

How to Avoid Writing Stupid Use Cases

Ian Spence

ispence@ivarjacobson.com



IVAR JACOBSON
CONSULTING

How to avoid writing stupid use cases

1. Go on holiday
2. Spend all your time attending conferences
3. Just start coding
4. Write stupid declarative requirements documents instead
5. Read use-case modeling by Kurt Bittner and Ian Spence

Thank You

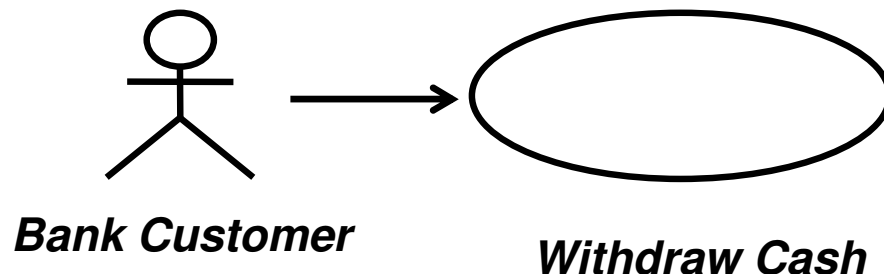
Agenda

- Useless Cases and Useless Models
- Avoiding Useless Models
- Avoiding Useless Cases
- Questions and Answers

What is a Use Case?

A use case describes a sequence of actions a system performs that yields an observable **result of value** to a particular actor

- Use cases are shown in UML diagrams



- Use cases are described in text

- They tell the story of the interactions between actors and the system



**Use Case
Specification**

Looking Inside a Use Case

- **Basic Flow**

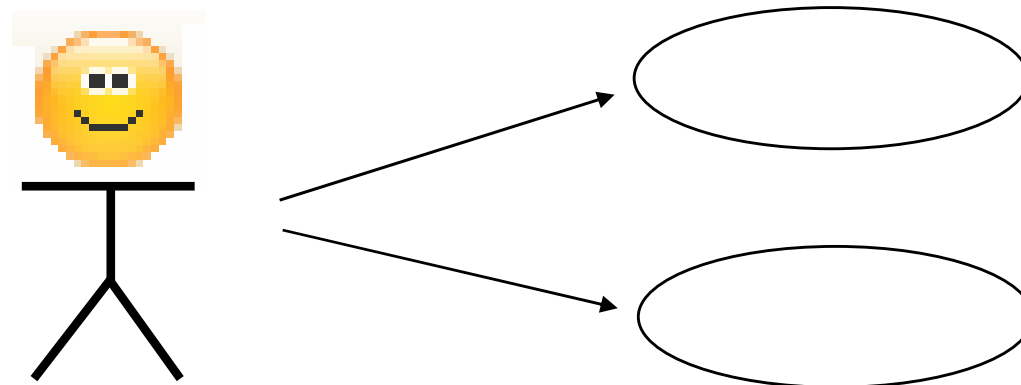
1. Insert Card
2. Validate Card
3. Select Cash Withdrawal
4. Select Amount
5. Confirm Availability of Funds
6. Return Card
7. Dispense Cash

- ▶ **Alternative Flows**

- A1 Invalid Card
- A2 Non-Standard Amount
- A3 Receipt Required
- A4 Insufficient Funds in ATM
- A5 Insufficient Funds in Acct
- A6 Would Cause Overdraft
- A7 Card Stuck
- A8 Cash Left Behind
- Etc...

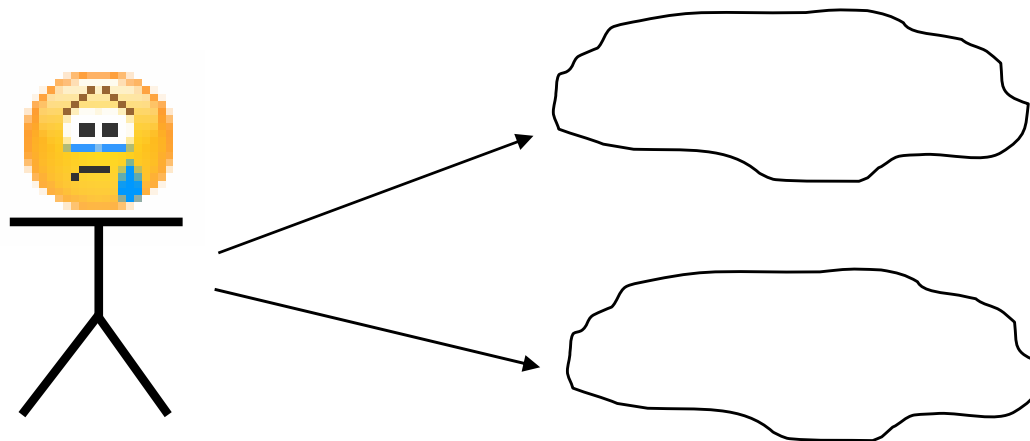
Useful use cases

- Are easy to understand
- Facilitate communication with the stakeholders
- Are shared by everyone on the project
- Capture the requirements of the system
- Support the development of the solution
 - Support the testing and development of the system
 - Support the scope management of the system
 - Support the planning of the project



Useless use cases

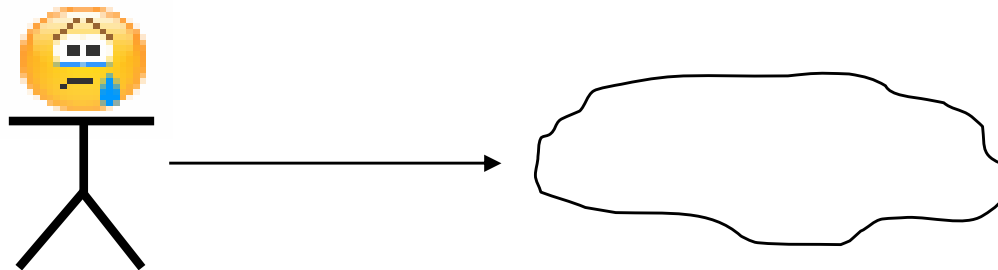
- Are either too complicated to understand or say nothing
- Alienate the stakeholders
- Are only used by the people who write them
- Contain no requirements
- Do not support the development of the solution
 - Cannot be used to support the testing and development of the system
 - Cannot be used for scope management or planning



Useless Case #1 – No Added Value

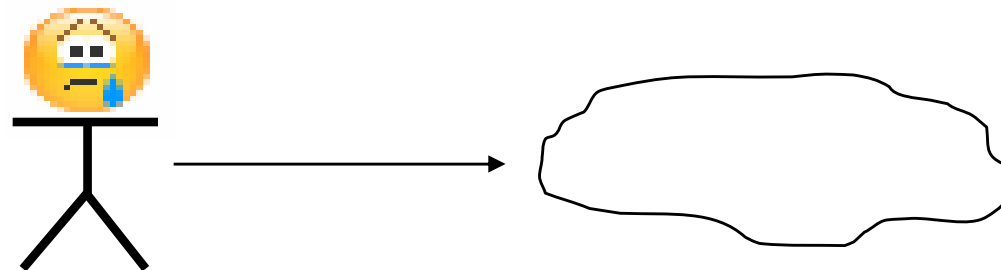
Maintain Reference Information

- The use case starts when the actor selects the maintain reference date option from the menu
- The actor selects the information to be maintained
- The system displays the information
- The actor changes the information
- The system saves the changes
- The use case ends



