll need from our system, if they are primary actors. We model secondary actors so we understand how our system uses external resources. In the Unified Modeling Language, the stick figure icon is how we show an actor on a use case diagram. This is the standard notation for an actor, although you may choose another icon that is more meaningful.



Actors are the external agents that use (or are used by) our system. Those that initiate activity are worth considering as a group. These *primary* actors stimulate the system to take action and form the basis of most of our usage descriptions. The other, *secondary* actors, interact with the system only because the system poses questions to them or issues a command. They are usually external programs or devices, although sometimes the system will direct a human to perform a task.

Most often, systems engage with an actor called the user. In fact, we often unconsciously equate an actor with this user. But such a narrow vision will often make us overlook significant areas of the system' s requirements. For example, many systems require support for administrators and technicians that periodically maintain and configure the system. These activities are quite different from the user's tasks. Systematic consideration of the various actors that are involved with our system will ensure a more complete understanding of what it must do.

Guidelines for Finding and Naming Actors

GUIDELINE: Focus on primary actors.

In the on-line banking system, we have a number of human actors. The one we initially focused on was the customer-user who accesses financial services including bill payment, account balance and statement inquiries, and funds transfer. An agent of the bank (or bank agent) can perform several tasks: customer maintenance, console operation, and bank administrative functions which include bank agent maintenance, and system configuration.

GUIDELINE: Group individuals according to their common use of the system. Identify the roles they take on when they use or are used by the system

Each role is a potential actor

Name each role and define its distinguishing characteristics. Add these definitions to your glossary

GUIDELINE: Focus initially on human actors. Ask:

- Who uses the system?
- Who installs the system?
- Who starts up the system?
- Who maintains the system?
- Who shuts down the system?
- Who gets information from the system?
- Who provides information to the system?

GUIDELINE: Name human actors by their role.

Specific people may play several roles; several actors may represent them. We could divide our bank agent actor into several, more distinct roles: console operator, bank agent administrator, customer administrator, and system configuration manager. These finer distinctions, while easily made, didn't really help us gain any new insights about system requirements for bank employee usage. While important, bank agent usage wasn't a high priority. The customer-user facilities were of primary interest to the project manager and sponsors. So we backed off and did not enumerate these kinds of bank agent actors.

GUIDELINE: Don't equate a job title with an actor name.

This wasn't a problem on the online banking application. Since we didn't directly interact with bank employees we didn't know their job titles. We were arm's length from end users, so it was easy for us to create a single bank agent category. However, we have seen several projects where jobs and titles get in the way of understanding of how users need to use the system. Supervisory job titles don't always equate with more features; usage often cuts across job function.

GUIDELINE: Don't waste time debating actor names.

Actor names should be nouns or noun phrases. Don't be too low level when naming actors. Don't be too abstract in describing or naming an actor. We didn't have the benefit (or bias) of knowing the name of any existing legacy applications at banks. The physical name of the transaction service, e.g. CICS, seemed too physical and not very descriptive; our next line of thought was that this actor represented our connection to existing legacy applications. So, we settled on calling this external actor a legacy connection and left it at that!

GUIDELINE: Be consistent in showing actors. Your choices are:

- Show *all actors that interact with the system*, even remote systems,
- Show only those *initiators of the contact*,
- Show only those actors that *need the use case*,
- Show only *human* actors, not the system

We recommend you use the first strategy, and distinguish actors that initiate contact as *primary actors*, and actors that the system touches as *secondary actors*.

GUIDELINE: An actor name for an existing system should refer to its common name.

GUIDELINE: Names of non-human actors are more recognizable if they simply remain the name of the system.

Don't invent clever, more abstract names if it causes confusion. In the online banking application it was fairly easy to find our non-human actors. We recorded information about On-line Banking System customers and their transactions in an Oracle Database, and accessed legacy systems (either CICS, IMS/DC) to perform financial transactions and pull current account information. This led us to two external actors: legacy connection and database. These actors mainly were of interest to the development team who needed to model objects that represented interfaces to these external actors; the project sponsor only cared about the kind of legacy connections that would be supported, and that Oracle was the database we had selected.

GUIDELINE: If you are building a system whose behaviors are based on privileges and rights of individuals rather than on their roles, record these variations in a manner that lets you track their impact on the design - don't try to solve it with actors alone.

Sometimes we need to know more about individual users than their actor roles. You may need to describe individuals' rights and capabilities, and note what privileges are required to exercise certain system functionality. Simply defining actors doesn't buy us enough information. This issue came up in our system design when we started considering version two On-line Banking System features. In release one, a customer-user could register and use all banking functions; in version two, a major requirement was that multiple users could be associated with a single customer. Each user might use a different set of the customer's accounts. A user could grant account visibility if he/she had appropriate privileges (the ability to do account and user maintenance). Initially, we debated splitting customer-user into primary- user and customer-user, but talked ourselves out of creating a new actor to solve our conceptual problem. It wasn't clear that 'primary user' was the right distinction. One clue was that our domain expert didn't like this idea at all. He felt that since all customer-users had the potential to do account and user maintenance, they shouldn't be arbitrarily divided into different actors. We also realized that our second attempt at factoring bank agent into roles hid the requirement that our system needed to let banks configure the capabilities of individual bank agents. Those that were trained in customer administration weren't likely to also perform console operator functions, but it was up to the bank to decide who could do what; it was up to our system to enforce and grant these capabilities.

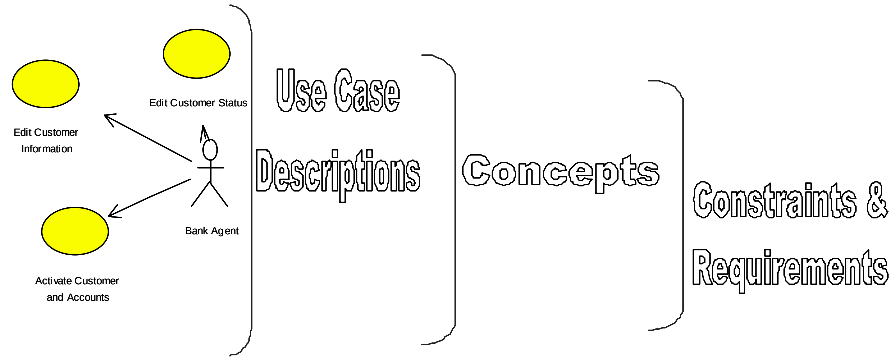
These activities are quite different from the user's tasks. Systematic consideration of the various actors that are involved with our software will ensure a more complete understanding of what the software must do.

Recipe: Finding Actors

1. Focus initially on human and other primary actors.
2. Group individuals according to their common tasks and use of the system.
3. Name and define their common role.
4. Identify systems that initiate interaction with the system.
5. Identify other systems used to accomplish the system’s tasks (these are secondary actors).
6. Use common names for these other “system” actors.

Use Case Models

A single use case describes a discrete *chunk* of the system’ s functionality. A *use case model* is a collection of related descriptions of our system’ s behavior. These descriptions are backed up by clearly understood concepts, and should satisfy system requirements.Use case descriptions are typically written from an external perspective; that of a user performing task-related activities. These descriptions form the basis of our view of how the various *actors* in our problem will interact with the program and flesh out one of our perspectives of the specification.


Edit Customer Status
Edit Customer Information
Bank Agent
Activate Customer and Accounts
Use Case Descriptions
Concepts
Constraints & Requirements

While you initially focus on use cases initiated by human actors, there are a number of other system initiated use cases that can be documented, such as:

- initializing the system on startup
- broadcasting change information to other active components
- backing up the database

Use Cases Can Be Related

Use case diagrams can show a big picture of the application by demonstrating what actors participate in what use cases, and by showing the relationships among the various use cases. Relations like “uses”, “depends”, and “extends” are added when this additional level of detail provides useful information.

GUIDELINE: Don’t show everything!

GUIDELINE: You can have more than one system view. Don’t try to put all of your useful information into one diagram.

The Unified Modeling Language defines these relationships between use cases:

Dependency—The behavior of one use case is affected by another

Being logged into the system is a pre-condition to performing online transactions. Make a Payment depends on Log In

