IVAR JACOBSON INTERNATIONAL

THE SMARTER WAY

Use-Case 2.0

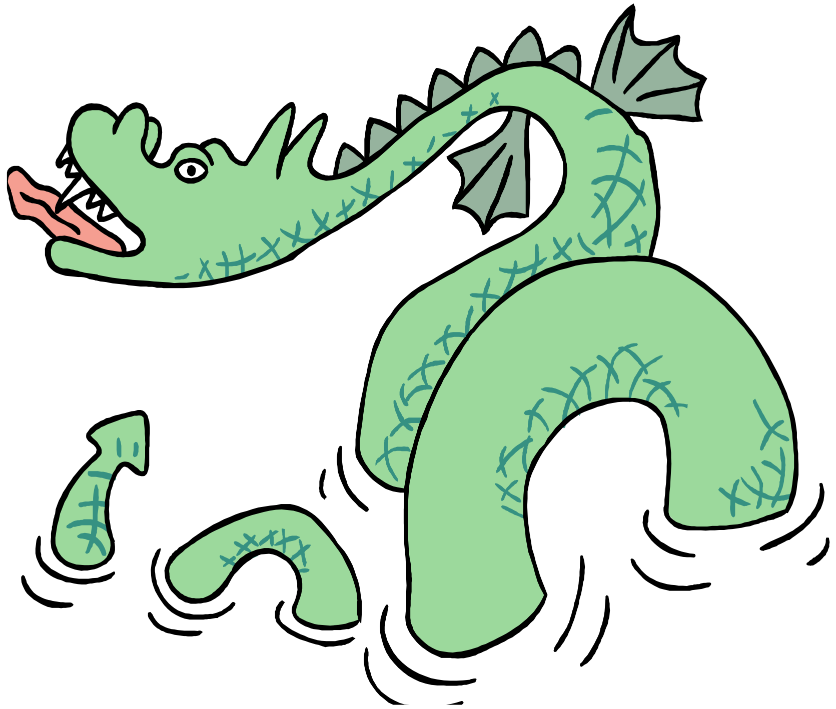
Module 7 - Adapting Your Use-Case Model

www.ivarjacobson.com

Objectives

- Understand the use-case relationships, *include* and *extend*
- Understand the appropriate use of use-case relationships
- How to avoid common use-case relationship mistakes

An introduction to use-case relationships



“If there is one thing that sets teams down the wrong path, it’s the misuse of the use-case relationships: *include* and *extend*”

“Here there be Dragons”

$$\begin{tikzcd} \cdot \arrow[r, dashed, "\textit{<<include>>}"'] & \cdot \end{tikzcd}$$

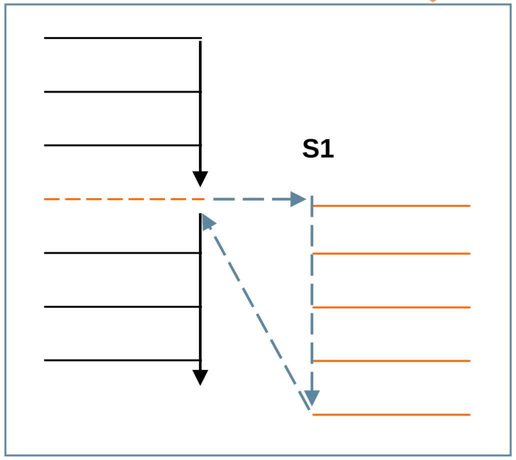
$$\begin{tikzcd} \cdot & \cdot \arrow[l, dashed, "\textit{<<extend>>}"'] \end{tikzcd}$$