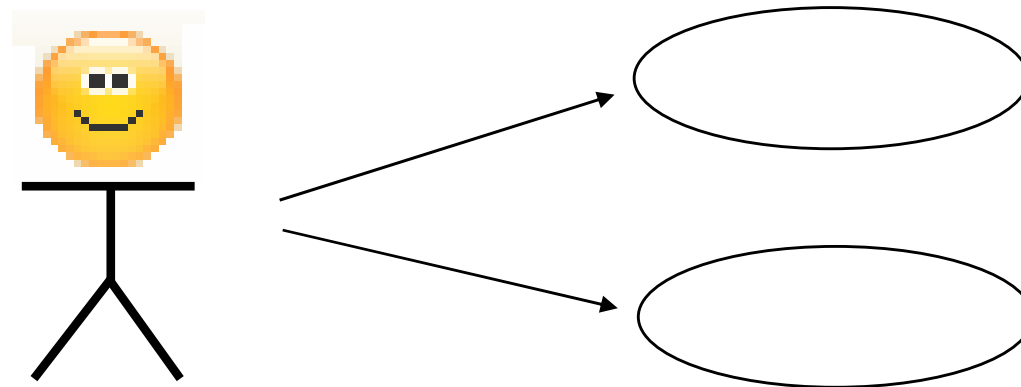
Useless Case #2 – All Screen Design

1. The use case starts when the Customer selects the browse product catalogue **menu item**. The system displays the **catalogue browsing window** with the available products displayed in a **scrolling list box**.
2. The Customer selects one or more product in the **list box**.
3. The Customer clicks on the **buy products button**.
4. For each selected item the system displays a **modal dialog** displaying the number of items in stock and an **entry field** to capture the number of items required.
 - a. The customer enters the number of items required and click the **order button**. The system records the items and quantity required, reserving them in inventory.
5. The system displays the **payment instructions capture screen**.
6. The Customer selects their method of payment from the **drop-down list** of credit card types.
7. The Customer selects the no marketing **check box**



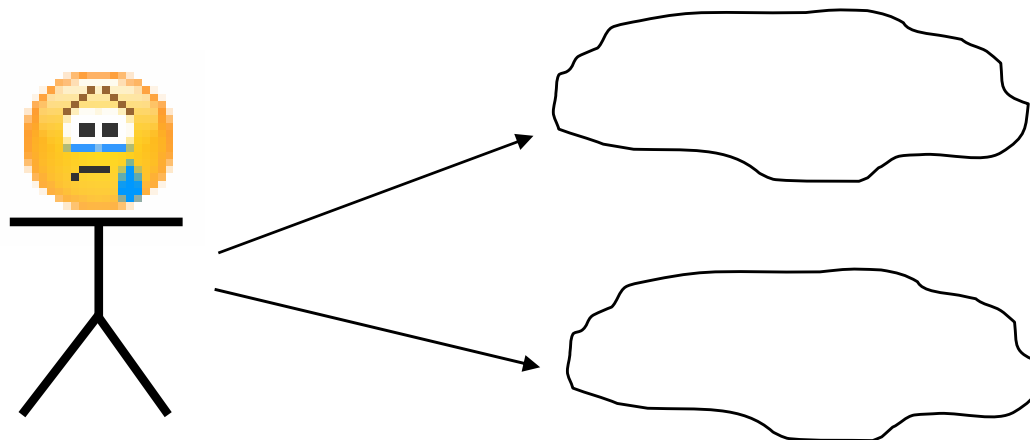
Useful use-case models

- Are easy to understand
- Facilitate communication with the stakeholders
- Are shared by everyone on the project
- Provide an overview of what the system will do
- Describe what the system will do to provide business value