Includes—One use case incorporates the behavior of another at a specific point

Make a Payment includes Validate Funds Availability

Extends—One use case extends the behavior of another at a specified point

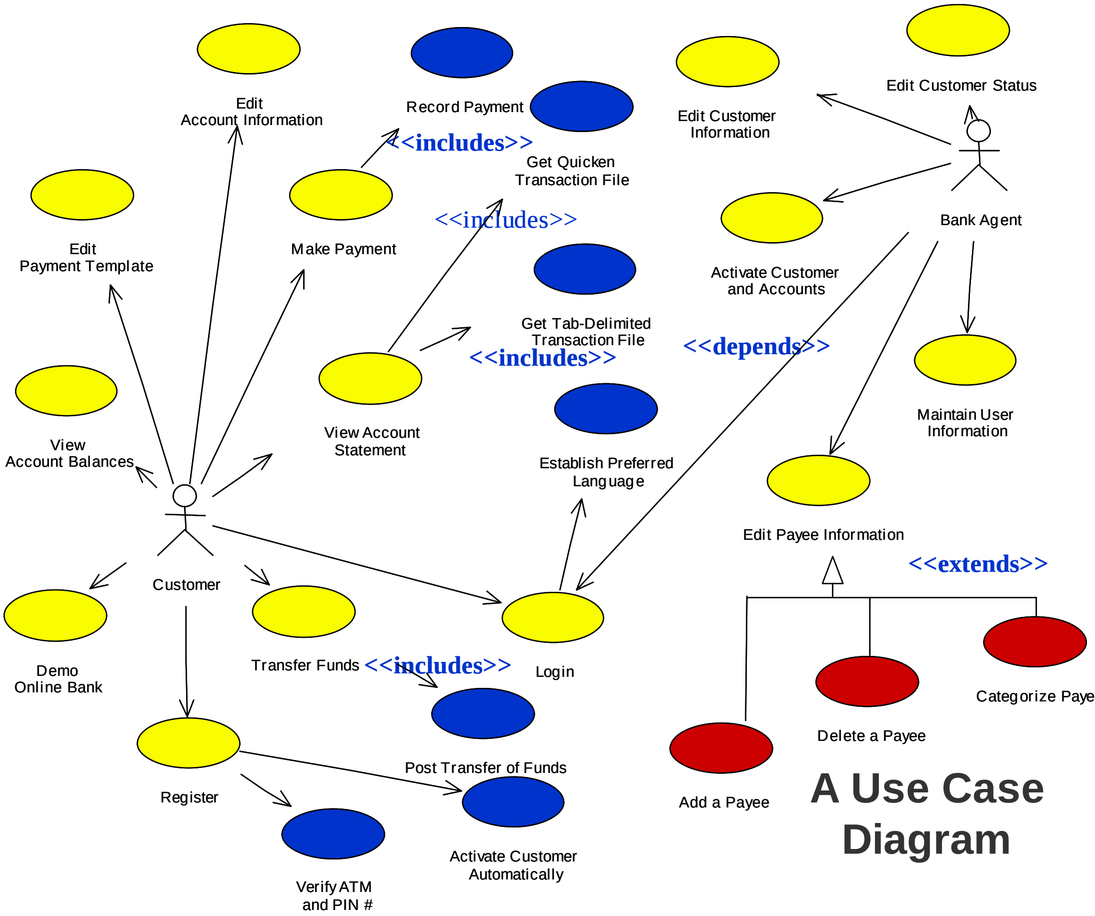
Make a Recurring Payment and Make a Fixed Payment both extend the Make a Payment use case

Generalize—One use case inherits the behavior of another; it can be used interchangeably with its “parent” use case

Check Password and Retinal Scan generalize Validate User

Use Case Diagrams

A use case diagram shows a high-level picture of the users and the use cases they participate in. In a complex system, several use case diagrams can be drawn to show different views of how the system is used. The Unified Modeling Language includes a graphical notation for representing use cases as ellipses and actors as stick figures. The lines drawn between actors and use cases indicate that the actor is initiating the use case. Use cases can call upon other use cases, indicated by the <<includes>> relationship, or vary the behavior of a use case, indicated by the <<extends>> relationship. The dependency relationship is shown by a dashed line. Generalization (not shown in the diagram below) is shown by an open arrow pointing to the use case being generalized. This is the same as the inheritance relationship between classes.



Guidelines for Drawing Use Case Diagrams

GUIDELINE: Identify the “shape” of your use case model, then draw one or more use case diagrams that present meaningful snapshots of your system’ s behavior.

GUIDELINE: Don’t include every use case in a single Use Case Diagram.

You can draw more than one use case diagram. A use case can be shown on more than one diagram, too. The purpose of a use case diagram is to convey a particular organization of use cases.

Some possible diagrams: A diagram showing core use cases and their initiating actors; a diagram that emphasizes the interactions and dependencies between two actors; a high-level diagram that identifies summary use cases; a detailed diagram that shows how certain core use cases are fulfilled by “including” supporting use cases; a diagram that identifies key variations with use cases that “extend” other use cases

Use Case Levels

Use cases can be written at various levels of abstraction. They can describe sweeping overviews of system functionality. These are termed “summary” use cases. Use cases can describe task related activities of users as they interact with the system. These are “core” or task level use cases. You can describe how your software behaves in support of core use cases. We term these “supporting” use cases. You can dig even deeper and describe how components in our software behave and interact. These “internal “use cases are of value to those designing how the responsibilities of the system are distributed between components.

The most useful level to consider from the external actor’ s understanding is the *core level*.This will be the focus of our writing in class. However, sometimes other stakeholders need to see the big picture and will need to read summary use case description. Developers need the extra precision found in supporting and internal use cases. Core level use cases are linked to lower level *supporting use cases*, and are part of higher level strategies.

Use Case Model Shapes

Depending on the nature of the system you are trying to describe, your use case model may assume one of several shapes. Alistair Cockburn, in *Writing Effective Use Cases*, identifies the sailboat shape. It is a use case model that includes a well structured set of core use cases that are defined to meet strategic behaviors outlined in a few summary level use cases. In this sailboat image, most of the use cases are core- those found at the waterline where the sailboat sits in the water. At the core level, you identify specific tasks of various actors using the system. Below this waterline are supporting use cases that are used to fulfill one or more core use case functions.

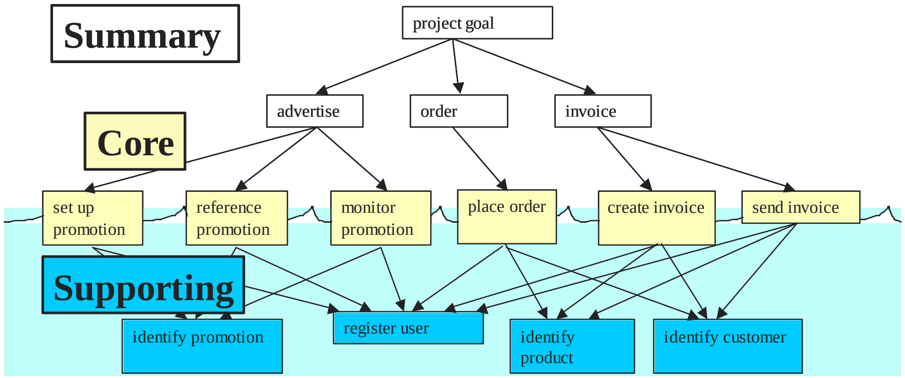


FIGURE 1. A sailboat shaped Use Case Model. A balanced number of core or task-level use cases.

A second characteristic use case model shape is the “hourglass”. This use case model is characterized by a small (could even be one) number of core or task-level use cases that call on a wide-range of supporting use cases. The core use cases could support several strategic goals. In this use case model shape, variations and complexities are typically hidden to the software user performing a core use case.

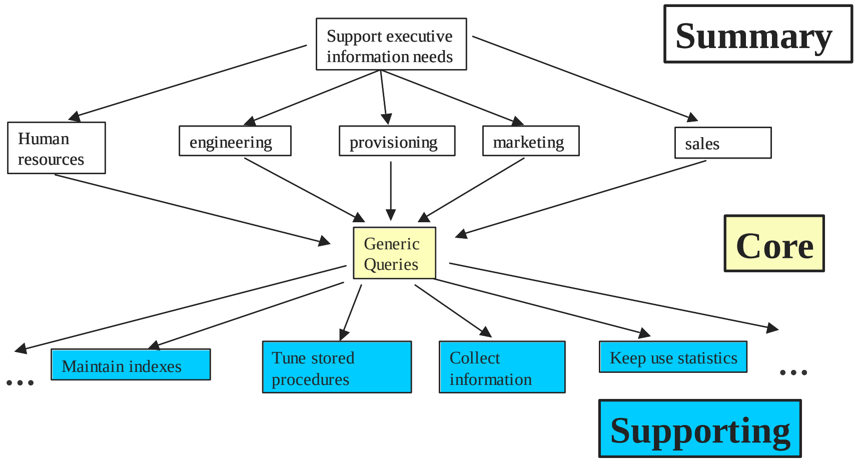


FIGURE 2. An hourglass shaped Use Case Model. Much of the complexity of the software is not evident to the user.

A third shape is a “pyramid”. In this Use Case model, there are many supporting use cases, each defining functionality that can be called on by a few core use cases. This is typical of a software application development environment or an operating system. Sometimes, there may be little or no distinction between core and supporting use cases: all may be exposed and usable by the same actors.

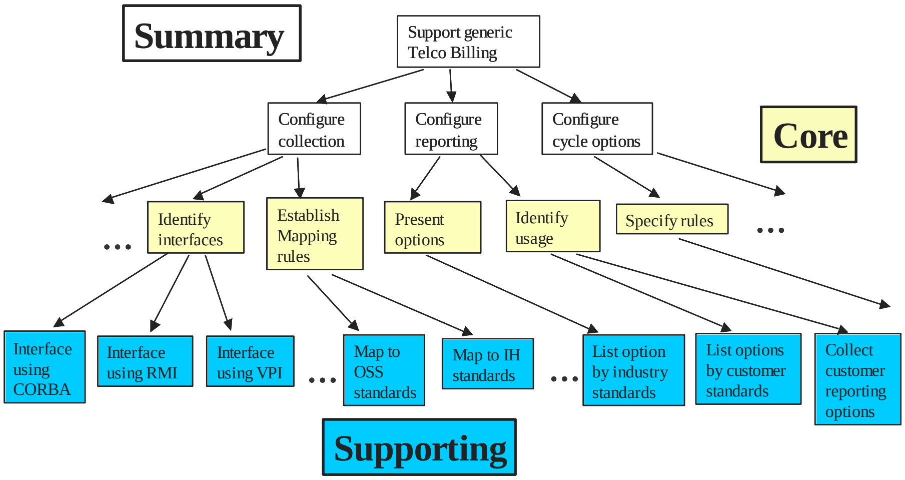


FIGURE 3. A pyramid-shaped Use Case Model. Core use cases resting on numerous supporting use cases.