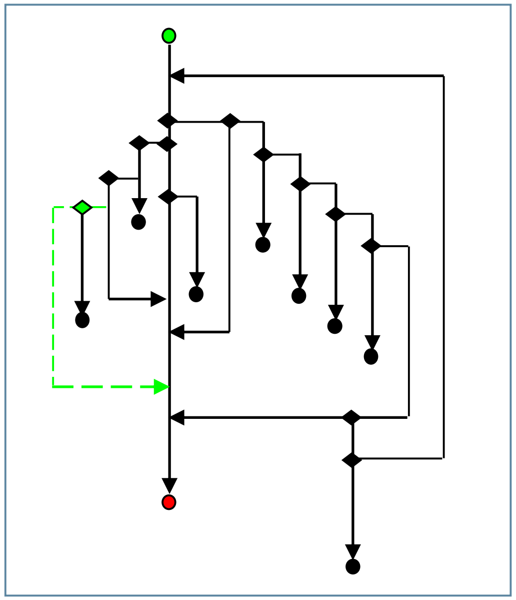
IVAR JACOBSON INTERNATIONAL

THE SMARTER WAY

Using named subflows / alternative flows to structure text

- Named subflows and alternative flows provide powerful techniques for structuring a use case's flow of events *without* resorting to use case relationships:
- These techniques should be used to their fullest before additional relationships between use cases are introduced
- Most systems can be fully described without resorting to use-case relationships



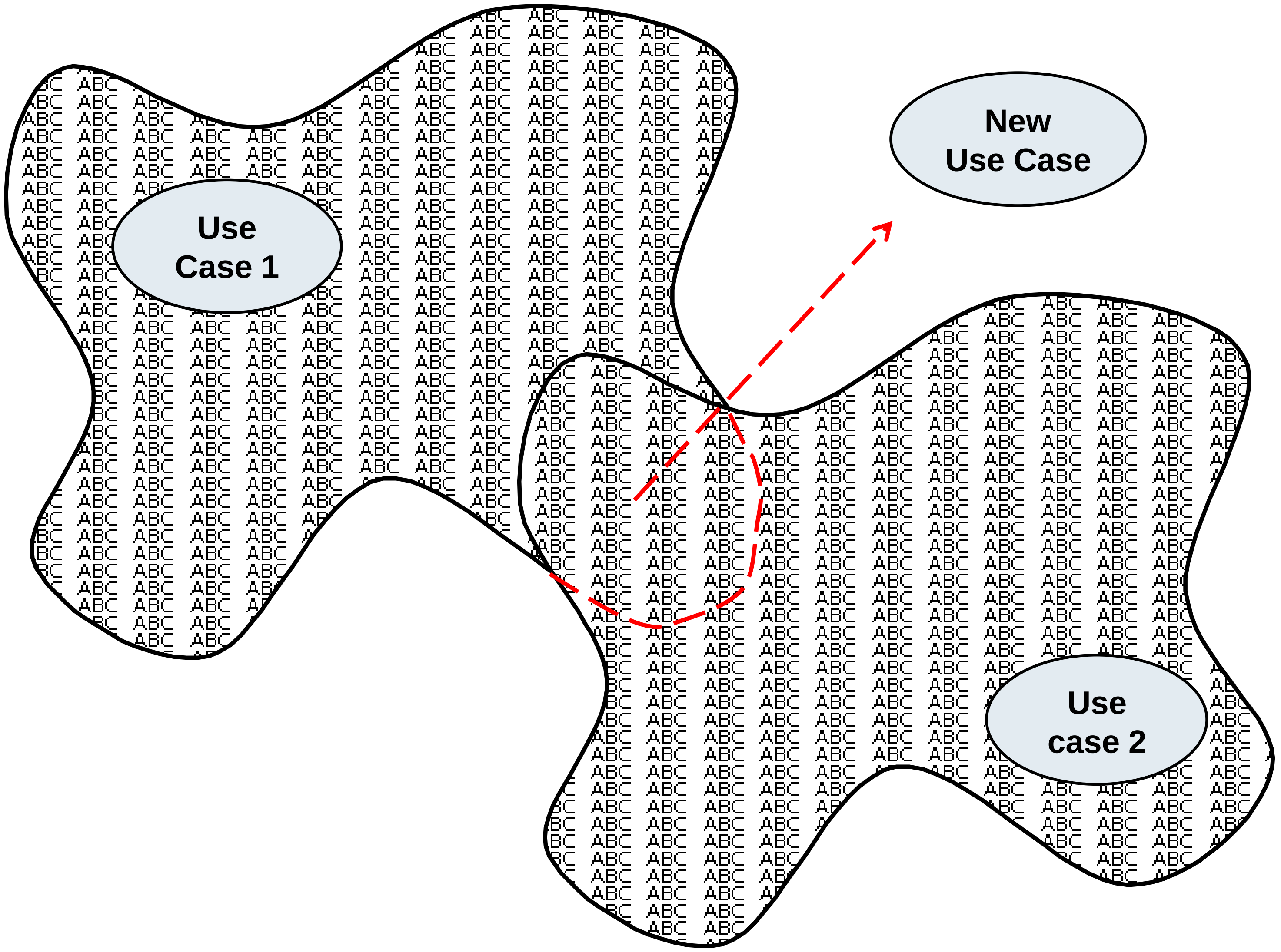


Defining Relationships Between Use Cases

- If most systems can be described without them, and most teams get into trouble when they try them, why persevere with use-case relationships?
  - Commonality of behavior between two or more use cases (*include*)
  - Adding additional behavior to an existing use case (*extend*)
  - Reducing complexity by isolating portions of use cases (*extend*)

