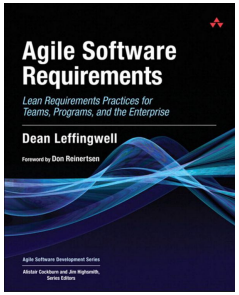
CE Department

Software Requirements Engineering

40688

These slides are designed to accompany Agile Software Requirements (2011) by Dean Leffingwell and support the university course Software Requirements Engineering, instructed by Mehran Rivadeh. Created and designed by Mahnaz Rasekhi.



Agile Software Requirements (2011)
Dean Leffingwell

Use Cases

Chapter 19

Mehran Rivadeh
mrivadeh@sharif.edu
Software Requirements Engineering
October 2025 - Fall 1404 - SUT

Contents

Introduction

Introduction

Use Case History

- ➔ It is a form of requirements capture and expression.
- ➔ Popularized **originally** within the context of the Rational Unified Process (**RUP**).
- ➔ Even outside RUP, they appeared in most contemporary works on software requirements and systems analysis.
- ➔ Use cases were also the container for **functional requirements** capture, analysis, and behavioral specification within the context of the Unified Modeling Language (**UML**).
- ➔ If you have been doing **iterative development**, you are probably using use cases too.

Introduction

In Agile Development

- The picture changed as the user story became the predominant form.
- Their value in lightning requirements expression, driving more and more incremental thinking, and generally increasing the agility of the teams that use them.
- User stories are good requirements tools.
- Use cases were largely banned from the agile tribes.
- One Certified Scrum Product Owner course stated, “Don’t use use cases. They are too hard to write, and users don’t understand them.”

Introduction

- As agile methods started to be applied to **larger systems**, however, something was **missing**: **context**.
- Simply put, although a nice itemized list of backlog items is easy to look at, tool, prioritize, and manage, it is inadequate to do the more **complex analysis** work that **larger systems require**.
- And even though we've called our backlog items **user stories**, they **don't** really **tell much of a story**.
- So, how to apply use cases in the development of complex software systems being developed in an agile manner?

Contents

1. The Problems With User Stories And Backlog Items

The Problems With User Stories And Backlog Items

- User stories and backlog items **don't give** the designers **a context to work** from:
  - When is the user doing this?
  - What is the context of their operation?
  - What is their larger goal at this moment?

The Problems With User Stories And Backlog Items

- User stories and backlog items **don't give** the project team **any sense of scope** or potential "**completeness**".
  - A development team estimates a project at (e.g.) 270 story points, and then as soon as they start working, that number keeps increasing, seemingly without bound. The developers and sponsors are equally depressed. How big is this project, really?

The Problems With User Stories And Backlog Items

- ➔ User stories and backlog items **don't provide** a **mechanism for** looking ahead at **upcoming work**.
  - Seeing a set of extension conditions (alternate flows) in a use case lets the analysts understand which ones will be easy and which will be difficult so they can stage the work accordingly.
  - With user stories, the extension conditions are usually detected mid-sprint, when it is too late.

Contents

2. Five Good Reasons To Still Use Use Cases

Find Good Reasons To Still Use Use Cases

First Reason

- The list of goal names provides executives with a short summary of what the system will contribute to the business and the users.
- It also provides a project planning skeleton, to be used to build initial priorities, estimates, team allocation, and timing.
- It is the first part of the completeness question.

Find Good Reasons To Still Use Use Cases

Second Reason

- The main success scenario of the use case provides everyone with an agreement as to what the system will and will not do.
- It provides the context for each requirement, a context that is hard to get any other way.

Find Good Reasons To Still Use Use Cases

Third Reason

- The extension conditions of the use case provide a framework for investigating all the little, niggling things that somehow take up 80% of the development time and budget.
- It provides a look-ahead mechanism, so the customer/product owner/ business analyst can spot issues that are likely to take a long time to get answers for.
- The use case extension conditions are the second part of the completeness question.

Find Good Reasons To Still Use Use Cases

Fourth Reason

- The use case extension scenarios provide answers to the many detailed, tricky business questions programmers ask:
  - "What are we supposed to do in this case?"
    - Normally answered by, "I don't know, I've never thought about that case".
- It is a thinking/documentation framework that matches the "if . . . then . . . else" statement that helps programmers think through such issues.