Example: Defining Usage Requirements

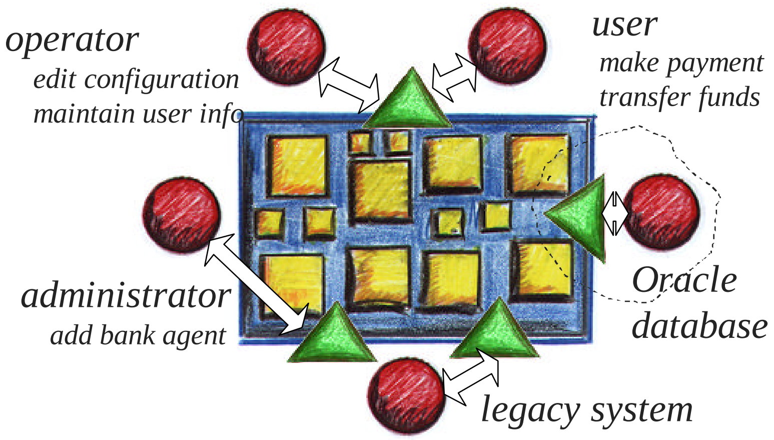
The On-line Banking System requirements consists of support for all of the tasks that the users need to perform with the system. They initiate the activities of the system and their agendas are reflected in the use cases:

- Login
- Register Customer
- View Account Balances
- View Account Statement
- Transfer Funds
- View Session Activities
- Select Setup Choice
- Edit User Information
- Edit Account Information
- Delete Account
- Edit Payment Template Information
- Make Payment

When you define what your system does for its users, you are also determining the boundaries of our system: what’ s inside?, what’ s out of scope?, what does your system do for its users?, how do they interact with it?, and how does it interact with other systems?

In the on-line bank, although the end users using the web were of primary concern, there were other people and systems that interacted with the software. These actors also initiated and participated in a number of use cases. External systems, and how they were used were important, too.The interactions and usage of legacy software were important to specify so that it could be isolated and viewed in a uniform way by other parts of the system. The use of the database was of concern to developers and the sponsors. Although the database was a secondary actor; the details of what was stored in the database, and the requirements for storing transaction details (not internal transactions) made it important to describe it in a manner that was understood by both parties.

Drawing the System Boundary
Actors and Use Cases



Use cases are only part of any system specification. Use cases are often accompanied by supporting information, pictures, more formal descriptions of algorithms, etc., that are meaningful to people who will build or use the system.

The sources of funding of the on-line bank were a consortium of South American banks, and a major computer manufacturer. They specified schedule, cost, deliverables, variability from one bank to another, support for legacy connectivity, user languages, development tools and languages, hardware platforms, and distribution requirements.

The user requirements came from representatives of the banks: the tasks to be performed on-line, the user interface, and the roles of the people that will use the application.The technical architect imposed a set of non-functional requirements on the system: reusability, performance characteristics, robustness, configurability, support for technology standards, error-handling, and fault tolerance. The patterns of usage were not nearly as difficult as the “internal”, structural and behavioral requirements imposed by the system architect and the sources of funding.

It is for this reason that use cases are only part of any system specification. They are accompanied by supporting information, pictures, more formal descriptions of algorithms, hardware componentry, etc., that are meaningful to people who will build or use the system.

V. Glossaries

The purpose of a glossary is to clarify terms so that team members can know what they are agreeing or disagreeing on. A common set of terms that are defined and understood forms the basis for all our descriptions. A glossary should be developed to accompany a use case model as well as other requirements documentation.

A glossary is a central place for:

- Definitions for key concepts
- Clarification of ambiguous terms and concepts
- Explanations of jargon
- Definitions of business events
- Descriptions of software actions

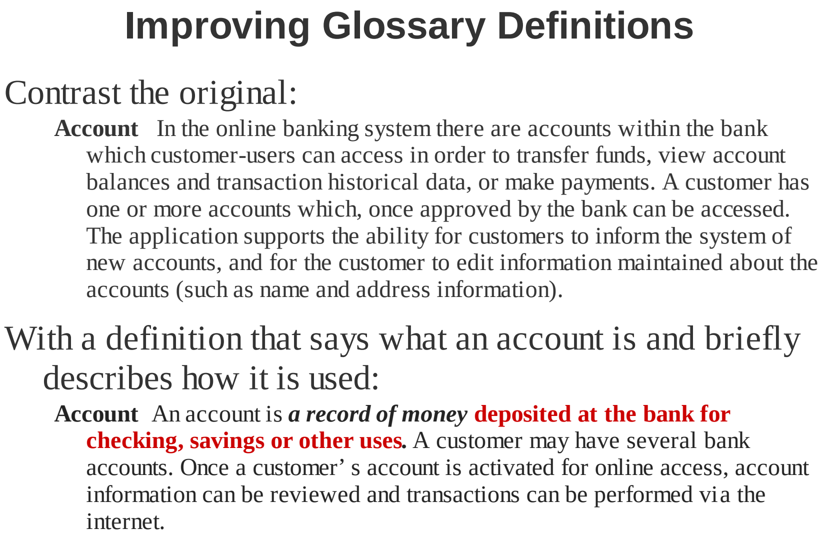
The glossary is built incrementally. Terms in the glossary form a working description of the concepts and events that exist in the various domains of the problem, and clarify the terms that we use to describe requirements and write use cases. A good glossary entry follows this form:

“Name of a concept” related to a “broader concept” + any distinguishing characteristics

For example:

A **compiler** is a **program** that translates source code into machine language.

Here is an example from the on-line bank contrasting an original version with an improved version, reworked for clarity:



Improving Glossary Definitions

Contrast the original:

Account In the online banking system there are accounts within the bank which customer-users can access in order to transfer funds, view account balances and transaction historical data, or make payments. A customer has one or more accounts which, once approved by the bank can be accessed. The application supports the ability for customers to inform the system of new accounts, and for the customer to edit information maintained about the accounts (such as name and address information).

With a definition that says what an account is and briefly describes how it is used:

Account An account is ***a record of money*** **deposited at the bank for checking, savings or other uses.** A customer may have several bank accounts. Once a customer’ s account is activated for online access, account information can be reviewed and transactions can be performed via the internet.

Experience has shown the value of developing a common set of terms for the development team. Seasoned developers, because of their wide experience in the domain, will have encountered multiple, varying definitions for many of the core concepts. A concept glossary levels the playing field and unifies these diverse points-of-view. For team members new to the domain, a concept glossary offers a jumpstart to understanding the domain, and is vital to understanding the requirements.

GUIDELINE: Write definitions for key concepts.

GUIDELINE: Build incrementally when writing requirements.

GUIDELINE: Add supplementary information.

Why is this concept important? What are typical sizes or values? Clarify likely misunderstandings. Explain graphical symbols

GUIDELINE: Determine an appropriate name for each concept.

GUIDELINE: Normalize names.

Identify behaviors that are the same but have different names. Identify behaviors that are different but have the same name.

GUIDELINE: Define acronyms and their concepts.

Example: OSS - Operations Support System: As defined by the FCC, a computer system and/or database used at a telephone company for pre-ordering, ordering, provisioning, maintenance and repair, or billing

GUIDELINE: Use pictures to relate concepts.

Example: We recommend defining terms and relating them with a picture as the best way to get across complex relationships. Here are some related concepts:

wire center- the geographical area served by a central office

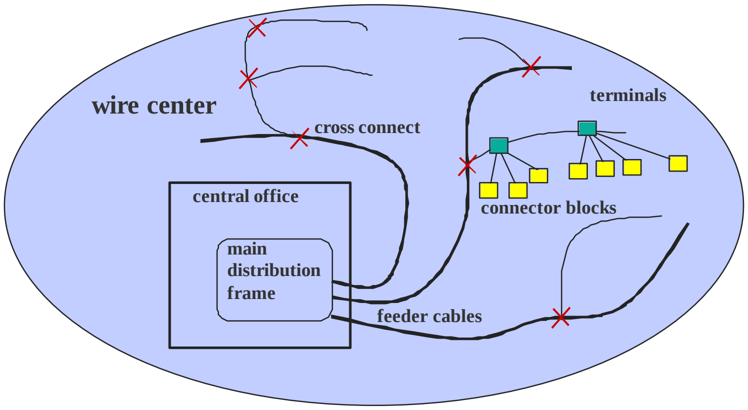
central office- a building where local call switching takes place

main distribution frame- a large connector at a central office, which connects switching equipment to feeder cables feeder cable- a large cable that connects to the main distribution frame at a central office and feeds into distribution cables

distribution cable- a cable that connects between a feeder cable and one or more terminals

and a picture showing how they are related:

A Picture Relating Hierarchical Concepts



GUIDELINE: Avoid vague words.

GUIDELINE: Avoid Using *is when* or *is where*.

Good: An overplot is an overlap between two or more graphic entities drawn at the same place on a page

Bad: An overplot is when two things overlap

GUIDELINE: Define a particular status as a list of possible states.

Example: A proposal’ s approval status is its current stage in the process for granting or denying it: *awaiting department approval, awaiting chair approval, awaiting board approval, or denied*.

GUIDELINE: Use team development and review to build consensus for definitions.

VI. A Template For Writing Use Case Descriptions

Here is a template for filling in additional information that can accompany the description of the interaction between the actor and the system. Several authors have proposed their versions of a Use Case Template. They are similar but have slight differences. This discussion presents an overview of elements that can be part of a use case template.

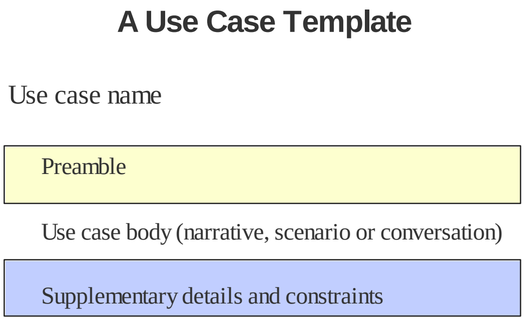

A Use Case Template
Use case name
Preamble
Use case body (narrative, scenario or conversation)
Supplementary details and constraints

FIGURE 4. The parts of a Use Case Template

We recommend you start by adopting a template that is fairly lightweight (we include more information in this template than you may need to get started). Depending on where you are in a project, you may start by only filling in part of the information in a template...and then add more details in a second iteration. For example, you may start by only writing a narrative and identifying the actor for the use case. Later, you may describe exceptions and add conversations or scenarios that expand on the basic narrative.