"Never introduce relationships between use cases until you have at least a draft of the flow of events of the use cases"

Using the Include Relationship



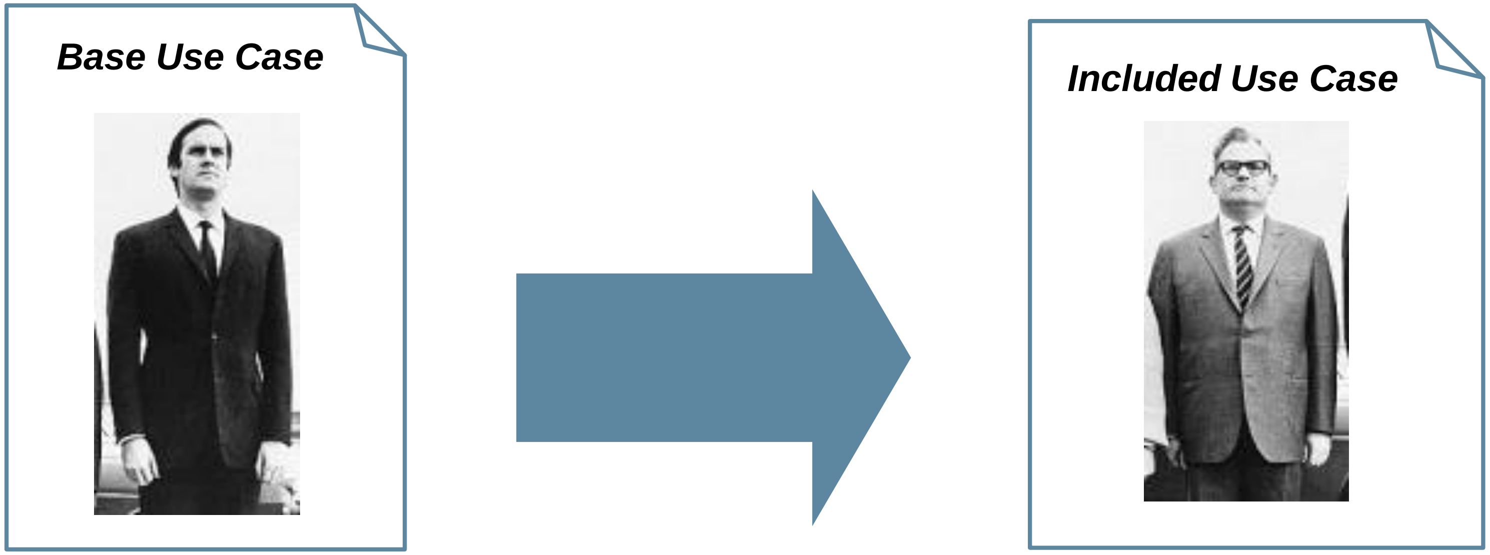
Using the Include Relationship

- The *include* relationship provides the ability to extract common sections from two or more use-case narratives
- In order to use the include relationship, you must have the same descriptive text in at least two different use-case narratives
  - Only introduce an *include* <u>after</u> a use-case narrative is written (beware of working only with diagrams)
  - Never introduce an include that is included by only one use case – totally pointless!