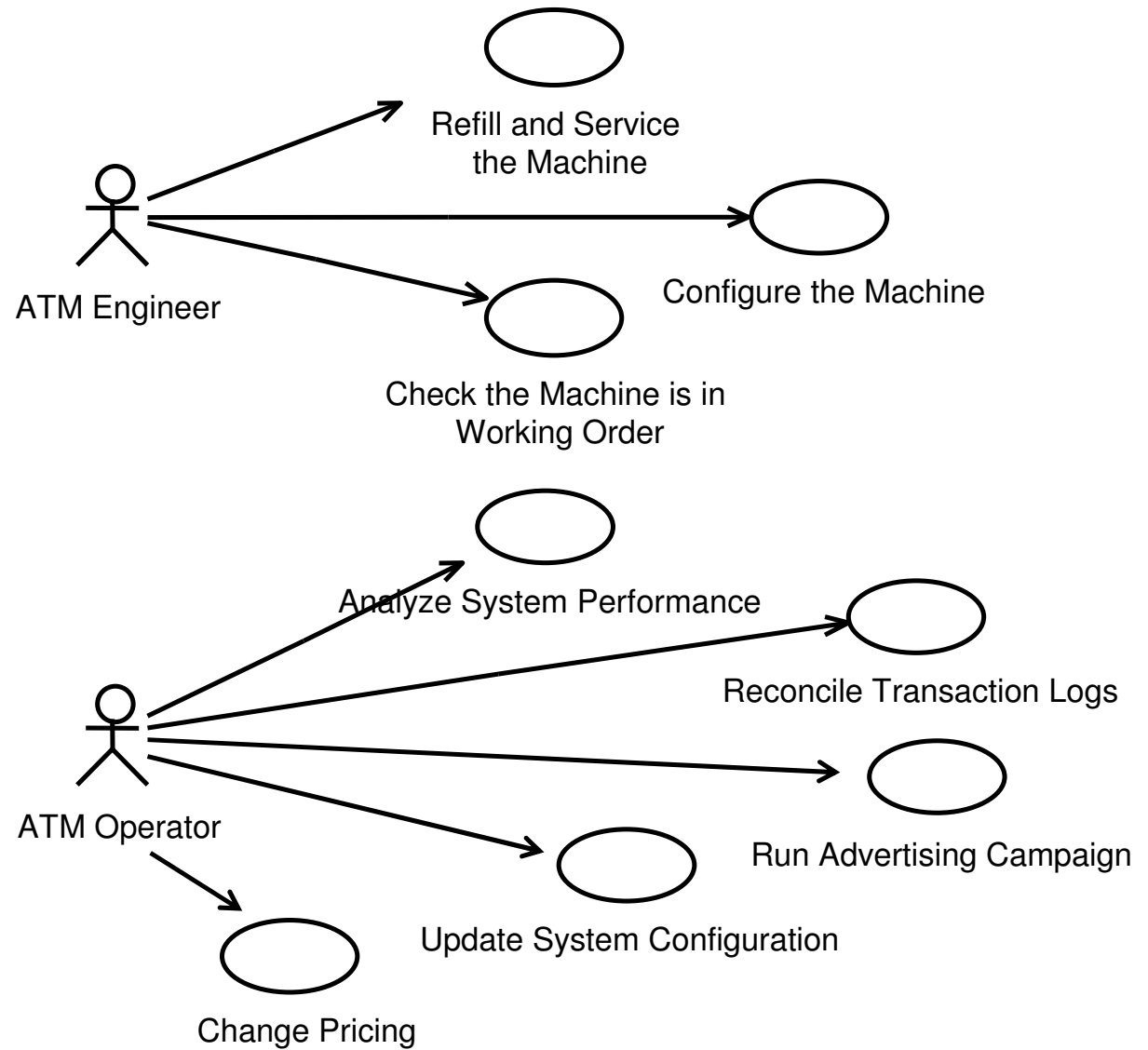
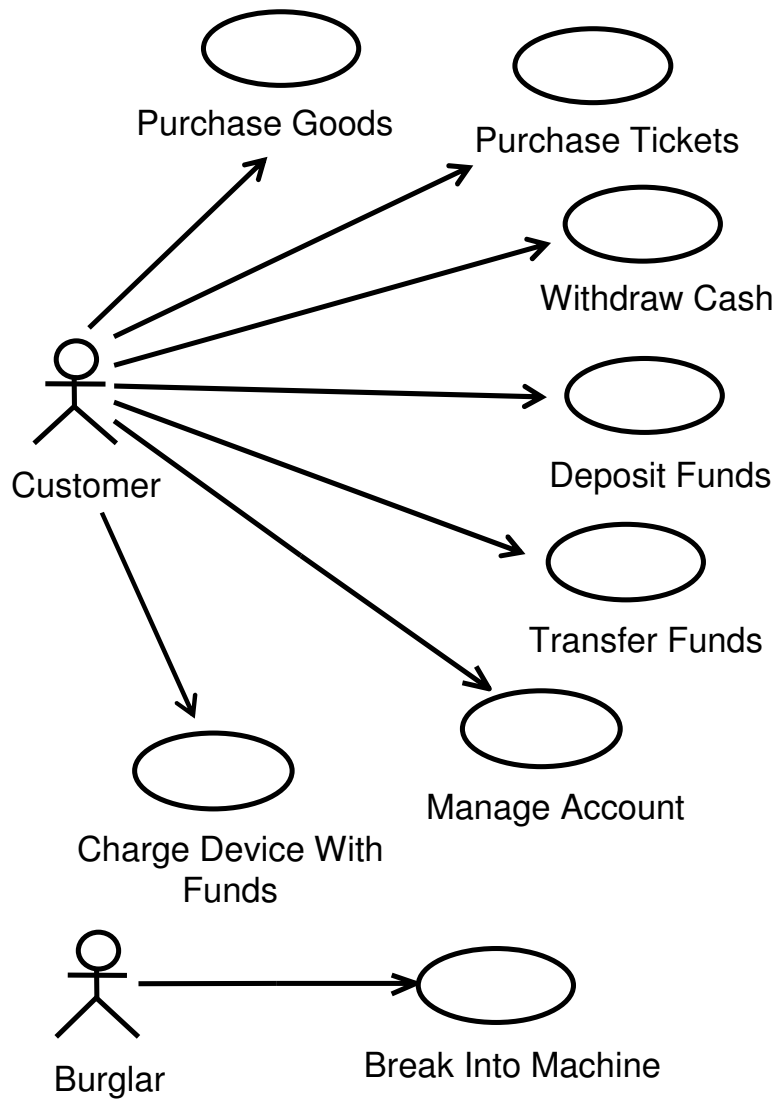


Useless use case models

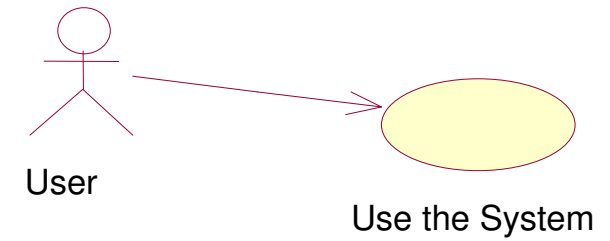
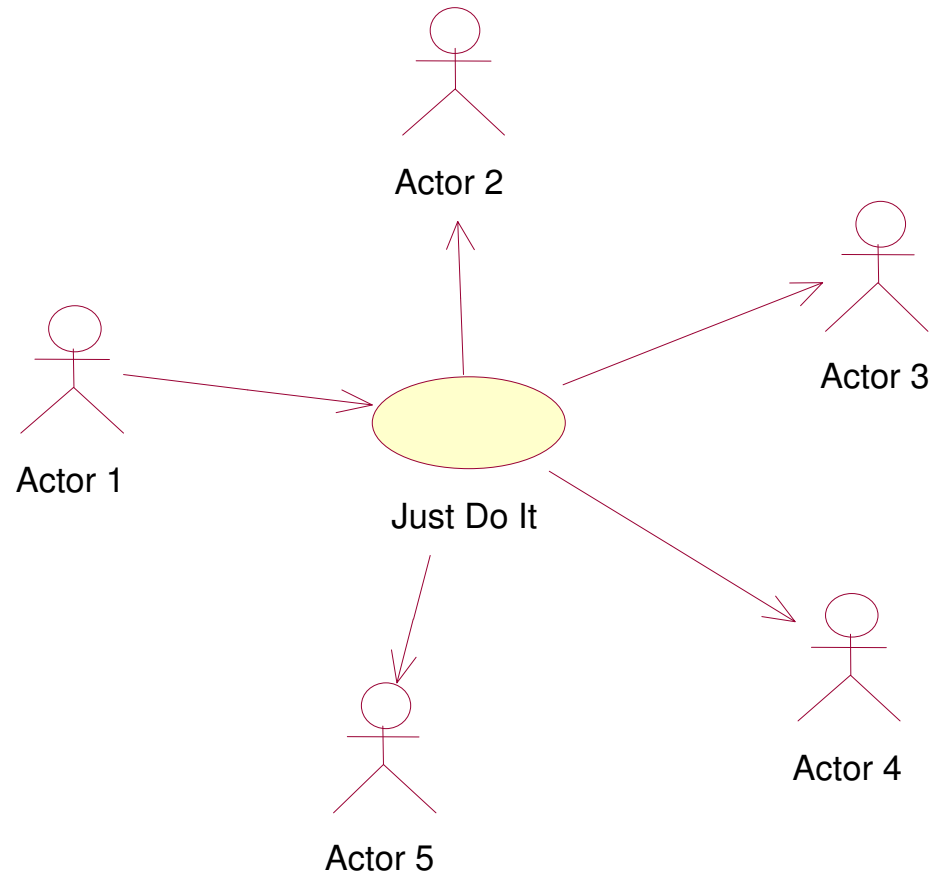
- Are either too abstract or too complicated
- Alienate the stakeholders
- Are only used by the people who create them
- Try to design the system
- Describe how the system will do things