It is useful to divide the template into three parts:

- the preamble- which defines the context of the use case
- the body - which describes the actor' s interactions with the system, and
- supplementary information - which adds details and constraints on the use case' execution

The Preamble

The preamble contains information that "sets the stage" for the behavior described in the body of the use case

In the preamble, you may find the following information:

- Level - summary, core, supporting or internal use case?
- Actor(s) - role names of people or external systems initiating this use case
- Context - the current state of the system and actor
- Preconditions - what must be true before a use case can begin
- Screens - references to windows or web pages displayed in this use case (if a UI is part of the system)

The Body

Description of the use case' s behavior. This description can be:

- A narrative - a free form paragraph or two of text.
- A scenario - a step-by-step description of one specific path through a use case
- A conversation - a two-column (or more columns if showing dialogs between multiple actors and/or system components) description of the dialog between the actor and the system.

In a single use case, you may write a narrative, and, once you've worked out how the actor will interact with the system, then write either a scenario or conversation. Nothing says that the body has to be restricted to one form. But most of the time we see writers start by writing a very brief narrative (of just a couple of sentences), then write either a scenario or a conversation that goes into more depth. They leave both forms around—the narrative as an overview (which only certain stakeholders read) and the other as an in depth presentation of actor/system interaction.

Supplementary Details

- Variations - different ways to accomplish use case steps
- Exceptions - errors that occur during the execution of a step
- Policies - specific rules that must be enforced by the use case
- Issues - questions about the use case
- Design notes - hints to implementers
- Post-conditions - what must be true about the system after a use case completes
- Other requirements- what constraints must this use case conform to
- Priority- how important is this use case?
- Frequency - how often is this performed?

GUIDELINE: When you start writing use cases, describe the key points. Typically, this means giving the use case a name, identifying the actor and writing the use case body (one of the three forms).

GUIDELINE: Fill in template fields as information becomes available.

As you write a narrative, you may think of an issue or a note to the designer. Jot these down when you think of them. Don't wait for the perfect time. The right time to add a detail is when it occurs to you. You can always note a fact, then fill in more complete details later.

GUIDELINE: Make clear how complete a use case is.

Daryl Kulak and Eamonn Gray in their book, *Use Cases Requirements in Context* identify four phases of a use case description: facade, filled, focused and finished.

Whether you pick these four "degrees" of completeness or some other measure of completeness, it is a good idea to note whether a use case is a first draft, whether it has been reviewed, when it has been revised or approved by various stakeholders, and "signed off" as finished.

GUIDELINE: For more formal projects, information about the current state and history of a use case can be added to the template..

Add this information as supplementary details. This lets readers of the use case see the main points first. If you include this information are part of the preamble, it adds clutter that has to be scanned over before the reader finds the main facts about the use case.

A use case description can start out simply, then get quite complex as template details are filled in. Start simply, writing down what you know and issues that need to be addressed. Through several revisions and refinements get to a "finished" use case.

VII. The Narrative Form

Narratives are free-form text in paragraph format. A narrative describes the intent of the user in performing the use case, high-level actions of the user during the use case, and refers to key concepts from the problem domain that are involved in the use case.

Below is an example narrative from the On-line Banking System Specification Documents. We have briefly described the purpose of *Log In* and what happens as a result of the user successfully completing the *Log In*. We've also included a set policies that relate to logging in, and have listed some exceptions that may arise during *Log In*.

Use Case: *Log In*

Log In is the primary entry point into the On-line Banking System. *Log In* verifies that the user is previously registered with the On-line Banking System, and that s/he has correctly entered user id and password information. After a successful login, a registered user can use the system's main functions. All others, regardless of whether they have registered or not, have access to the On-line Banking System Demo and Registration Page.

Recipe: Writing Use Case Narratives

1. Give the use case a descriptive name.

GUIDELINE: Begin the use case name with an active verb.

2. Identify the actor that uses the use case.
3. Identify the intended audience of the use case.
4. Specify the actor's goals for the use case.

GUIDELINE: Use active verbs to describe the actor' s goal.

5. Write a description consistent with the name and the user's goal; one that elaborates the use case.

GUIDELINE: Maintain a single point-of-view: the actor' s.

GUIDELINE: Describe intent, not action.

GUIDELINE: Capture the simple, normal use cases first. You will describe the variations as *secondary* use cases later.

GUIDELINE: If the use case changes the state of some information, describe the possible states.

GUIDELINE: Write the use case description at a level appropriate for the intended audience.

GUIDELINE: Leave out details of user interface, performance, and application architecture. Put these details in a central document, and reference these requirements in the use case.

6. Describe any business rules or policies that affect the use case in a separate place: either in a policies section below the use case body; or in a global policies section. Reference globally applicable rules or policies in the use case policy section of the template.

VIII. Scenarios and Conversations: Writing More Detailed Usage Descriptions

One key to developing a usage model is knowing how much to describe. A closely related question is, "What' s the best way to present detailed information?" Use case narratives are general descriptions about how a system supports an actor' s goal. There may be numerous ways to achieve any goal. Sometimes it helps to clarify things by concretely describing actions and information for a specific situation.

Scenarios and conversations are forms that are useful to show in more detail how an actor achieve' s a specific goal.

How many use cases should be written? A glib answer: "As many as it takes to get the main ideas across." The number is highly dependent on how close your intended audience is to the problem, and how many details they need spelled out.

Here' s one general word of advice: Write to be read. If it clarifies and brings understanding to your system' s behavior, write narratives to describe the general situation, then augment those narratives with specific descriptions. If your readership only looks at details, narratives likely won't be of value.

GUIDELINE: High-level use case names state a general goal. Write one narrative use case for each general goal, and as many scenarios or conversations as it takes to get the main ideas across.

For example:

Narrative: Make a payment

Describe what online payment means and typical ways of making them

Write scenarios or conversations that describe more specific goals:

Scenario 1: Make a recurring payment

All the steps in paying my monthly phone bill …

Scenario 2: Make a non-recurring payment

All the steps in paying a fixed amount …

Scenario 3: Make a regular payment

All the steps in paying a monthly loan …

Sometimes, your use cases are read by diverse audiences. Some want to only see details. Others only want “big picture” overviews. In the interests of keeping everything together, and not creating a maintenance problem, we suggest you bundle both a general and a specific description together in a single use case.

GUIDELINE: Write two “versions” of the same use case: one version a narrative, the other version a more detailed form.

Example:

First, write a narrative

The “View Recent Account Activity” narrative describes generally how users view the current or previous account period’ s transactions

Then, choose an appropriate form. Rewrite the use case body at this lower-level of detail

The View Recent Account Activity conversation includes the details of optional actions, such as downloading a file containing recent transactions in several different formats

Leave the narrative as an overview. Consider adding an “overview” section to your template if you have always have diverse readers for your use cases.

What is a Scenario?

Scenarios are one means to describe a specific path through a use case. A scenario list specific steps toward that goal. It describes a sequence of events or list of steps to accomplish.Each step is a simple declarative statement with no branching. A scenario may describe:

- Actors and their intentions
- System responsibilities and actions

All steps should be visible to or easily surmised by the actor. We typical state a statement by naming who is performing the step. Our goal is to convey how the system and actor will work together to achieve a goal. Even though a scenario can show more detail, resist putting in too much detail. Much of that detail can be placed in the preamble or supplementary parts of the use case template.It should be clear where a scenario starts. Describe the steps in achieving the actor’ s goal. End there.

For example, we might write a scenario toward the user’s goal of “Register a Customer.” This specific scenario explains a variation of this task called “Register Customer with Auto-Activation.”

Example Scenario: Register Customer with Auto-Activation

1. User enters registration information:
   - *Required information*: user name, email address, desired login ID and password, and confirmation password
   - *One of*: account number and challenge data, or ATM # and PIN
   - *Optional*: language choice and company
2. System checks that password matches confirmation password.
3. System validates required fields and verifies uniqueness of login ID
4. System verifies customer activation information.
5. System creates and activates customer on-line account.
6. System displays registration notification.

Recipe: Writing Scenarios

The purpose of a scenario is to describe the *flow of events* in the use case. These events can be initiated by the user or performed by the system, but should express the steps of the process as the user understands it.

1. For each use case, determine the “happy path” to the actor’s goal.

GUIDELINE: Ignore other possible paths through the use case at first. Write these “secondary” scenarios later.

GUIDELINE: Refer to the specific use case that the scenario elaborates, if the use case has been written.

2. Write a scenario as a sequence of steps, ordered by time.

GUIDELINE: Every step in a scenario should be visible to or easily surmised by the user.

GUIDELINE: Write each step as a simple, explanatory statement.

GUIDELINE: Keep information and actions concrete.

GUIDELINE: Focus on ordering and definition of steps.

GUIDELINE: Factor lower-level details into new descriptions.

GUIDELINE: Keep steps ar roughly the same level of abstraction.

In following example, several steps have been compressed to keep actions at the same level.

Mixed level of detail:

1 | Check for required fields

2 | Capture user ID and password
Ask security component for validation
Issue SQL statements to security database for logon authorization…

3 | Open connection to bank server
Read account summaries…

Fixed:

1 | Check for required fields

2 | Login user to domain

3 | Display account summaries and bulletin

3. Number the steps.

GUIDELINE: Don’t get carried away. Keep the numbering one level deep. Remember, the goal is clarity.