IVAR JACOBSON INTERNATIONAL

THE SMARTER WAY

Using the Include Relationship



I am in charge. Just as with a sub-flow, "I decide when and where I use him"

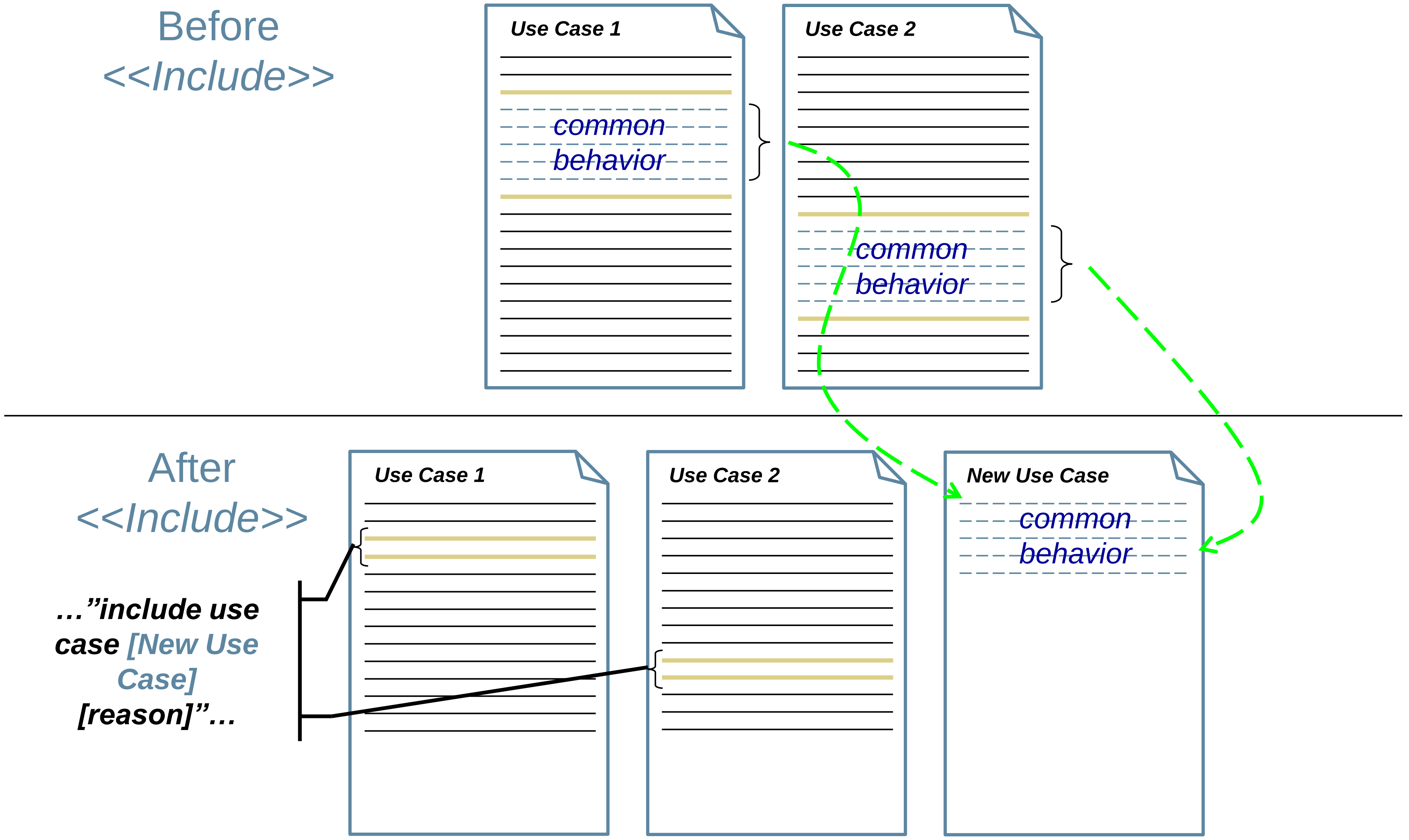
I am Just like a sub-flow, "I have no knowledge of him"

The included use case has no knowledge of the base, including use case.