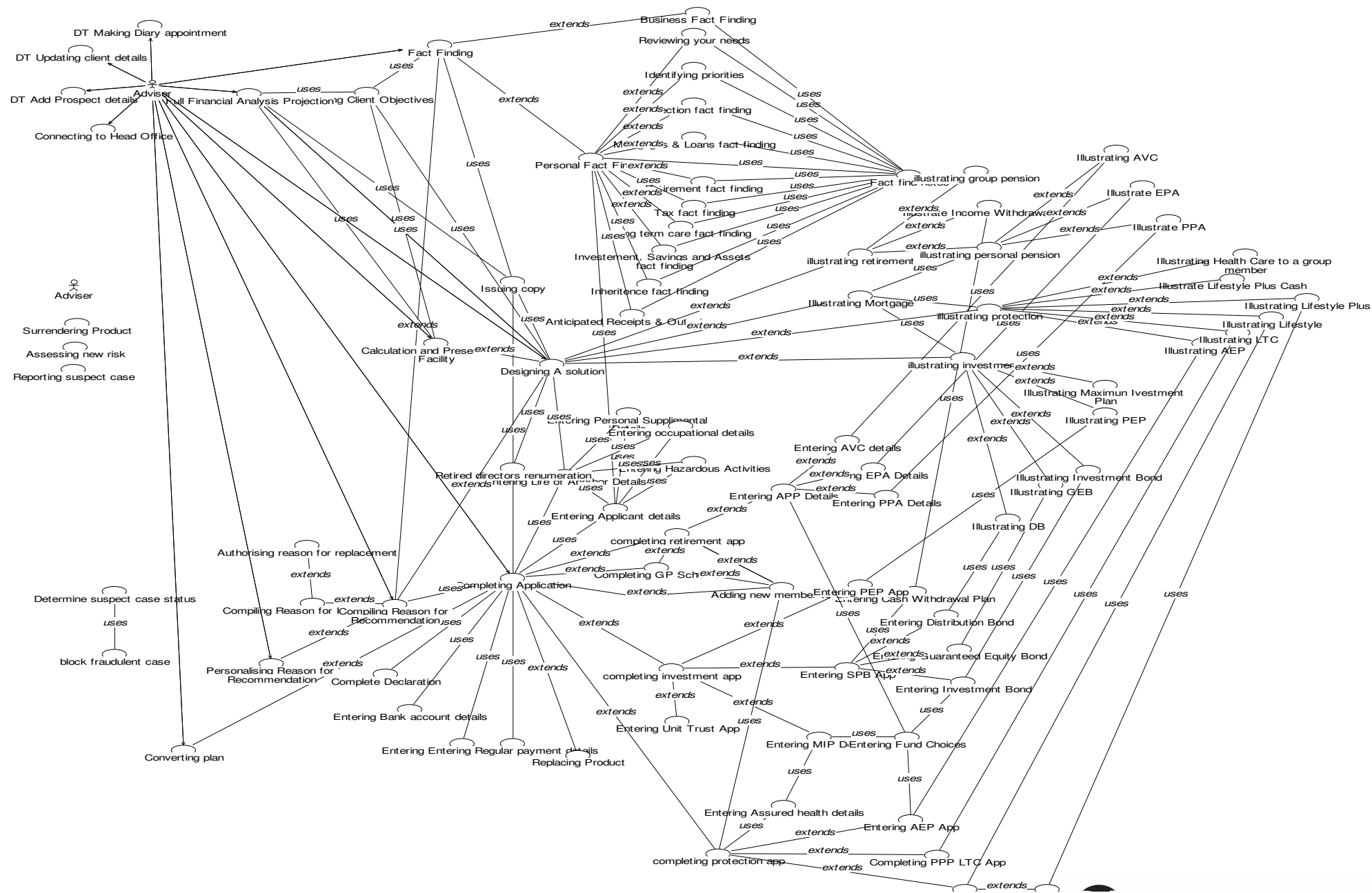
A useful use-case model



Useless Model #1



Useless Model #2

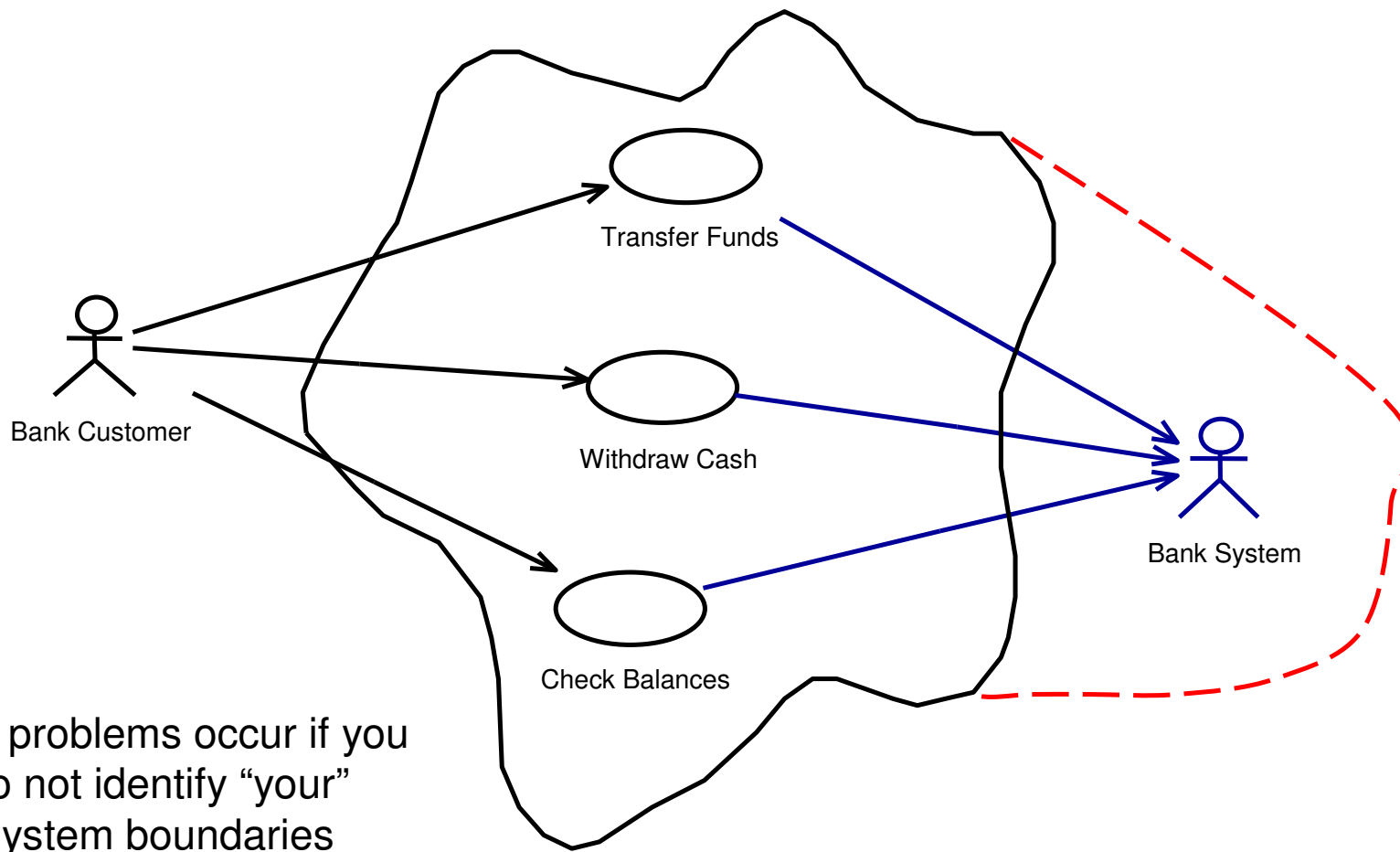


Agenda

- Useless Cases and Useless Models
- Avoiding Useless Models
- Avoiding Useless Cases
- Questions and Answers