Find Good Reasons To Still Use Use Cases

Fifth Reason

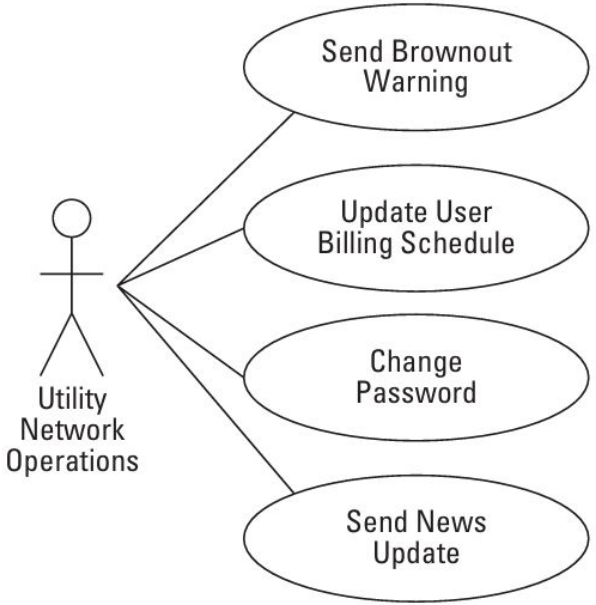
- The full use case set (use case model) shows that the developers/analysts have thought through
  - Every user's needs
  - Every goal they have with respect to the system
  - Every business variant involved
- This is the final part of the completeness question.

Contents

3. Use Cases Basics

Use Case Basics

- A use case describes a sequence of actions between an actor and a system that produces a result of value for that actor.
- Example: Use cases for the utility network operations center



Use Case Basics

- A use case describes **a sequence of actions** between an actor and a system that produces a result of value for that actor.

A Sequence of Actions

- A set of interactions between the actor and the system.
- It is invoked when the actor provides some input to the system.
- Each action is atomic; that is, it is performed either entirely or not at all.

Use Case Basics

- A use case describes a sequence of actions between an actor and a **system** that produces a result of value for that actor.

System

- The system works for the actor.
- It executes some function, algorithmic procedure, or other activity.
- The system takes its orders from the actor as to when to do what.

Use Case Basics

- A use case describes a sequence of actions between an actor and a system that produces **a result of value** for that actor.

A Result of Value

- Like a user story, the use case must deliver value to a user.
- Therefore, the resident pushes the opt-in button is not a good use case (the system didn't do anything obvious for the user).
- But the resident pushes the "opt-in" button and the system starts to shed load is a meaningful use case.

Use Case Basics

- A use case describes a sequence of actions between an actor and a system that produces a result of value for that **actor**.

Actor

- The particular actor is the individual or device that initiates the action.
- Mark, the resident; a message from the utility shed some load; create a message on the user's TV.

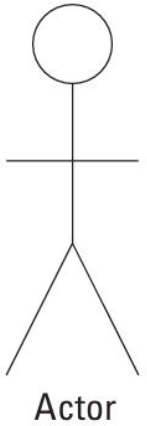
Contents

3. **Use Cases Basics**

Use Case Actor

Use Case Actor

- An actor is someone or something that interacts with the system.
- There are generally three types of actors:
  - Users
  - Other system or applications
  - A device



Use Case Actor

Users

- Users act on the system; this is the type of actor most people think of when they think of a use case.
- For example, I am an actor on the word processing system I'm using to write this chapter.

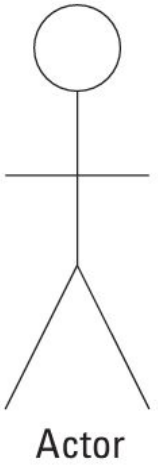
Other Systems or Applications

- Most software interacts with other systems or other applications. This is another primary source of actors.
- Here's an example: The author's word processing application interacts with a Web service to access and insert clip art.
- The author's word processing application is an actor on the Web service.

Use Case Actor

A Device

- ➔ Many applications interface to a variety of input and output devices.
- ➔ For example, the consumer's refrigerator is an actor on the Tendril platform.



Contents

Use Case Structure

Use Case Structure

The use case itself is a text-based structure of logical elements that work together to define the use case.

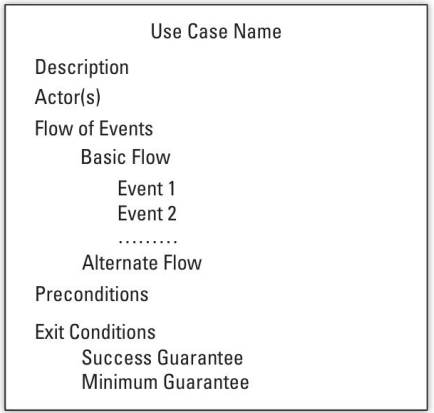
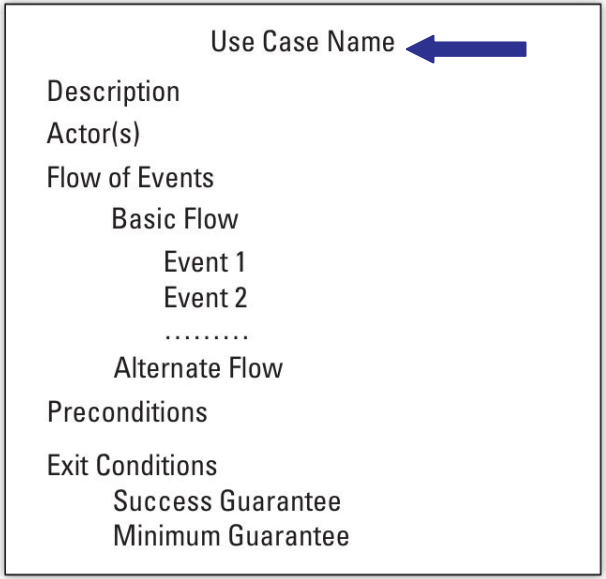