Using the Include Relationship

Before
<<Include>>

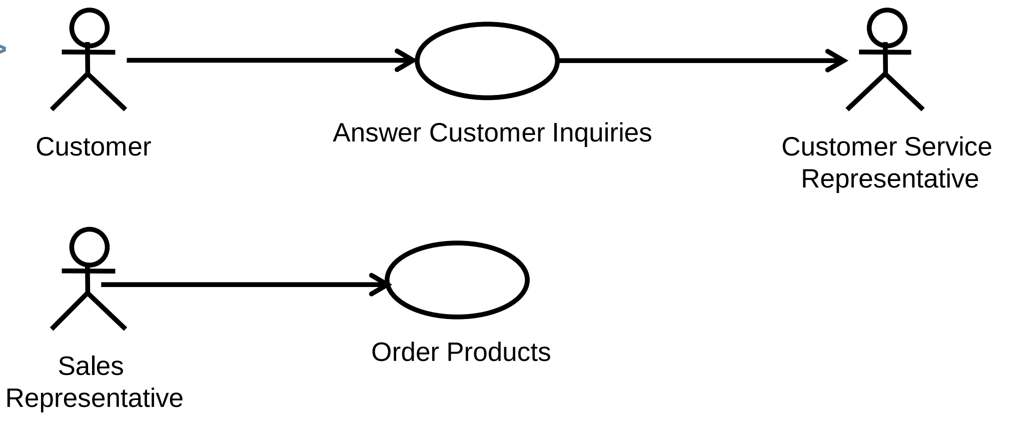
After
<<Include>>



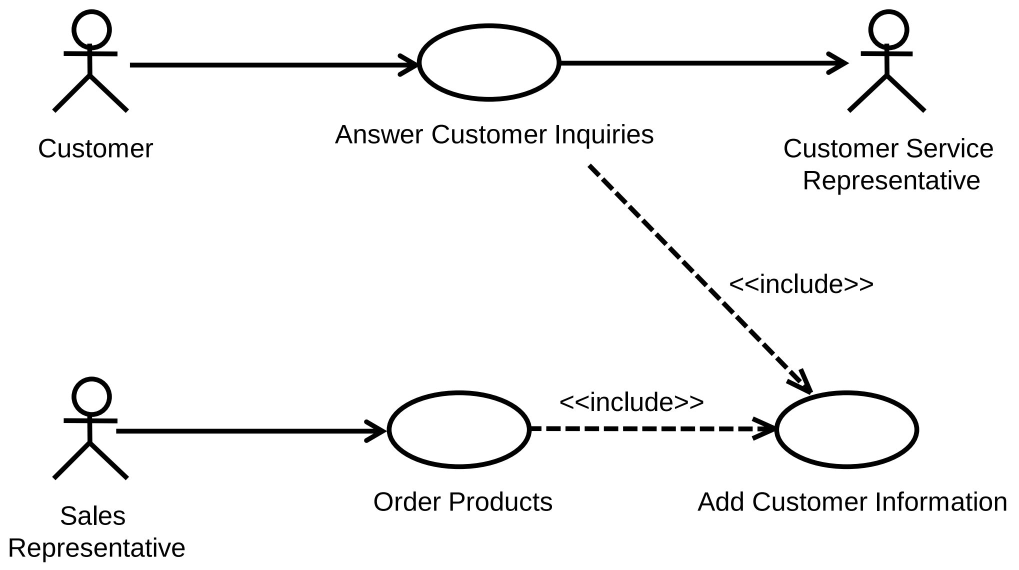
…"include use case [New Use Case] [reason]"…

Using the Include Relationship

Before <<*Include*>>

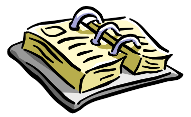


After <<*Include*>>



Discussion: Using the Include Relationship

- Find and read the Use-Case Narrative "*Include*" handout
- Observations:
  - By using an included use case, the two original use cases are simplified
  - The definition of *customer information* has been hived off to the glossary
  - In creating the new included use case, parts of both original use cases had to be re-written (not to mention the writing of the new use case)
- The included use case *never* has specific knowledge of the use case that included it
  - Hence included use cases are reusable


Handout:
Include Relationship

Common Errors Using the Include Relationship

- Used to perform functional decomposition:
  - Treating the included use case as some form of menu option
    - The including use case ends up as a shell and can no longer provide value on it's own
- The behavior in the included use case is expanded outside the context of the use cases that include it
  - Scope creep
  - When an include is used it means that the whole of the included use case is part of the including use case

Using the Extends Relationship

- The extend relationship is used where optional or exceptional behavior is inserted into an existing use case
  - The original purpose was a mechanism for specifying options that could be added to an existing product
- The extending use case needs no change to the use case it extends, possible circumstances of use:
  - Descriptions of features that are optional to the basic behavior, "optionally purchased"
  - Complex exception-handling that would otherwise obscure the primary behavior, (alternative flows that are longer than the main flow)
  - Customization of the requirements model for specific customer needs
  - Scope and release management. E.g., behavior that will not be introduced until later release

Using the Extends Relationship

Extending Use Case

Base Use Case

I know my place. Just like an alternative flow "I have detailed knowledge of the existence and the potential extension points of him"

… "But I have no knowledge of even the existence of him"

The extended, base use case has no knowledge of the extending use case.