Focus on the System Boundary

- Considering other systems as actors forces you to confront the boundary of the system you are creating

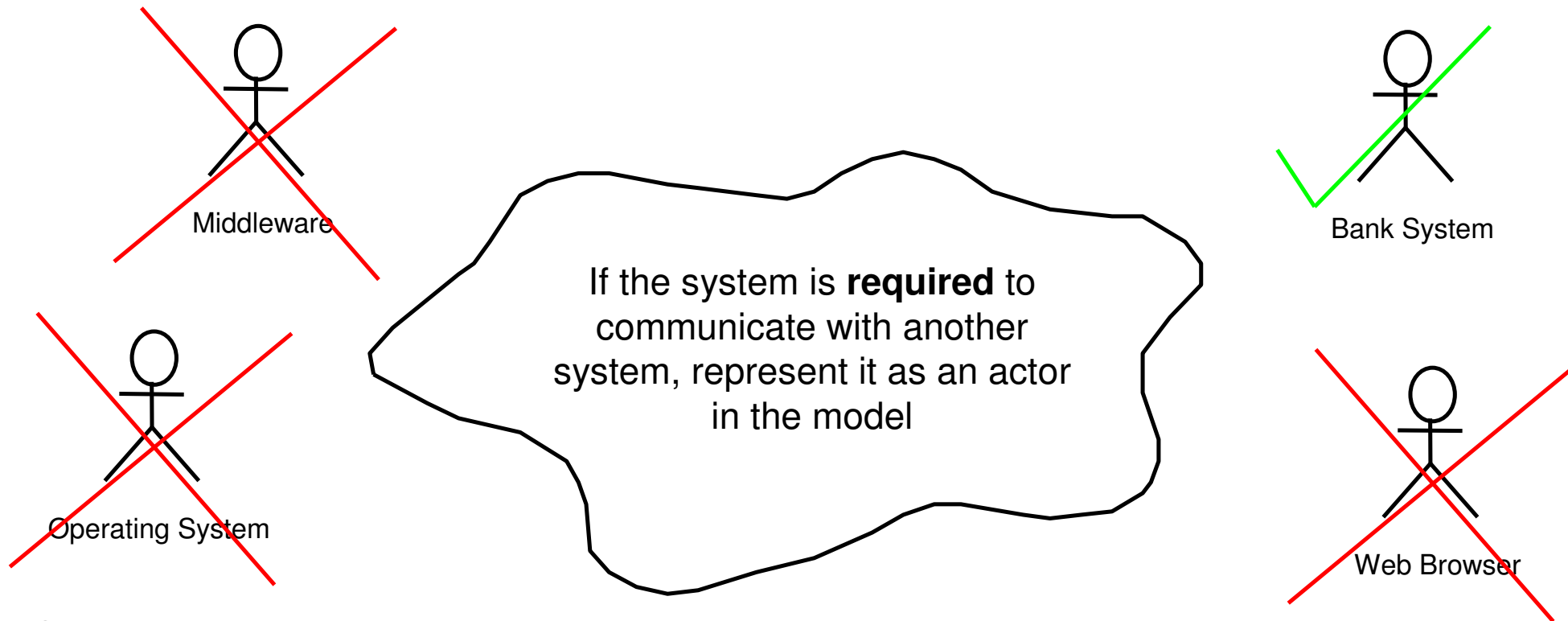


Where is the information that will be required to support the behavior?

'Big' problems occur if you do not identify "your" system boundaries

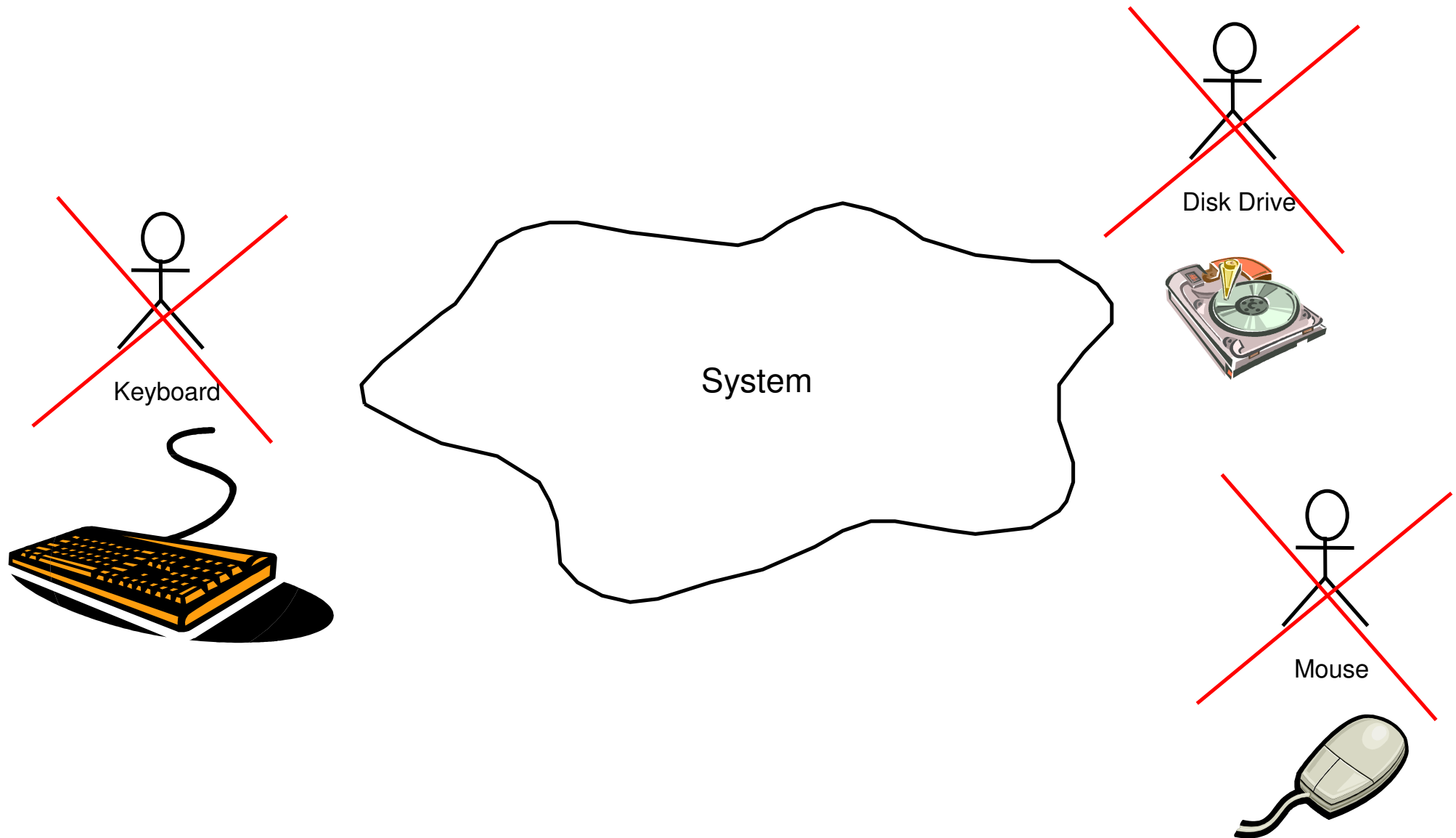
Don't pre-empt the Design

- Is the other system really an actor or part of the system's assembly?