Figure 19–1 Use case template

Use Case Structure

Name

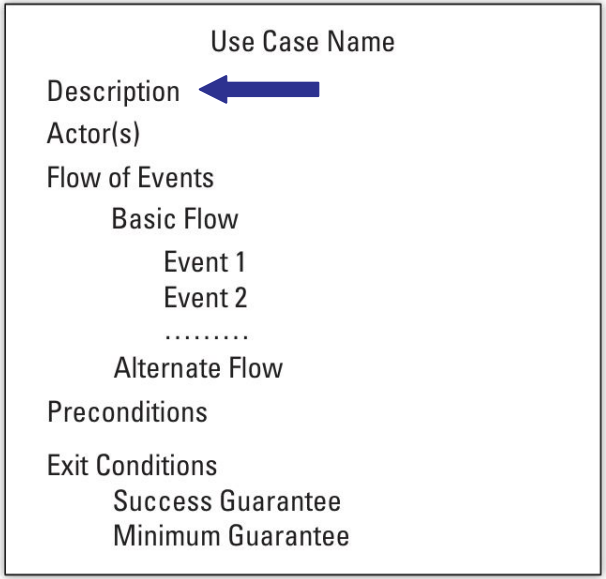
- The name describes the goal, that is, what is achieved by the interaction with the actor.
- The name should be a few words in length, and it must be unique.
- Names such as “pop up an emergency warning message” and “shed refrigerator load” are good examples.
- They are short, are descriptive, and define the goal.



Use Case Structure

Brief Description

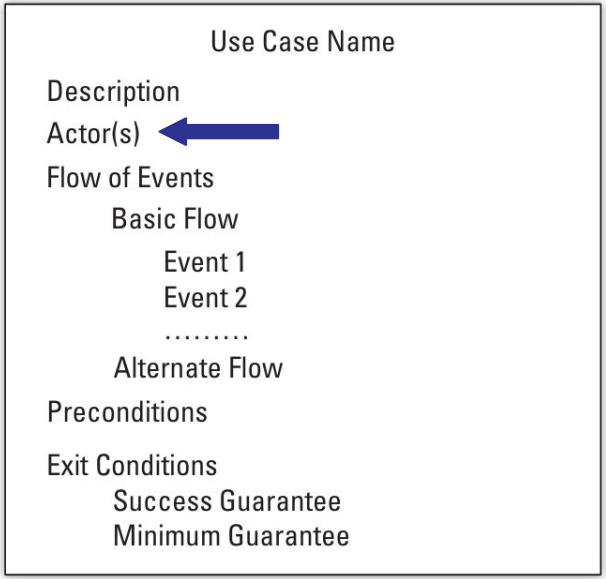
- The purpose of the use case should be described in one or two sentences.
- Here's an example: this use case describes what happens when the Tendril System receive an event notification from the utility.



Use Case Structure

Actor(s)

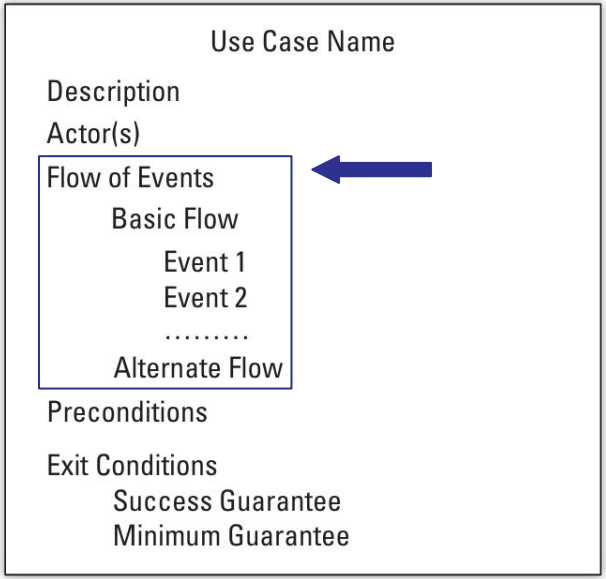
- A use case has no meaning outside the context of its use by an actor, so each actor who participates in the use case is listed with the use case.



Use Case Structure

Flow of Events

- The main body of the use case is the event flow, usually a textual description of the interactions between the actor and the system.
- **The basic flow:** It is the main path through the use case,
- **Alternate flows:** They are executed only under certain circumstances.