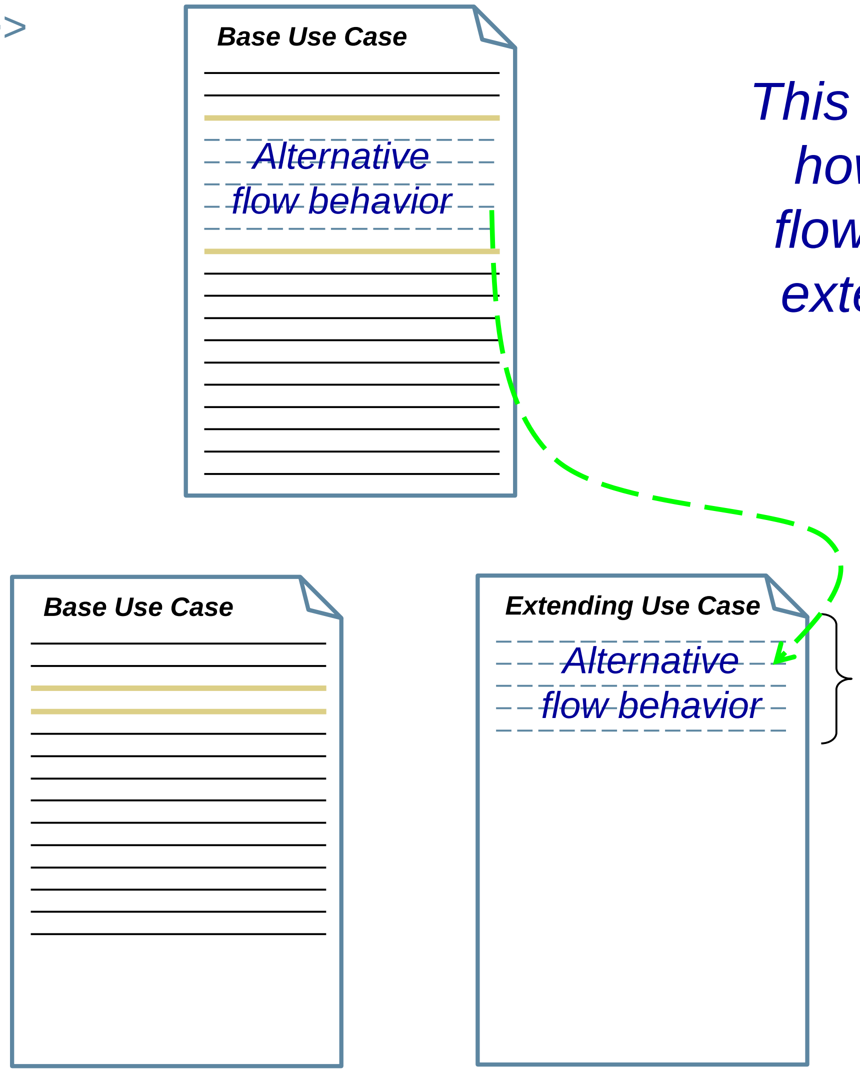
IVAR JACOBSON INTERNATIONAL

THE SMARTER WAY

Using the Extends Relationship

Before <<*extend*>>

After <<*extend*>>

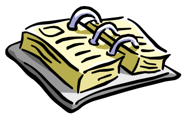


This illustration shows how an alternative flow can become an extending use case

…"*extends* use case [Base Use Case] at {extension point} where [condition]"…

Discussion: Using the Extend Relationship

- Find and read the Use-Case Narrative "Extend" handout
- Notes:
  - The extension allows us to add-on features of the system in a simple way
  - The base use case must remain intact and valuable on it's own
  - The extension cannot modify the base use case