4. Look for steps that might repeat within the scenario.

GUIDELINE: To show repetition, use *repeat* or *while* statements.

GUIDELINE: Avoid the tendency to write pseudocode unless your audience are programmers who only understand code.

5. Look for steps that depend on a condition.

GUIDELINE: To show that a step depends on a condition, use an *if* statement.

GUIDELINE: When the logic for expressing a conditional statement becomes too complex, write another, *alternative* scenario.

GUIDELINE: Distinguish between *variations* and *exceptions*. Describe recovery from exceptions in a supplementary note or another scenario if the recovery is complex. Document variations in either a supplementary note, or another scenario if the actions are interesting.

6. Look for sequences of steps that repeat *across* scenarios.

GUIDELINE: Don't do this early in your project. Later, factor out portions of a scenario that repeat in other supporting scenarios, give them a name, and refer to them within the core use case with a reference to the supporting use case' name.

7. Look for optional steps.

GUIDELINE: Preface optional steps or actions with "Optionally,..". Indent optional steps for clarity.

8. Show the range of values of data that is used in the scenario.

GUIDELINE: If the user changes the information, specify the possible states that the information might go through.

What is a Conversation?

A conversation describes a significant sequence of interactions between an actor and the system, or between one part of our system and another. It is a detailed description of a Use Case that clearly defines the responsibilities of each participant.

There are two central parts to a conversation, a description of requests or inputs, and a corresponding description of the high level actions taken in response. Together, these "side-by-side" descriptions capture a sequential ordering of communications.

Like a scenario, it can show optional and repeated actions. Each action can be described by one or more substeps

The focus of a conversation is to detail the types of interactions, the flows of information, and the first-level system logic of the system, all from the user's point-of-view. If desired, it can also be used to drill down to the deeper levels of system logic, as seen by a developer.

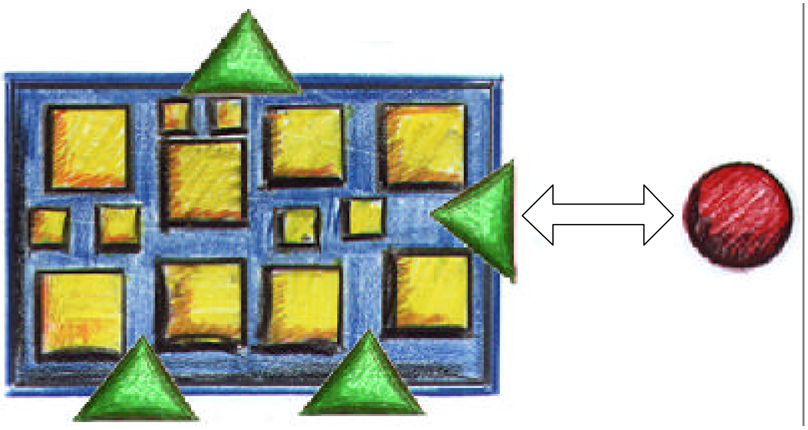


FIGURE 5. A conversation shows a dialog

Because of the various ways in which a user task can be performed successfully, there may be one or more conversations for a single use case narrative.

Often, it is too big a gap to move directly from informal Use Case written in narrative form to design. Also, use cases written at this higher level are full of ambiguities and extraneous details that have little to do with what our system must do for the user. As you restructure use cases into conversations, you add more detail by:

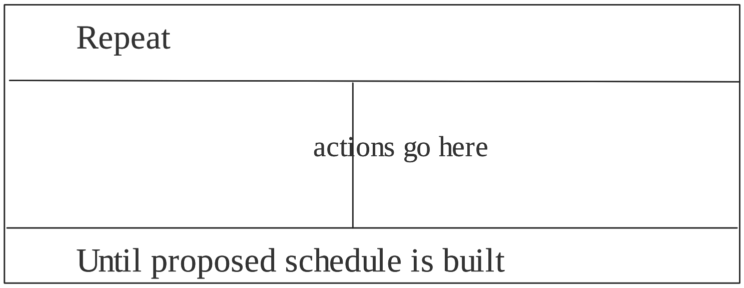
- showing branching and looping,
- describing constraints on what our software should do,
- describing the context in which the conversation occurs,
- identifying the actors that initiate the activity,
- defining the “standard” course of action and alternatives to it,
- raising unanswered questions, and
- adding design notes.

TIP: This supporting information is often as important as is defining the order to the user's actions.

Notation

Use a table format to record a conversation several stylistic shorthand conventions. Other use cases invoked during a conversation are marked in bold text. These use cases could be “used” (UML's “includes” relationship), or transfer of control could passed via a “goto”. These control flow conventions proved extremely relevant to the UI designer and application server implementation, but are unimportant to a high-level view of a use case.

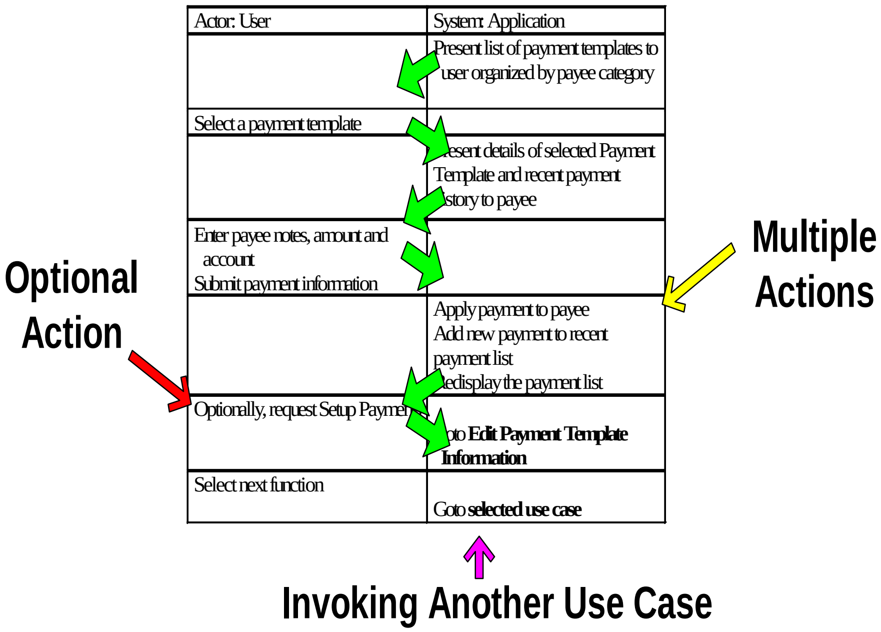
Optional actions, for example (Indicate Setup Payees), are labelled. Show looping or repetitive steps by merging adjacent cells in a row to bracket the beginning and end of a block of repeated or optional actions:



Repeat	
actions go here	
Until proposed schedule is built	

FIGURE 6. Showing Repetition, or an optional block of actions

Placing dialog in adjacent cells of the same row shows an interactive round (an actor action that invokes a nearly simultaneous system response). Placing the system response in the row immediately after the provoking action denotes a batch round.



Actor: User	System: Application
	Present list of payment templates to user organized by payee category
Select a payment template	
	Present details of selected Payment Template and recent payment history to payee
Enter payee notes, amount and account Submit payment information	
	Apply payment to payee Add new payment to recent payment list Redisplay the payment list
Optionally, request Setup Payment	Goto **Edit Payment Template Information**
Select next function	Goto **selected use case**

FIGURE 7. Conversation notation

This is an example of a dialog between the customer-user and the system. We used a table format to record this dialog and several stylistic shorthand conventions. Other use cases invoked during payment were marked in bold text. These use cases could be “used” (UML’ s uses relationship), or transfer of control could passed via a “goto.” These control flow conventions proved extremely relevant to the UI designer and application server implementation, but are unimportant to a high-level view of a use case.

In the on-line bank, our web-based interface design did not allow for simultaneous interactions; instead information would be batched and passed along with an action tied to a button. A more traditional window application or a Java applet has the potential for many more overlapping activities.

Writing Conversations

Knowledgeable experts from diverse backgrounds can readily construct conversations. Conversations can either be developed by a team or drafted by an individual then reviewed, explained, and revised by a small group. It is important that teams who develop conversations blend the talents of developers, users, and other specialists. Each contributor has a unique and valuable perspective. No perspective should dominate, yet a certain interest may take center stage during a working session. It is important that side concerns be recorded, and worked through, perhaps as an outside activity. Respect and appreciation for the concerns of others is important; teamwork and a spirit of joint development is crucial. For example, in one working session, we dived into technical design details for several minutes, backed up to re-examine whether the flow of the conversation we had proposed was still workable, then summarized what issues were solved and what new ones were raised by a single decision. Technical, user interface and business issues were all discussed in a single session while holding everyone's attention.

One key to building a good conversation is to preserve its dual purpose of

1. recording of the important events and information that are conveyed between the user and the system; and
2. guiding developers who will be creating the object design model.

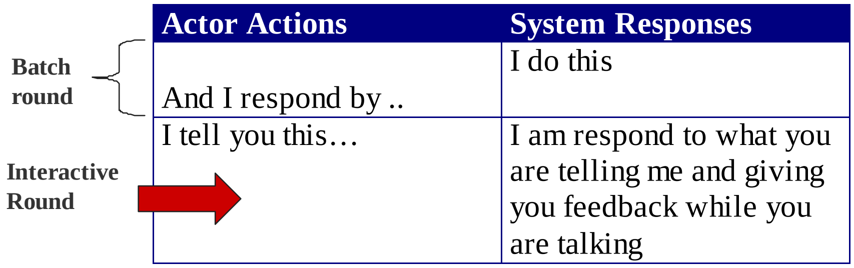
To meet these objectives, conversations must be written at a fairly high level. It often is the case that sequencing of model responses (for example the details of recording a payment transaction) are not accurate reflections of the tasks that the system must do. Yet, they need not be early on. What is important, is that an interdisciplinary team is sketching out how they expect their software system to work.