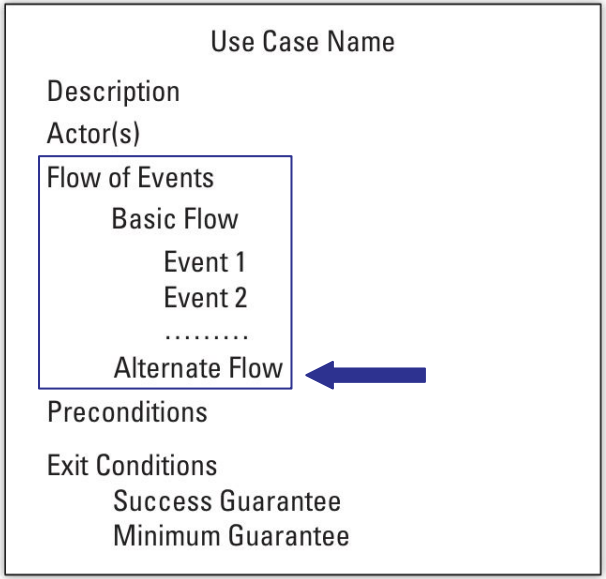
Use Case Structure

Flow of Events

- ➔ It is the alternate flows (or alternate scenarios) that provide much of value to the agile system builder.
- ➔ It is here where they are forced to think through all the “what ifs” that might affect our user story.

Example:

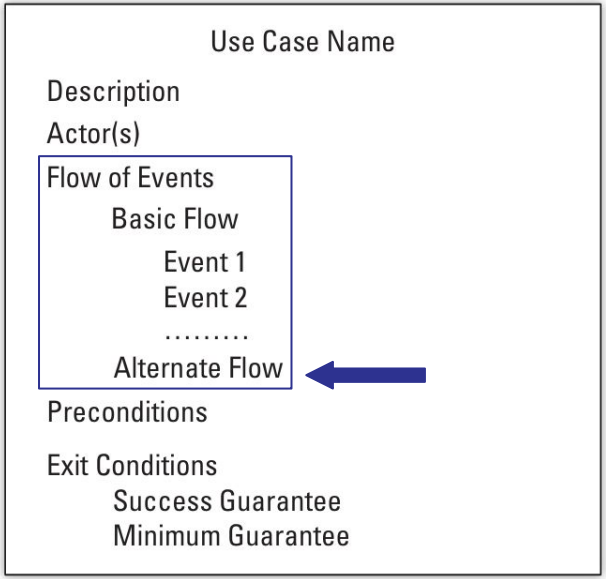
- ➔ “what happens if the device does not respond to the message?”
- ➔ “If I’m programming the system, and an event happens_what happens then?”



Use Case Structure

Flow of Events

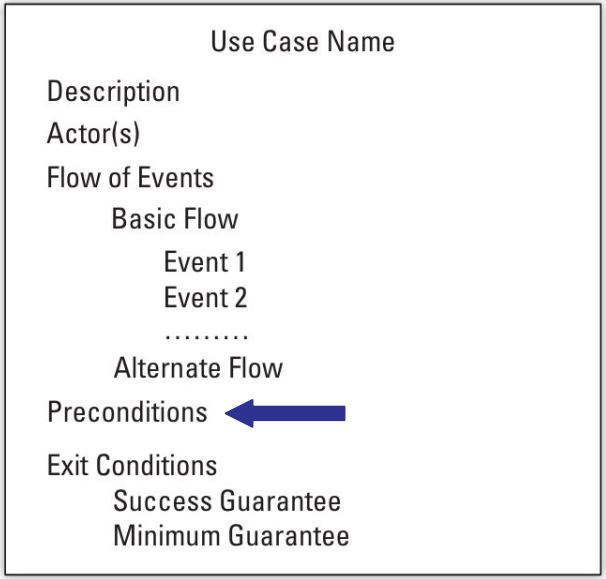
- Understanding all the alternate flows of a use case defines the various (usually less likely but equally important) scenarios that the system must handle with grace in order to assure reliability and quality.



Use Case Structure

Preconditions

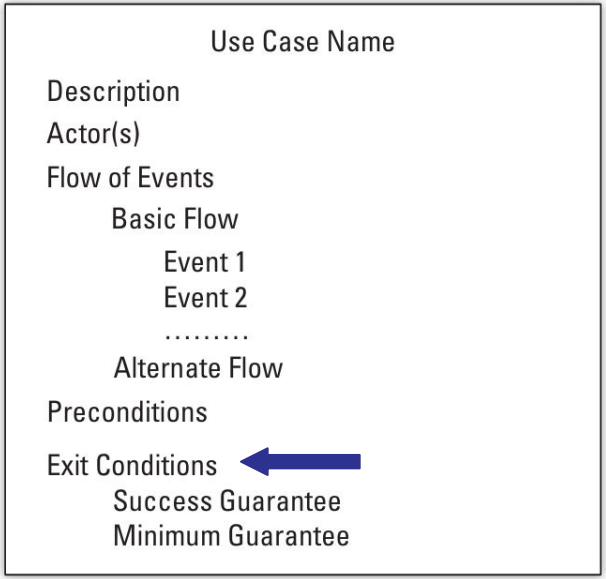
- ➔ They are those conditions that must be present in order for a use case to start.
- ➔ They usually represent some system state that must be present before the use case can be used.
- ➔ For example, a precondition of the set thermostat to shed load use case is that the system must have opt-in for load shedding enabled.