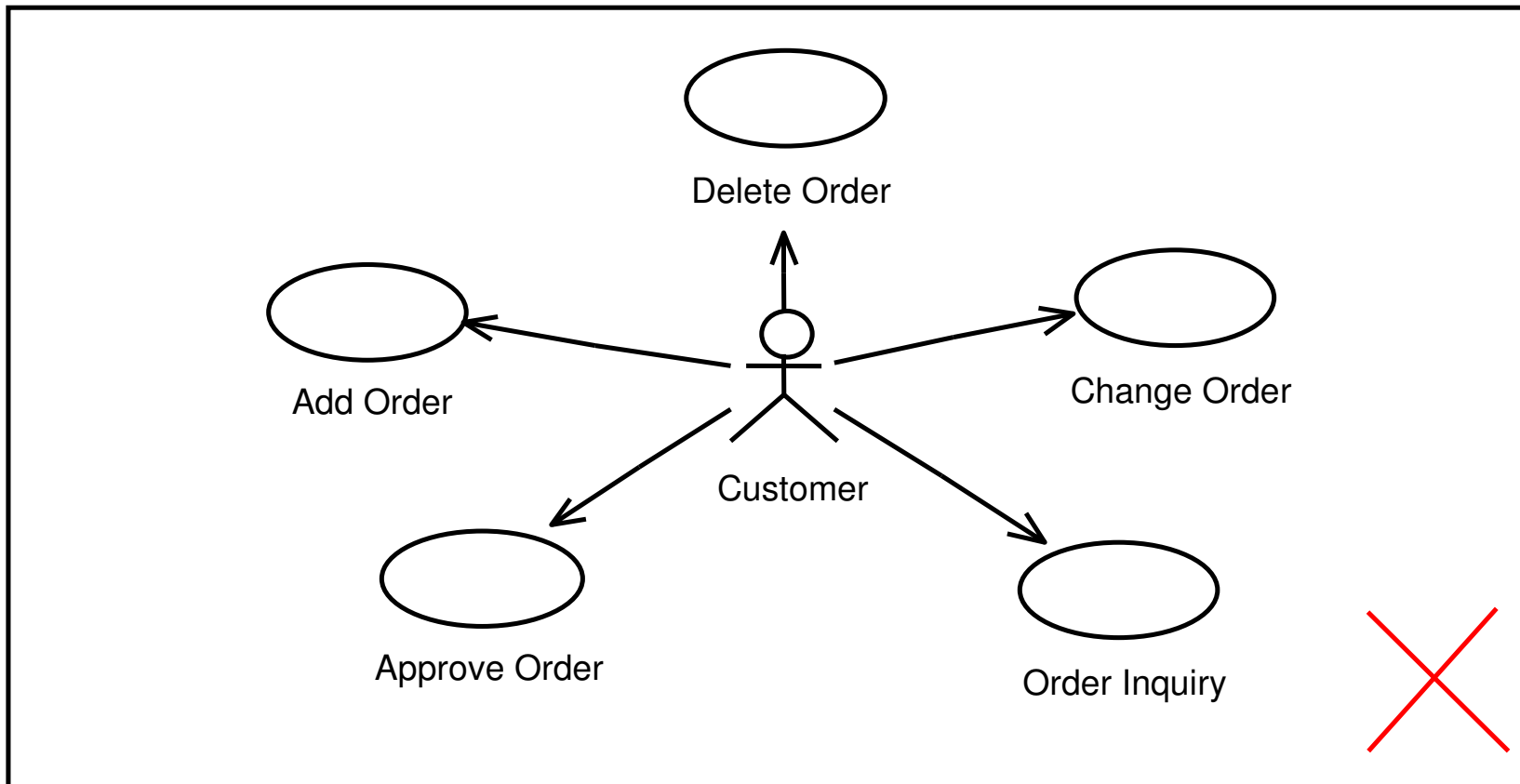
“Use-Case Modelers must **never** pre-empt designers and try to use the use-case model to design the system”

Don't Confuse the Actors with the Devices They Use



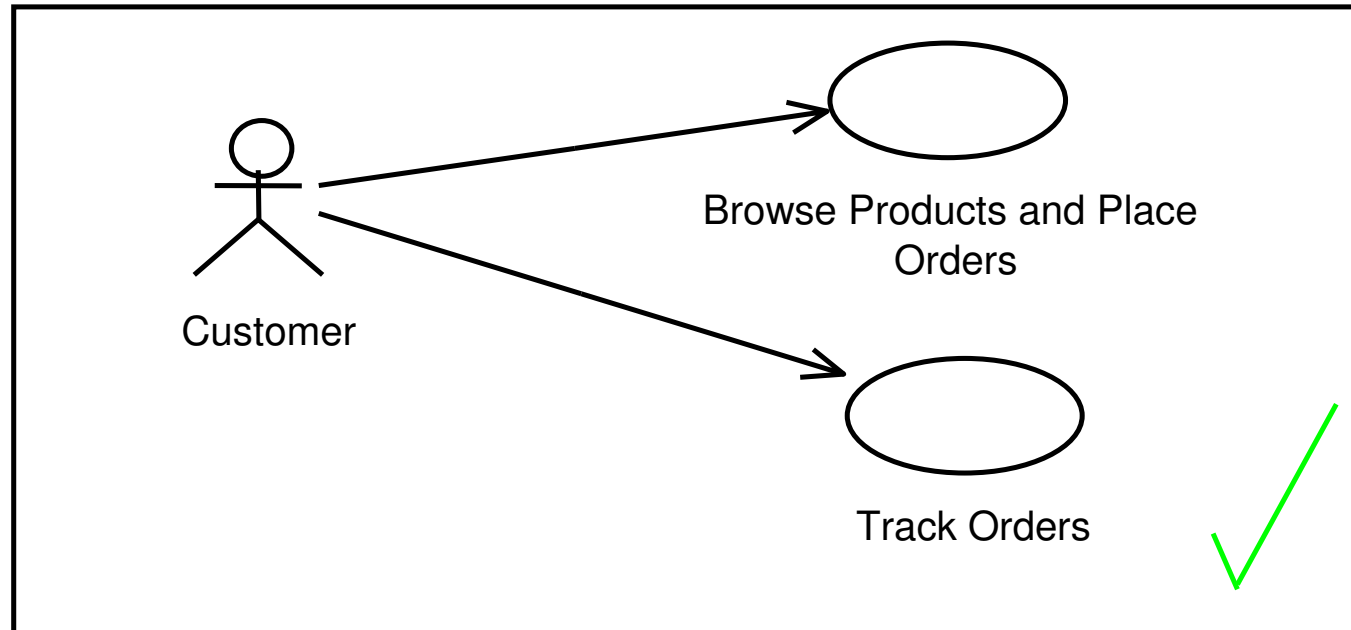
Don't Confuse Use Cases with "Functions"