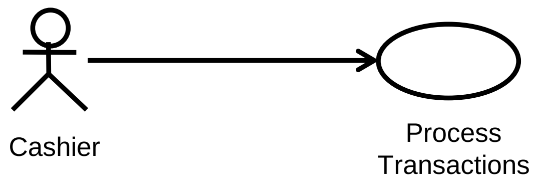


Handout:
Extend Relationship

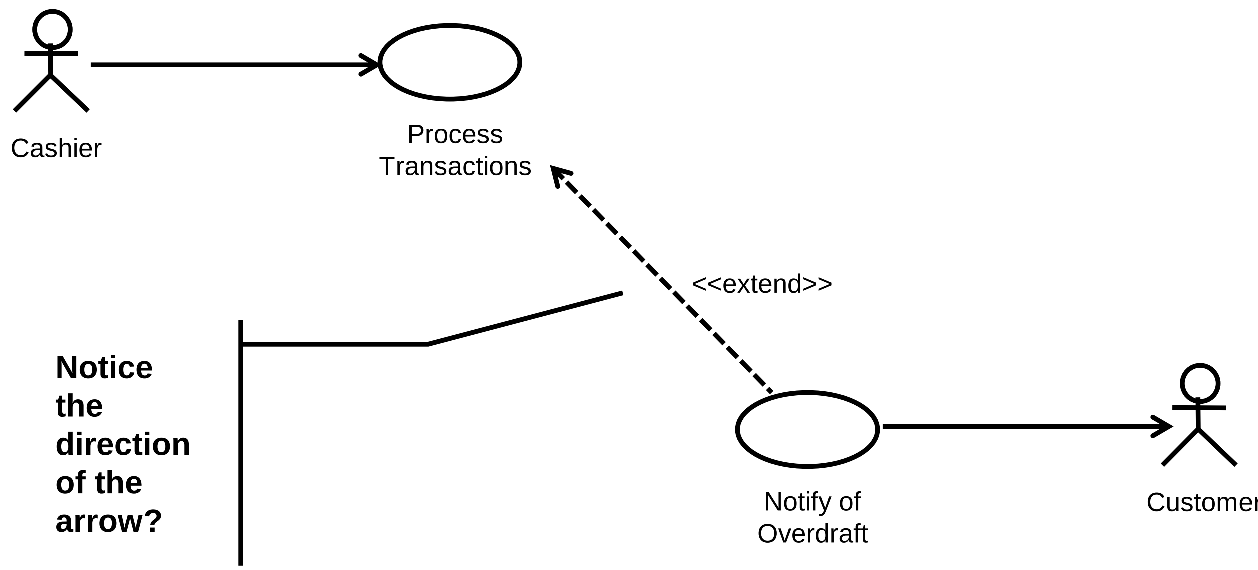
THE SMARTER WAY

Using the Extends Relationship

Before <<*Extend*>>



After <<*Extend*>>



Notice the direction of the arrow?

Making Decisions of Extension Use Case

- If an alternative flow is primarily providing behaviour that is optional, meaning that the system could be delivered without it, it could be considered a candidate for becoming an extending use case. The decision to make it an extending
  - Making it a separate use case makes it easier to manage from a versioning and configuration perspective
  - The use cases will actually be owned and maintained by different people, perhaps because different expertise is required for the extension
  - Separating it from the original use case makes both use cases easier to understand
- Extension always adds behaviour to a use case
- The basic behaviour of the use case always remains intact

IVAR JACOBSON INTERNATIONAL

THE SMARTER WAY

Extension Points Revisited

- Extension Points can be public or private
  - Private – visible only within the use case in which they occur
  - Public – visible externally to extending use cases
- The extension point mechanism is used for both alternative flows and extending use cases – {Extension Point}
- There is a section in the use-case narrative to declare public extension points

Use Case – Browse Products and Place Orders

Public Extension Points

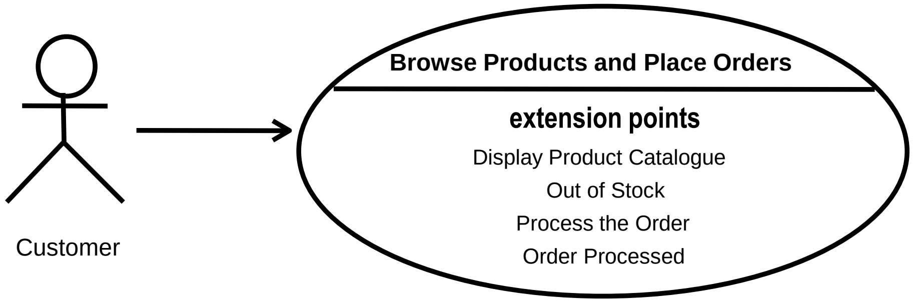
{Display Product Catalogue}
{Out of Stock}
{Process the Order}
{Order Processed}