Use Case Structure

Exit Conditions

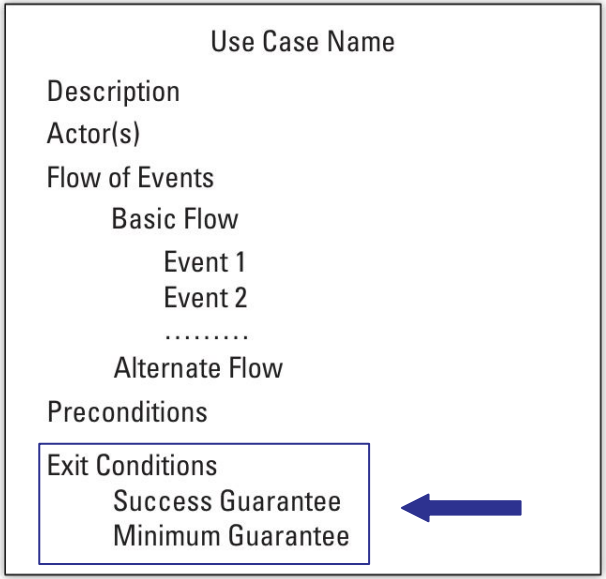
- They describe the state of the system after a use case has run its course.
- These can include both a success guarantee, which describes the state of the system after a successful execution, and a minimum guarantee, which describes the state of the system if the execution of the use case fails for some reason.



Use Case Structure

Exit Conditions (Cont.)

- For example, a success guarantee of the set thermostat to shed load use case is that the system remains in the load shedding state.
- A minimum guarantee might be that load shedding is not initiated but an error message is displayed.



Use Case Structure

System or Subsystem

- In a system of subsystems, it may be necessary to identify whether a use case is a system-level use case (one that causes multiple subsystems to interact) or a subsystem use case.
- In either case, the team needs to identify what system or subsystem a use case is identified with.

Use Case Structure

Other Stakeholders

- ➔ It may also be useful to identify other key stakeholders who may be affected by the use case.
- ➔ For example, a manager may use a report from the system, and yet the manager may not personally interact with the system and would therefore not appear as an actor.

Use Case Structure

Special Requirements

- The use case may also have special requirements that apply to that specific use case.
- Often these are nonfunctional requirements (support 10,000 homes without performance degradation) that apply to the specific use case.

Contents

3. **Use Cases Basics**

A Guide To Building The Use Case Model

- An individual use case describes how a particular actor interacts with the system to achieve a result of value for that specific actor.
- The set of all use cases together describes the complete behavior of the system.
- The complete set of use cases, actors, and their interactions constitutes the use case model for the system.
- Building the use case model for the system is an important analysis step, one that will become the basis for understanding, communicating, and refining the behavior of the system over the course of the project.

A Guide To Building The Use Case Model

We have found a simple five-step approach to be an effective way to build the use case model.

1. Step 1: Identify and Describe the Actors
2. Step 2: Identify the Use Cases
3. Step 3: Identify the Actor and Use Case Relationships
4. Step 4: Outline the Flow of the Use Cases
5. Step 5: Refine the Use Cases

A Guide To Building The Use Case Model

Step 1: Identify And Describe The Actors

1. Who uses the system?
2. Who gets information from this system?
3. Who provides information to the system?
4. Where is the system used?
5. Who supports and maintains the system?
6. What other systems or devices use this system?

A Guide To Building The Use Case Model

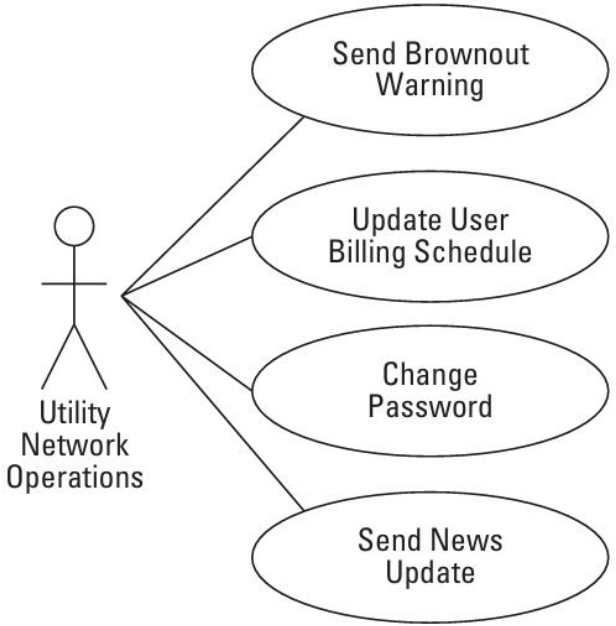
Step 2: Identify The Use Cases

Once the actors are identified, the next step is to identify the various use cases that the actors need to accomplish their jobs.