Incorrect Use of use cases as menu options or functions:



Don't Confuse Use Cases with "Functions"

Use cases that combine functions to reflect the real value to the actor:

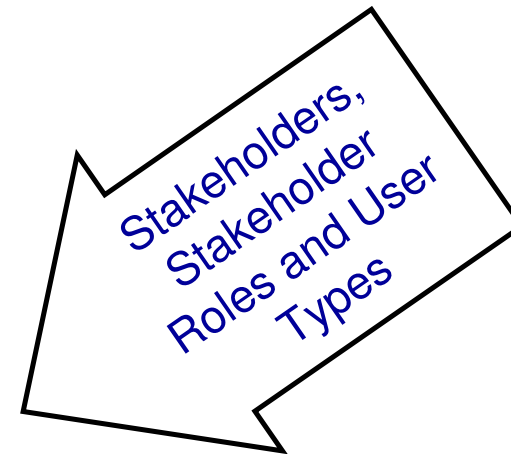
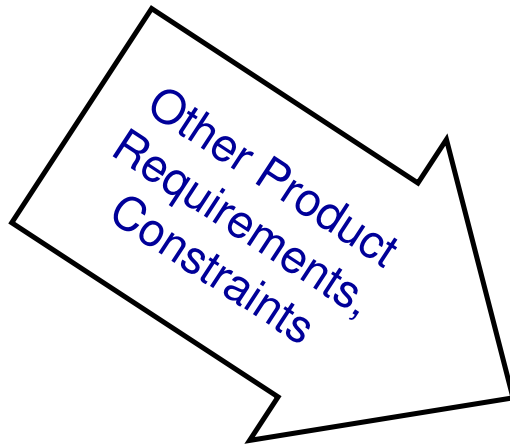


- Take care to focus on value to the user
 - The communicative value of the model will soon be lost if functional decomposition creeps it.
 - Don't end up drowning in hundreds of use cases

Derive the Use Cases from the System's Vision

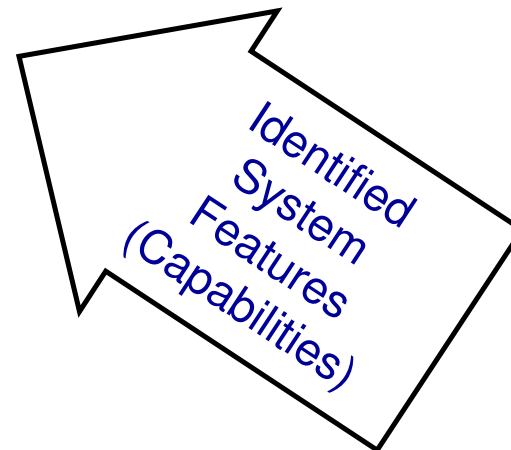
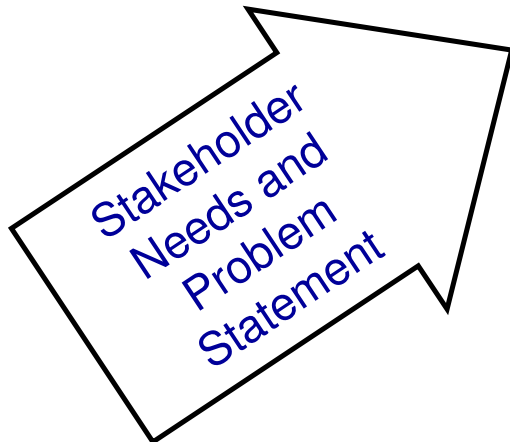
As with 'actors', All existing requirements information should be leveraged to help identify candidate Use Cases

Do the use cases conform to any constraints placed upon the system?



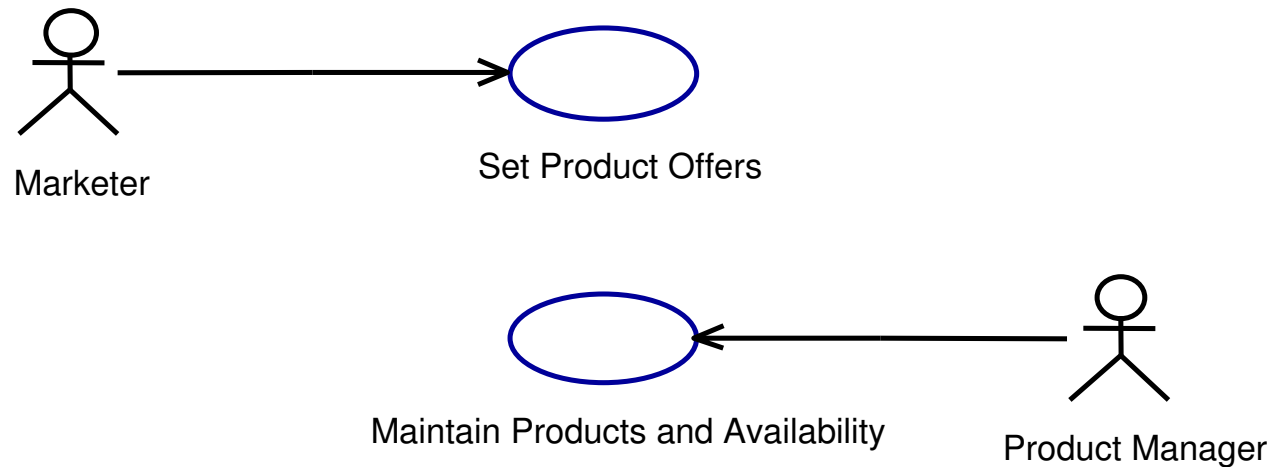
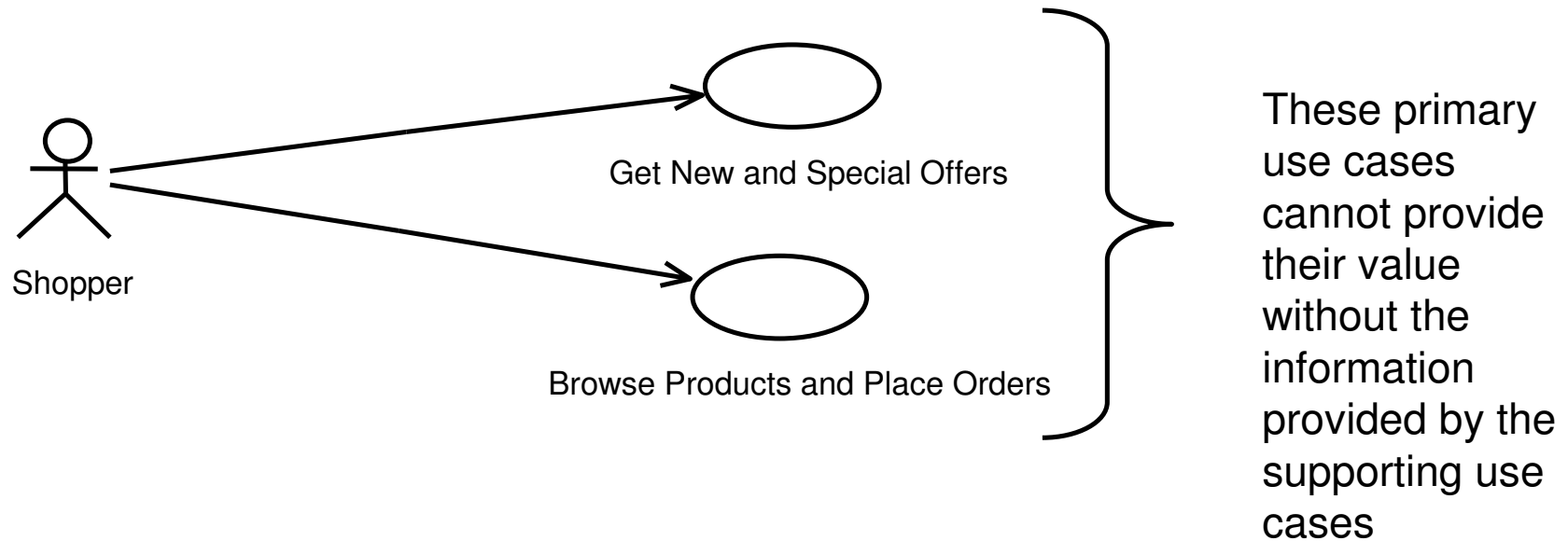
Are all the users responsibilities sufficiently addressed by the system?