Conversations capture the flow of communication between actor and system. If the nature or the amount of information changes significantly, the demands on your object model also change. So we suggest that you include sufficient detail, and reflect changing interaction and interface design if conversations are to actually guide object modeling.

What is a dialog?

The basic form of conversation is the dialog. A dialog is a conversation where both sides participate in a structured sequence of rounds of interaction. Each round is a pairing of an action taken by the user, and the software system's response to this action. A round is one of two types. It is either an interactive round or a batch round. An interactive round features interplay of user actions and system responses. For example, the validation of a single key press among many is typical of an interactive round. In contrast, filling out several entry fields and then submitting them all at once is a more typical of a batch round.

This sequence of rounds establishes a necessary ordering of the interactions, and details the individual activities of the user and the system's response in each round. Many of our conversations between human actors and our system are of this form



Actor Actions	System Responses
	I do this
And I respond by ..	
I tell you this…	I am respond to what you are telling me and giving you feedback while you are talking

FIGURE 8. General form of a conversation

Choosing Between Conversations and Scenarios

The detailed form you choose to use depends on two primary factors:

- whether or not your system has meaningful dialogs between its users; and
- personal preference

Use a scenario when:

- a simple list of actions is sufficient
- actor-system interactions aren’t interesting

Chose a conversation when:

- there are many interactions and you want to describe them
- you want to show more details in your system responses
- you want to separate the roles of actor and system and clearly identify at each point the system does for the actor

We have written conversations for systems where there isn’t a lot of interaction between actor and system. This becomes readily apparent from looking at the staggered pattern of filled in cells.

Most projects write high-level use case narratives, then standardize on one of the two more detailed forms to describe all use cases. Whether you want to write conversations or scenarios may not be obvious until you understand the nature of your system’ s interactions with its users.

Recipe: Writing Conversations

GUIDELINE: Write a conversation if it is important to show the patterns of interaction between the actor and the system.

GUIDELINE: Write a conversation if you want to show the first cut at system-actor actions in greater detail.

GUIDELINE: If you have written a scenario and find that it does not offer you enough detail, rewrite it as a conversation.

1. List the actor actions in the left column and the system actions in the right column.

GUIDELINE: Leave out presentation details.

GUIDELINE: Maintain a consistent level of detail.

GUIDELINE: Don’t embed alternatives in your action descriptions.

It can get complicated for your readers to deciphers nested “if then... else, if..” statements if they are liberally sprinkled through your action statements. You can keep the statements simple if you write the “happy” path description in the body of the conversation. Call out exceptions and variations below.

Pseudo-code:

Conversation: Registration with Automatic-Activation

	10. If bank supports automatic activation with ATM and PIN then... If ATM and PIN #s are valid then....

Fixed:

Conversation: Registration with Automatic-Activation

	10. Validate ATM and PIN #

Exception

Step 10: ATM and PIN #s are invalid- Report error to user

GUIDELINE: Don’t mention “objects” in system responses.

Remember that your readers what to know what is happening from an external perspective, not what the system is doing behind the scenes. For example, rather than stating “create customer and account objects” you can rewrite the system’ s response to more clearly explain what the system has done to benefit the actor: “record customer account information”.

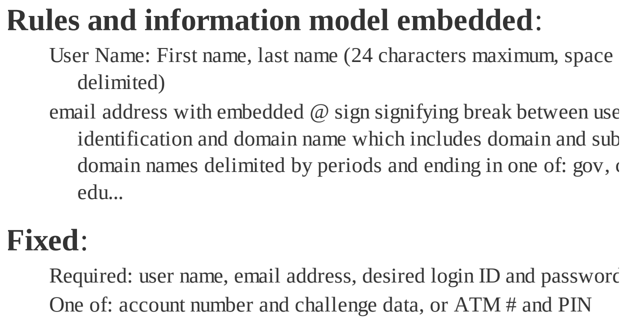
GUIDELINE: Write conversations with a small group (maximum of 3).

When we first started defining the On-line Banking System, we wanted every developer to understand all facets of the system. This quickly proved impractical and slowed everyone down. So, two of us focused on use cases and conversations, interacting primarily with the system architect and domain expert. Eventually one person took over maintaining use cases; everyone else used them as reference material. For example, the developer who designed and implemented the application server only raised questions when conversations were unclear or inconsistent, and was quite content to not always work from the latest documentation until things settled down. The project manager and project sponsor didn't read these documents at all (unlike other projects we've worked on where management enjoys reading and commenting on them in detail).

2. When the system has an *immediate* response to an actor action (such as validating a key stroke), list them in the same row.

GUIDELINE: Leave out information formats and validation rules.

These are best kept in a separate place that can be maintained and updated as business procedures and policies may change. Only summarize what information is presented to or collected from the actor in the conversation.



Rules and information model embedded:

User Name: First name, last name (24 characters maximum, space delimited)

email address with embedded @ sign signifying break between use identification and domain name which includes domain and sub domain names delimited by periods and ending in one of: gov, edu...

Fixed:

Required: user name, email address, desired login ID and passwor

One of: account number and challenge data, or ATM # and PIN

3. When the response is delayed until an entire actor action is complete, list it in the row immediately below the row with the actor action.
4. Write any assertions about the software's states during the conversation.
5. As you consider the actions in the conversation, document any ideas about "how" in a *Design Notes* section.
6. Test the conversation with a walkthrough.

GUIDELINE: Use specific examples to walk through use cases and conversations.

GUIDELINE: Trying to abstract or write more general conversations too early tends to create problems. It is better to deal with specific situations first, then review and combine things as appropriate, after you have the big picture. This strategy led us to write different conversations to record different typical uses. For example, we discovered two common ways customers could make payments, one for paying same amount to the same vendor and one where the amount paid varies. This led us to write two separate conversations, Make Similar Payment and Make Payment. In version two, when we would support automatic payments, Make Recurring Payment would be added to our Payment Use Case conversations.

7. Check conversations for completeness.
8. Relate the conversations through their preconditions and postconditions.

IX. Other Descriptions, Exceptions and Variations

"Other" requirements are those that are not captured within the body of a use case, or within or within other parts of the use case template. They can either be kept in a central place or can document the use case where they seem to apply. In fact, if you are following a rigorous requirements specification process, you may gather and record many requirements that, while they *may* impact your system' s usage and design, belong elsewhere.

Keep Common Requirements in a Central Place

GUIDELINE: Document requirements spanning use cases in a central place.

For example: "Financial transactions must be secure.", and "System must run 7x24"

GUIDELINE: Refer to specific "central" requirements by name in the use cases that they impact if this impact is not obvious to the reader and it' s important to know.

Note Specific Requirements in Use Cases they Affect

GUIDELINE: Document specific requirements in the use case that they pertain to.

For example: "Registration response time must be less than one minute."

GUIDELINE: Look for requirements that are invisible to the actor.

For example: "System must not lose any requests", or "Application servers will be widely distributed"

GUIDELINE: Look for performance requirements that affect system behavior.

Design Notes

Design notes, if part of your use case template can "round out" your usage descriptions with ideas that occur to you that might be useful during design. Since a use case isn't a descriptions of a solution, don't write these details there. But if you think of a good design idea, you may want to jot it down and keep it with your use cases.

GUIDELINE: Add design notes as they occur to you when writing scenarios and conversations.

For example: "Errors and warnings about registration information contents should be collected and returned to the user in a detailed message rather than stopping at the first detectable error", and "Payments should be shown in time order, with the current date first."

GUIDELINE: Write design notes as hints or suggestions, not as instructions to the designer. Don't be too detailed.

Alternatives

Distinguishing which courses of action are the "main" paths is often difficult. We have two options for documenting alternative courses of action (variations) and points of potential error (exceptions) in a use case. If the alternative can be stated simply, we embed *if-statements* in the description. For a slightly more complex alternative, we note these alternatives in supplemental text below the basic path. Or, when the alternative flow of events is complex, we can write completely new use cases for these alternatives. In the latter two situations, we reference the point in the original use case where the alternative takes place.

Variations

A *variation* can be a different action on the part of either the actor or the system. When you see this possibility, be opportunistic! Don't let the insight go by. Capture the conditional choice in an *if* statement, describe the difference in supplemental text below the use case body, or write another use case. One that incorporates the alternate action. Note the name of the new use case so that you can write it later.

Exceptions

On the other hand, actions that have the potential for errors, again, on the part of either the actor or the system. Treat these errors similarly to variations, but note them under a separate heading in the supplementary part of the use case template. They are the source of many of the error-handling requirements of the system.

GUIDELINE: Describe the exception and its resolution. Identify whether it is recoverable (e.g. the actor can continue on with his/her task in some fashion) or unrecoverable.

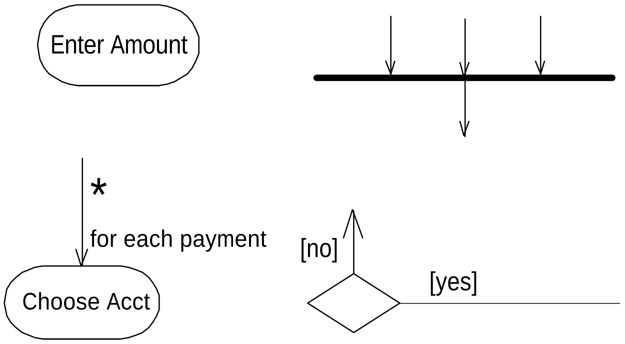
GUIDELINE: For each recoverable exception describe how the actor/system needs to respond to make forward progress.