1. What will the actor use the system for?
2. Will the actor create, store, change, remove, or read data in the system?
3. Will the actor need to inform the system about external events or changes?
4. Will the actor need to be informed about certain occurrences in the system?

A Guide To Building The Use Case Model

Example: Use cases for the utility network operations center



A Guide To Building The Use Case Model

- Use case has a name that represents the goal of the use case.
- The name is a few words or short phrase that starts with an action verb and communicates what the actor achieves.
- Names help communicate what the system does and create context.
- Along with the name, there is a brief description that further describes the intent of the use case.

A Guide To Building The Use Case Model

Step 3: Identify The Actor And Use Case Relationships

- Although only one actor can initiate a use case, some use cases involve the participation of multiple actors.
- When the actors and use cases interact in concert, that's when the system becomes a system.
- In this step, each use case is analyzed to see what actors interact with it, and each actor's anticipated behavior is reviewed to verify that the actor participates in all the necessary use cases.

A Guide To Building The Use Case Model

Step 4: Outline The Flow Of The Use Cases

- The next step is to outline the flow of each use case to start to gain an understanding of required system behavior at the next level of detail.
- Of particular interest at this time is the flow of events, including the basic and alternate flows, as Figure 19–3 (next slide) illustrates.
- Outline the basic flow first. This is the flow that represents the most common path (the "**happy path**") from start to finish through the use case.

A Guide To Building The Use Case Model

Step 4: Outline The Flow Of The Use Cases

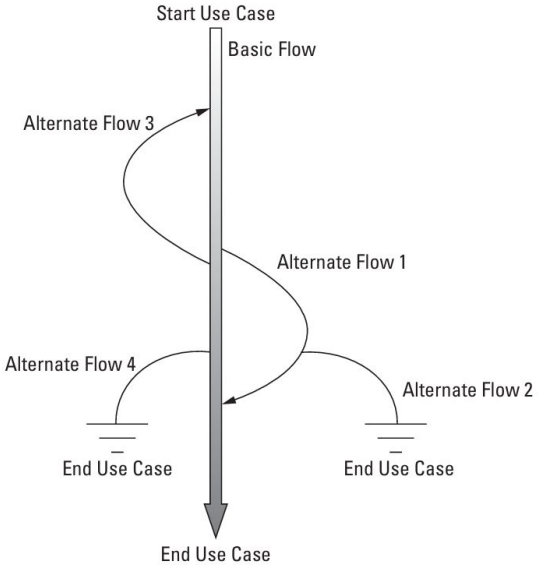
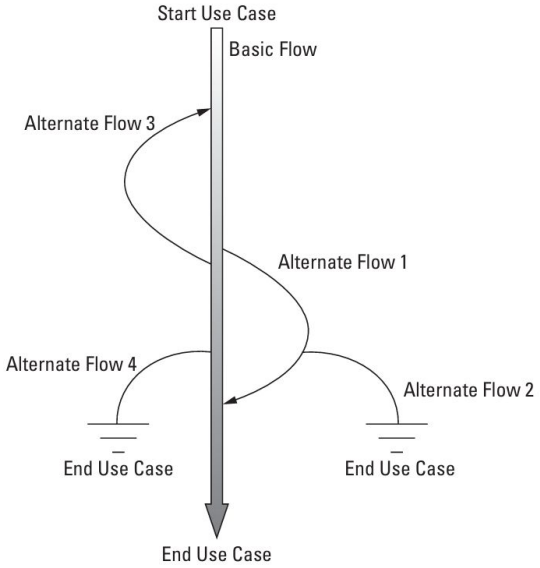


Figure 19–3 A use case has one main flow and some number of alternate flows.

A Guide To Building The Use Case Model

Step 4: Outline The Flow Of The Use Cases

- In addition to the basic flow, the use case will have a number of alternate flows based on both regular circumstances and exceptional events.