Extension Points Revisited

- Only extension points that represent locations at which the use case can be extended should be made public
- New extension points can be declared in the Public Extension Points section of the use-case where the use-case narrative itself is under strict configuration control thus:

{extension point name}

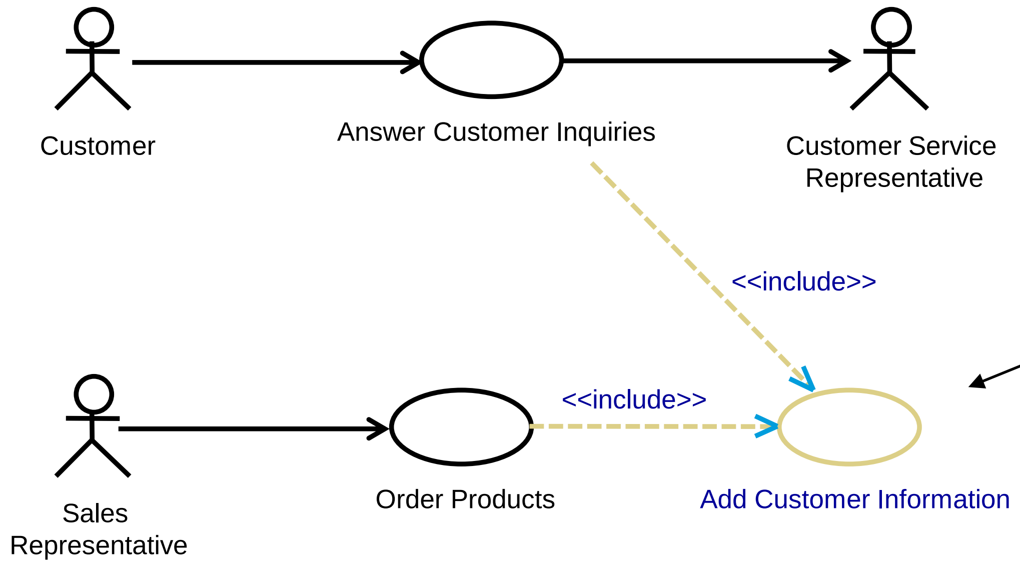
At <some location in the use-case narrative>, or before <some location in the use-case narrative>, or after <some location in the use-case narrative>



Public extension points can be shown as part of the use case on use-case diagrams, in a compartment named extension points.

Evaluating the Resulting Use-Case Model

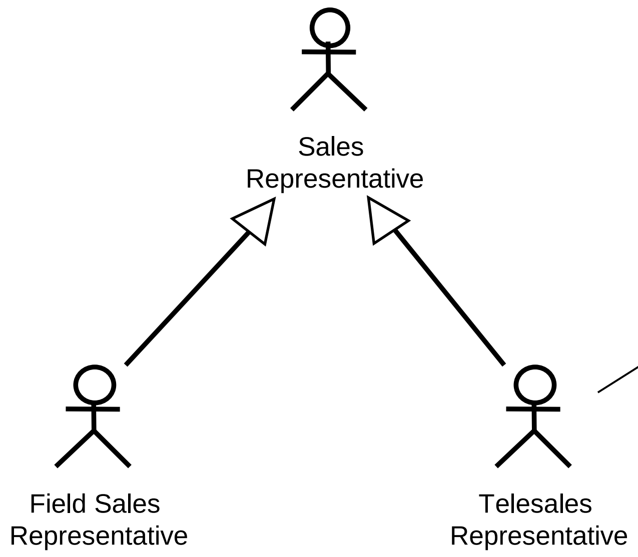
- The basic use cases of the system should reflect the value provided by the system



can you take away the "includes" and "extends" and still see the value?

Defining Relationships Between Actors

- The only relationship *between* actors is *generalization*
  - It is used to show similarity between actors
  - The main value is to show that some group of actors share common responsibilities or common characteristics



The Field and Telesales Representatives inherit characteristics from the Sales Representative, including the *communicates* relationships with other use cases

Summary

- Use-Case relationships are often misused and should be undertaken with care
- Never structure the use-case model before you have written any use-case narratives
- The *include* relationship is for situations where there is truly common behavior to more than one use case
- The *extend* relationship is primarily used to extend the behavior of an existing use case