GUIDELINE: For each unrecoverable exception, make clear what state the system returns to after detecting this condition, and how the actor is notified of this condition.

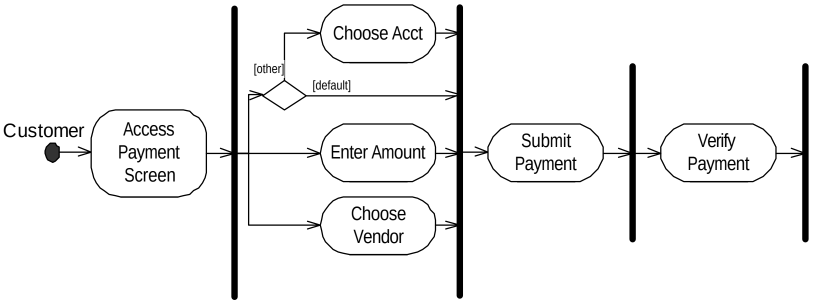
Activity Diagrams

Activity Diagrams are a UML standard way of describing sequences of actions, the dependencies among them, and the parallelism and synchronization characteristics. Use them as a way of visualizing activities at several levels: the process level that demonstrates how different use cases interact, the task level that shows the activities of a user when performing a use case, and the subfunction level that shows the internal workings of a single step, whether it be performed by a user or a computer program.

These elements show an activity (the oval), synchronization of activities (the synchronization bar), decision-making (the diamond), pre and post conditions (the *guards*, text inside the square brackets annotating the arrows), and iteration (the asterisk on annotating an arrow).



This activity diagram demonstrates the actions that take place when "Making a Payment". It is at the task level and describes a single use case.



GUIDELINE: Use an activity diagram to describe a single use case. The goal is to understand what actions that take place and the dependencies between them

GUIDELINE: Use an activity diagram to understand workflow across use cases. Activity diagrams are great for showing connections and dependencies between the use cases of an application.

GUIDELINE: Use an activity diagram to show parallel activities. Activity diagrams are particularly good at showing parallelism, synchronization, and pre and post conditions.

Recipe: Writing Exceptions

The typical paths through the use case is specified in *primary* descriptions. Alternatives to these paths can be written as *secondary* use cases, or named in *variations* and *exceptions* sections.

1. Look for potential exceptions in each primary use case.

GUIDELINE: When looking for exceptions, ask:

- Is there something that could go wrong at this point? (exception)
- Is there some exceptional behavior that could happen at any time?

GUIDELINE: Be opportunistic! Document the exceptions whenever they occur to you.

The first step is identifying the exception. The next step is resolving how it will be handled.

GUIDELINE: Keep the exceptions at the same level of abstraction as the use case description.

2. Determine which exceptions should be written as separate use cases.

GUIDELINE: Defer writing these secondary use cases until you feel satisfied with your primary ones.

GUIDELINE: Write secondary use cases according to the recipe and guidelines for primary ones.

3. Document the exceptions.

GUIDELINE: Describe the exception condition. Note whether it can be recovered from or not. Describe the actions the actor or system take to recover; or to end the use case in an unrecoverable situation.

GUIDELINE: Choose the clearest way to describe how the exception is handled

Options include:

- Briefly describe what happens, or
- Refer to another use case that describes the exception handling

4. Refer to the place in the original use case where the exception takes place.

GUIDELINE: Insert footnote numbers or tags into the main scenario, and tag the alternatives with the same number.

Recipe: Writing Variations

The typical path through the use case is specified in the *body* of the use case. Alternatives to these paths can be written as *secondary* use cases, or named in *variations* sections.

1. Look for potential alternatives in each use case body.

GUIDELINE: When looking for variations, ask:

- Is there some other action that can be taken at this point?

GUIDELINE: Be opportunistic! Document the variations whenever they occur to you.

2. Determine which variations should be written as new use cases.

GUIDELINE: Defer writing these secondary use cases until you feel satisfied with your primary ones.

GUIDELINE: Write secondary use cases according to the recipe and guidelines for primary ones.

3. Document the variations.

GUIDELINE: If the variation is easily added to the use case body, put it there. Show that variations are optional by indicating that one of several choices can be made for a particular step.

GUIDELINE: If the description of variations clutters up a use case description, write about it in the supplementary part of the use case template.

Refer to the place in the original use case where the variation takes place.

GUIDELINE: Insert footnote numbers or tags into the main body, and tag the variation with the same number.

Assertions

Assertions about our system’ s behavior are useful for:

- Generating the flow of system events
- Determining use case dependency relationships
- Understanding the states of the application

We make three kinds of assertions: preconditions, postconditions, and constants.

Pre-conditions

Pre-conditions are what must be true of the state of the application for the use case, scenario or conversation to be *applicable*. They can also imply the possibility of some order of the use cases, as we will see in the next section on post-conditions.

Post-conditions

Post-conditions are what must be true of the state of the application as a *result* of completion of the use case, scenario, or conversation.

For example, in the On-line Banking *Make a Payment* use case, debiting an account leaves the system in one of two states:

- InGoodStanding
- OverDrawn

These post-conditions of *Make a Payment* lead to two different system states. In the first case: “OverDrawn” leads to *not permitting another make payment* use case to execute (until the Account is InGoodStanding) because InGoodStanding is a precondition for Make a Payment. In the second case, InGoodStanding enables another Make a Payment to be executed.

GUIDELINE: Document pre and post-conditions where the system responds differently as a result.

We have seen many people struggle with the question, “what’ s a good post condition?” A bad post condition adds clutter and doesn’ t add any information. Restatements of the actor’ s goal don’ t add information. If the goal is to Make a Payment, then saying that a payment has been made doesn’t add any information. A good test of a post-condition is that it states something about the system that may or may not be obvious from completing a use case. And, ideally, a post-condition may enable another use case to be executed.

Example of a poorly stated post-condition that restates the use case goal:

Post-condition: Customer has withdrawn cash

So what? The customer receives cash but what does this say about the next time he/she wants to withdraw case, or any other use case?

Fixed:

Post-condition 1: Account balance is positive

Post-condition 2: Account is overdrawn

Note that the user may have achieved his/her goal, to withdraw cash, but depending on the amount withdrawn and the account’ s balance, his/her account may be in one of two possible states after successfully withdrawing cash. Now that’ s interesting!

GUIDELINE: Specify pre- and post- conditions only when you need to be formal

Once you add pre- and post-conditions to one use case, you will need to add them to dependent ones! A use case model that only has pre and post-conditions on a few use cases begs the question, is this complete or are there gaps in this specification?

GUIDELINE: Check for completeness of use case dependencies by asking how each use case is enabled, and the conditions it sets that enable others.

Example: Pre-conditions should make clear when a use case can execute

An account must be in good standing and the daily withdrawal limit not exceeded in order to withdraw cash

Post-conditions may be relevant to other systems

Being overdrawn may trigger transaction fees

Pre-conditions may be set by other systems

An account can be overdrawn through direct payments

GUIDELINE: Complete the specification of pre and post-conditions by documenting the possible states of the system after each exceptional condition, and each variation of a step.

Example:

Often, there are multiple post-conditions for one scenario or conversation

At least one for each successful goal...

Customer receives cash? Account is overdrawn or Account balance is positive

One for each exception...

Account daily limit would be exceeded - Customer withdraws lesser amount? Account is in good standing and Account daily withdrawal limit reached

Amount would exceed overdraw limit - We refuse to disburse cash? account is in good standing

One or more for each variation...

Fast cash? Account is overdrawn or Account balance is positive

Constants

Constants, sometimes called *invariants* are what must be true of the state of the application *during* the entire progress of the use case, scenario, or conversation. They are often contextual and must not be changed at any moment during the use case.

GUIDELINE: Be careful about getting too formal. Assertions tend to make requirements look incomplete if they vary in their formality.

GUIDELINE: Use pre-conditions to make it clear *when* a conversation might execute.

GUIDELINE: Write post-conditions as if you were going to use them as a basis for writing a test plan. You are.

GUIDELINE: Write constants to describe conditions that should not change during the conversation.

X. Use Case Model Checklist

At the end of the day, the goal of a usage model is to convey how a system behaves and responds to its users. A good usage model conveys how a system behaves, and how behaviors are related.

You can look over a use case model to:

- Check for internal consistency between use cases
- Identify “central” use cases
- Identify unmet or externally satisfied preconditions
- Review the actor’ s view for completeness
- Review the handling of exceptions
- See that use case dependencies, extensions and includes relationships have been documented

Organizing Your Use Cases

Organizing use cases is important. A pile of usage descriptions, arranged alphabetically, doesn't orient readers to the usage terrain. We suggest that you choose an organization that helps orient your typical reader.

Some possible organizations:

- by level (summary first, core next, supporting, then internal ones last)
- by actor
- by type of task
  arranged in a workflow

Be consistent. Keep various forms of a single use case together.

XI. A Use Case Writing Process

The task of writing can be shared, but the best way to develop a common language is for teams to work on developing a rhythm to their work. Sometimes it is best to get group consensus, othertimes it is best to work alone (or in a small group) to create use cases that others can review. Writing, like programming, can be done solo, then reviewed as a group. Once you pick a template and learn the common ideas, you can try writing solo, then critiquing as a group. Group review can lead to a common style and format for usage descriptions. We suggest this process as one way to work collectively and individually to develop a use case model:

Full Team	Small Teams or Individuals	The Products
Align on scope, level of abstraction, actors, goals, point-of-view		**Actors, Candidate Summary Use Case Names**
	Write summary descriptions	**Narratives**
Collect and clinic, brainstorm key use cases		**Candidate Core Use Case Names**
	Write detailed descriptions	**Scenarios OR conversations**
Collect and clinic, identify gaps and inconsistencies		**Potential new Use Cases**
	Revise and add precision	**Revised Use Cases with Supplementary Details**

FIGURE 9. A Process for Developing a Use Case Model that includes both team and individual work.