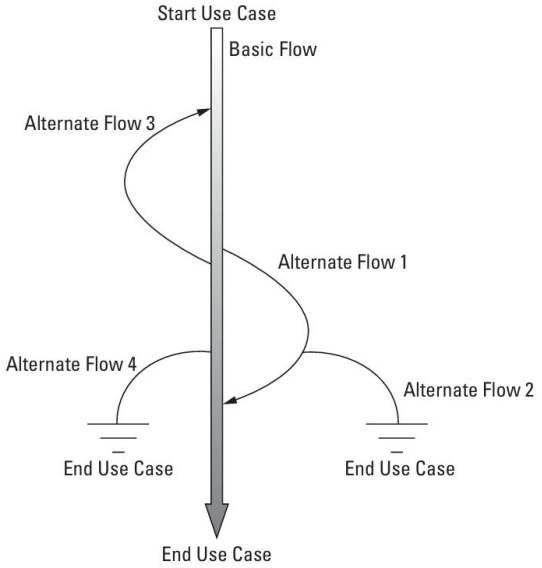


A Guide To Building The Use Case Model

Step 4: Outline The Flow Of The Use Cases

Basic flow

1. What actor's event starts the use case?
2. How does the use case end?
3. How does the use case repeat some behavior?

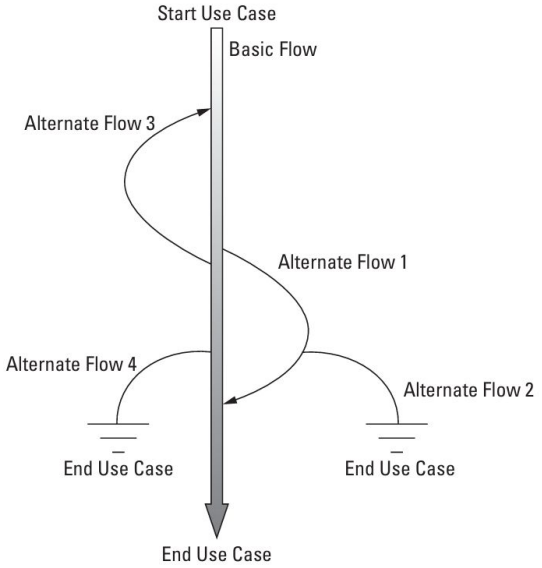


A Guide To Building The Use Case Model

Step 4: Outline The Flow Of The Use Cases

Alternate flow

1. What else can the actor do?
2. How will the actor react to optional situations?
3. What variants might happen?
4. What exceptions to the usual behavior may occur?



A Guide To Building The Use Case Model

Step 5: Refine The Use Cases

Later, the use case may be refined to another level of detail, or perhaps the team will move directly to the stories that implement the use case.

A Guide To Building The Use Case Model

Step 5: Refine The Use Cases

Consider all alternate flows, including unusual exception conditions:

- ➔ It is usually straightforward to identify the primary alternate flows of a use case since these are driven by explicit actor choices.
- ➔ However, the "**what ifs**" are a **primary source** of **concern**, and these must be fully explored in alternate flows. "What if the resident is programming the system when an energy event occurs?"
- ➔ All these exceptions must be understood, coded, and tested, or the application may not behave as expected.

A Guide To Building The Use Case Model

Step 5: Refine The Use Cases

Preconditions:

- The refinement process will identify state information that controls the behavior of the system.
- Preconditions describe the **things** that **must be true before** a use case **starts**.
- For example, a precondition to programming vacation settings might be that the user has set the time zone. If that has not been done, the use case cannot be successfully executed.

A Guide To Building The Use Case Model

Step 5: Refine The Use Cases

Exit conditions:

- These describe the persistent states the use case leaves behind.
- They can include the success guarantee (state after successful execution) and the minimum guarantee (state after unsuccessful execution).

Contents

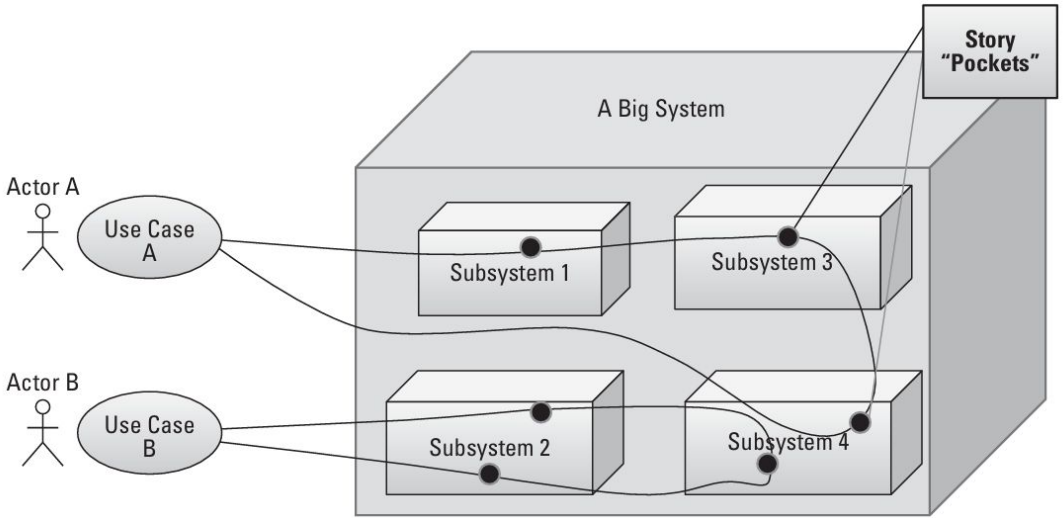
4. Applying Use Cases

Applying Use Cases

- If your **system** is **complex** and is composed of subsystems, use cases provide a mechanism to help **developers** think about **all** the **possible paths** through the system and subsystems that the **users may** encounter.
- In turn, this helps the team understand where user stories are likely to be needed to implement the required functionality.
- Every time a new use case touches a subsystem, there are probably new stories that must be developed for that subsystem.