Note: Although not everyone is a skilled writer, most developers can write good use cases. It is a matter of writing and reading good use cases (and then adopting a common style). This involves practice and critical review.

XII. Tips and Techniques

We have pulled many commonsense writing guidelines from Ben Kovitz' s wonderful book *Practical Software Requirements*. They were either paraphrased or taken verbatim from his chapter on writing. Other guidelines on what extra efforts can have big payoffs come from our experience. If you apply these principles to your writing of use cases and other technical writing, your readers will be the beneficiary of your efforts.

Broad Principles

GUIDELINE: Read other people’ s writing. If your own documents are hard to understand, you don’t notice because you already know what it’ s supposed to say.

Writing is a craft. If writing is a large part of your job, people will judge you not on the basis of your thinking, but on the basis of your writing.

GUIDELINE: Write for human beings.

- Is there a way to express this that would be easier to understand?
- Am I overloading the reader with too much information at once? Should I provide some sort of roadmap, or break it up into smaller sections or smaller sentences?
- Which details are more important to my readers and which are less important? How can I make clear which details are which?
- Is this statement too abstract for my readers to understand without illustration? Are these details too narrow and disconnected for my readers to understand without explaining the underlying principle common to them all?
- What reasonable misinterpretations could my readers make when reading this passage?
- Will my readers see any benefit from reading this section? How does it relate to my specific reader’ s job? Does anyone have a reason to care about this? Will people see this as a waste of time?
- What is the feel of the writing?
- Is the document boring? Would anyone want to read it?

GUIDELINE: Choose the best alternative for expressing your thought, despite the rules.

GUIDELINE: When you have information that can be presented in a list, it is usually the best way. People like lists

GUIDELINE: Choosing the way to say something should derive from the content.

GUIDELINE: use a consciously designed organization for your document. Then there is “ a place for every detail, every detail in its place.”

GUIDELINE: Reinforcement makes a document understandable. Illustrations, overviews, section headings. Repetition, on the other hand, is decoy text.

Decoy Text

GUIDELINE: Avoid metatext. Text that describes the text that follows.

GUIDELINE: Avoid generalities.All information in a requirements document should be specific to the software to be built.

GUIDELINE: Avoid piling on words or explanations.

Remove clutter at all levels. You can clutter sentences, words, paragraphs, or sections of documentation with extra meaningless words. Overbearing templates also contribute to clutter.

An example:

Piling on: Business Use Case

Clutter Removed: Use Case

Another example:

Piling on: Requirements Specification Document

Clutter Removed: Requirements

GUIDELINE: Keep extraneous documents out of your requirements document. Schedules, acceptance criteria, traceability matrices, feedback forms, etc.

Avoiding Common Mistakes

GUIDELINE: Put related material together. Avoid making your document a jigsaw puzzle.

GUIDELINE: Don't mix requirements with specification. The what with the how. Don' t confuse means with ends.

GUIDELINE: Choose the most appropriate vocabulary for expressing a requirement. Don't force fit your descriptions into inappropriate diagrams, charts, and tables just because they are "usual".

GUIDELINE: Avoid "Duckspeak" (from *1984*). Meaningless sentences expressing conformation to standards.

For example, "The order data validation function shall validate the order data."

GUIDELINE: Know the vocabulary of your readers and use it. Don't invent unnecessary terminology.

GUIDELINE: Be aware of what content you are putting in your document. Don't mix levels.

Jumping back and forth between program design, requirements, and specification will only confuse the reader.

GUIDELINE: Don't start with a table of contents taken from another document.

This is equivalent to forcing the content of one document into the table of contents of another

GUIDELINE: Use consistent terminology.

GUIDELINE: Don't write for the hostile reader. Assume the reader will try to understand.

GUIDELINE: Make the requirements document readable. If it is not, the development staff won't read it.

Poor Uses of Documentation

GUIDELINE: Avoid documentation for the sake of documentation. Don't try to make your documentation an end in itself.

GUIDELINE: Requirements documents are not written to impress the customer with double-talk.

GUIDELINE: Don't write a CYA document. In these cases, most information must be communicated to the development staff by oral tradition.

GUIDELINE: Write questions about unsolved issues.

Put them with the appropriate use case description (or with the document you are working on) to show you' re not done.

Example: Should the credit check be performed after the Order is submitted or before? What happens if credit is denied?

GUIDELINE: If you are unclear about a detail, don' t write fiction; it could become fixed.

Guidelines for each element of a Use Case Template

In addition to the above general guidelines for writing, we offer these specific guidelines for writing use cases drawn from our direct experience.

Use Case Name:

A name of some actor task to be accomplished with the system. Name it from the actor's point of view

Good Example: ***Place an Order****, or* ***Cancel an Order****, or* ***Make a Payment***
Bad Example: ***Process Order Record***
This is named from the system's point-of-view
Bad Example: ***Placing an Order***
This is not stated with an active verb

Narrative Description:

A high-level narrative paragraph describing activities of a task

Actors:

Role names of Person or External System initiating this use case

Good Example: ***bank customer***

Bad Example: ***novice user***

This is a skill level, not a role. If novices do things differently, than skilled users, then perhaps their different forms of interaction might be described… but the role is user (not novice or skilled user)

Context:

A description about the current state of the system and the actor
Good Example: ***The bank customer is a primary user***
Bad example: ***The customer wants cash***
So what? Expressing desires clutter our descriptions. Always assume actors want to accomplish some goal, and that the system is ready to respond. Don't state the obvious.
Bank customer: ***The bank customer is logged on***

This is obvious. Don't state the obvious. It adds clutter.

Level:

Is it Summary, Core, Supporting or Internal?

Example:

Place Order (summary)

Order Long Distance Phone Service (core)

Enter Customer Address (supporting)

Obtain Secure Connection (internal)

Preconditions:

Anything significant about the system that must be true. Usually stated in terms of key concepts and their states.

Good Example: ***A bank customer's account is in good standing***

This must be true before he can make a withdrawal

*Bad Example****: The bank customer is logged in***

This is context, not something true about the state of the system

Post conditions:

Anything that has changed in the system that will affect future system responses as a result of successfully completing the use case. Usually stated in terms of key concepts and their states.

Good Example: ***The bank customer's account is overdrawn***
This means that the customer cannot make another withdrawal until the account balance is positive

Bad Example: ***The bank customer received cash***

This says nothing about how the system will respond in the future

Business Policies:

Business specific rules that are always true that must be enforced by the system.
Test for whether a policy is application specific or a business policy: Who established this policy? Was it the application designer, or was it the way we do business?

Good Example: ***Shipping dates must not fall on Sunday or holidays***

*Bad Example**: The system must determine the shipping date***

This is a statement of something the system must do, a system responsibility, not a rule that the system will enforce.

Application Policies:

Limits on the way than an application can behave.
Here's a simple test for whether a policy is application specific or a business policy: Who established this policy? Was it the application designer, or is this the way we do business?

Good Example: ***A user cannot incorrectly enter a password more than three times during a login attempt***
Bad Example: ***The password is encrypted then matched with the stored encrypted password***

This states how the system is going to validate the password, a system responsibility

Alternatives:

Deviations from a step that occur due to exceptions or decisions made by the system or actor. An alternative can either be a variation or an exception.

Variations *Optional actions for a step that are normal variations (not errors)*
Exceptions *Errors that occur during the execution of a step*
An alternative form can be written as either

- Step number. Variation or Exception Name – Brief statement of how this alternative will be handled,

Example:

Scenario: Identify Customer
1. Operator enters name
2. System finds and displays near matches

Variations:
1a. Operator enters billing address
1b. Operator enters phone number
1c. Operator enters customer address

Exceptions:
2a. No near match found—Notify operator to retry search
2b. Too many near matches found—Notify operator how many matches were found, and give option to narrow search or display matches

or, if handling the alternative warrants it:

- Step number. Reference to Use Case that describes the interactions with the system to handle the alternative

Good example:

Scenario: Identify Customer
1. Operator enters name
2. System finds and displays near matches

Exceptions:
2a. Too many near matches found—use ***Narrow Search Request***

Issues:

Questions that need to be resolved about this use case, scenario or conversation.
Issues should be stated simply. If you know who should resolve this issue, identify them.

Good Example:

Should a credit check be performed for new customer before placing orders? Should credit checking be performed if an order exceeds a certain amount? To be resolved by: John

Bad Example:

What about credit checking?

(What is meant by this question? Is it unclear exactly what the issue with credit checking is.)

Design Notes:

Design decisions that occur to you as you describe the usage

Good Examples:

If the bank does not permit automatic activation, the fields for ATM and PIN number should not be displayed*. (Hints to the application designer)*

User Beware! If the user enters an incorrect ATM PIN number, it is possible that he could be suspended from use of his/her ATM. We must be sure to let the user know about that error.
(Important notes about how the errors should be presented to the user—from the analyst's perspective)

Bad Example:

All errors should be reported to the user.

(Too vague. What's a designer to do with this note?)

Screens:

References to windows or web pages that are displayed during the execution of this use case

Good Examples:

Include a reference to a hand drawn "sketch" of a UI or a mock-up (this is good in early prototyping).
Include a prototype screen "captured" off the display. Label important important elements where information is gather and/or presented, and important user actions occur.

Bad Example:

Include detailed screens after they are implemented
(Too specific. What's the point of showing this level during requirements?)

Priority:

How important is this?

Frequency:

How often this is performed?

Good Example: ***200 times a month***
Bad Example: ***200 times*** (What' s the unit of time?)