Applying Use Cases

Use cases traversing a system identify where stories are needed.



Applying Use Cases

Once identified, the use cases can then be used to drive incremental development, one story at a time.

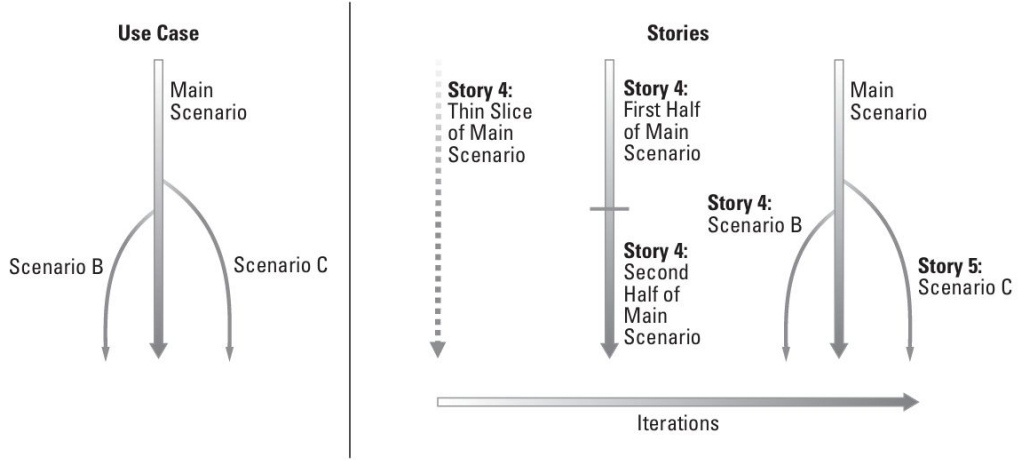


Figure 19–6 Sequencing user stories in iterations

Source: Kroll/MacIsaac, *Agility and Discipline Made Easy*, © 2006

Applying Use Cases

Tips For Applying Use Cases In Agile

When you apply use cases in agile development:

- Keep them lightweight_no design details, GUI specs, and so on.
- Don’t treat them like fixed requirements.
- Like user stories, they are merely statements of intended system behavior.
- Don’t worry about maintaining them; they are primarily thinking tools.
- Model them informally_use whiteboards, lightweight tools, and so on.

Contents

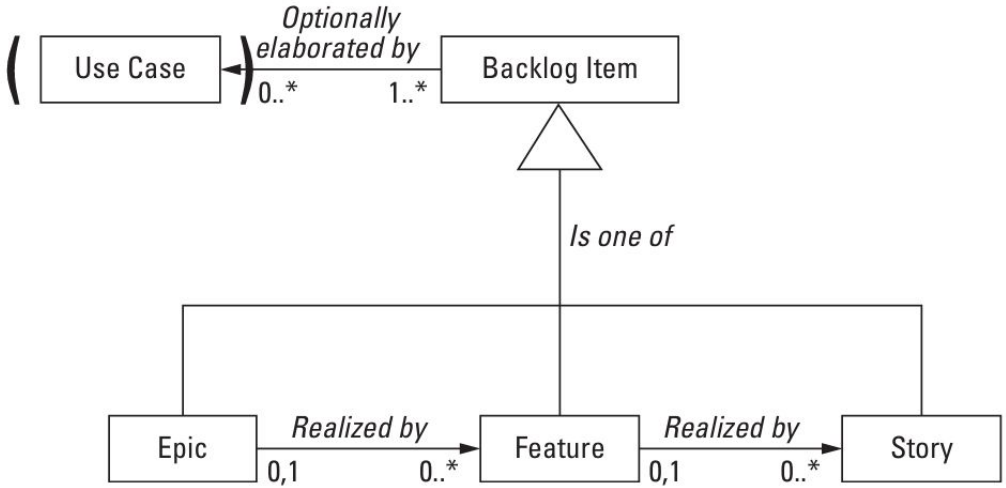
5. Uses Cases In The Agile Requirements Model

Use Cases In The Agile Requirements Format

- Use cases are an optional technique.
- Teams can use to better understand desired system behaviors.
- We haven’t yet introduced them in the agile requirements meta-model.
- Since functional system behaviors are captured as backlog items, we’ll associate use cases there, as an optional requirements modeling and elaboration artifact.
- You can apply use cases to describe more complex system behavior, most typically at the feature and epic levels, where they serve to better illustrate what we mean by that big thing.

Use Cases In The Agile Requirements Format

Use cases may be used to elaborate desired system behavior.



End of Chapter 19

Contributions

- Author of Reference Book: **Dean Leffingwell**
- Course Instructor: **Mehran Rivadeh**
- Slide Creator: **Mahnaz Rasekhi**
  - These slides are primarily based on Agile Software Requirements by Dean Leffingwell, with occasional adaptations to enhance clarity and engagement.