Does the system provide the behavior to satisfy the stakeholder needs and solve the problem?



Are the set of identified use cases capable of delivering all the features defined?

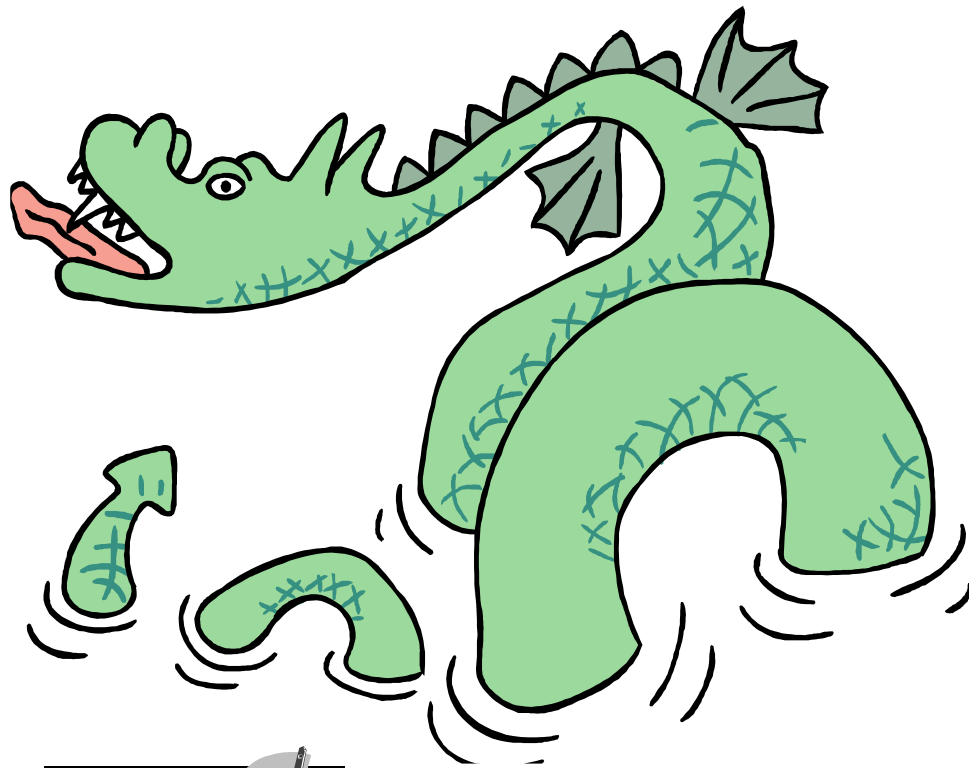
Don't forget the Supporting and Operational Use Cases



Handling Common Problems

- Evolve the set of actors alongside the set of use cases
 - The two activities go hand-in-hand, simultaneously and iteratively
- Avoid functional decomposition
- Maintain focus
- Synthesize don't analyze
- Don't describe outside the system
- Don't just draw pictures
- Don't mix business and system use cases
- Remember good use-case models have no levels
 - Ensure that each use case provides something of value

Don't Over Structure the Model.



"Here there be Dragons"

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

----->
<<include>>

<-----
<<extend>>

A Good Model Helps to Create Good Use Cases

- **The key to a successful start with use cases:**
 - Understand the purpose and boundary of the system
 - When Identifying Actors work from the specific to the general
 - Don't forget external systems that interact with the system being developed
 - A use case should provide independent value to the actor, if you have to execute several use cases in a sequence to add 'value', you have gone wrong
 - Use the vision and the high level requirements it contains to validate your use case model.
 - Evolve the set of actors and use cases alongside each other in an iterative and incremental fashion.

Agenda

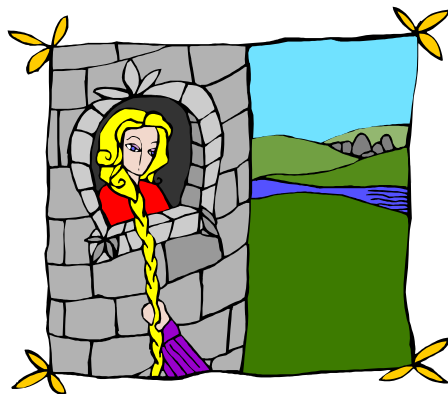
- Useless Cases and Useless Models
- Avoiding Useless Models
- Avoiding Useless Cases
- Questions and Answers

Write Simply, Directly and Deliberately

- Write in *active voice*:
 - Direct declarative statements, “*the system validates the amount entered*” rather than “*the amount entered should be validated by the system*”
- Write in *present tense*:
 - What the system does, not what it will do, “*the system validates the amount entered*” rather than “*the system will validate the amount entered*”, when will it do it?
- Write in *newspaper style*:
 - Simple sentences in a top down format. From major to minor issues.

Treat the Use-Case Like a Story

Once Upon a Time



They all lived happily ever after....

The End



●
“The use case starts when the actor [Actor] does [something]”



“The [Actor] does [xxx]; then the system does [yyy] in response”



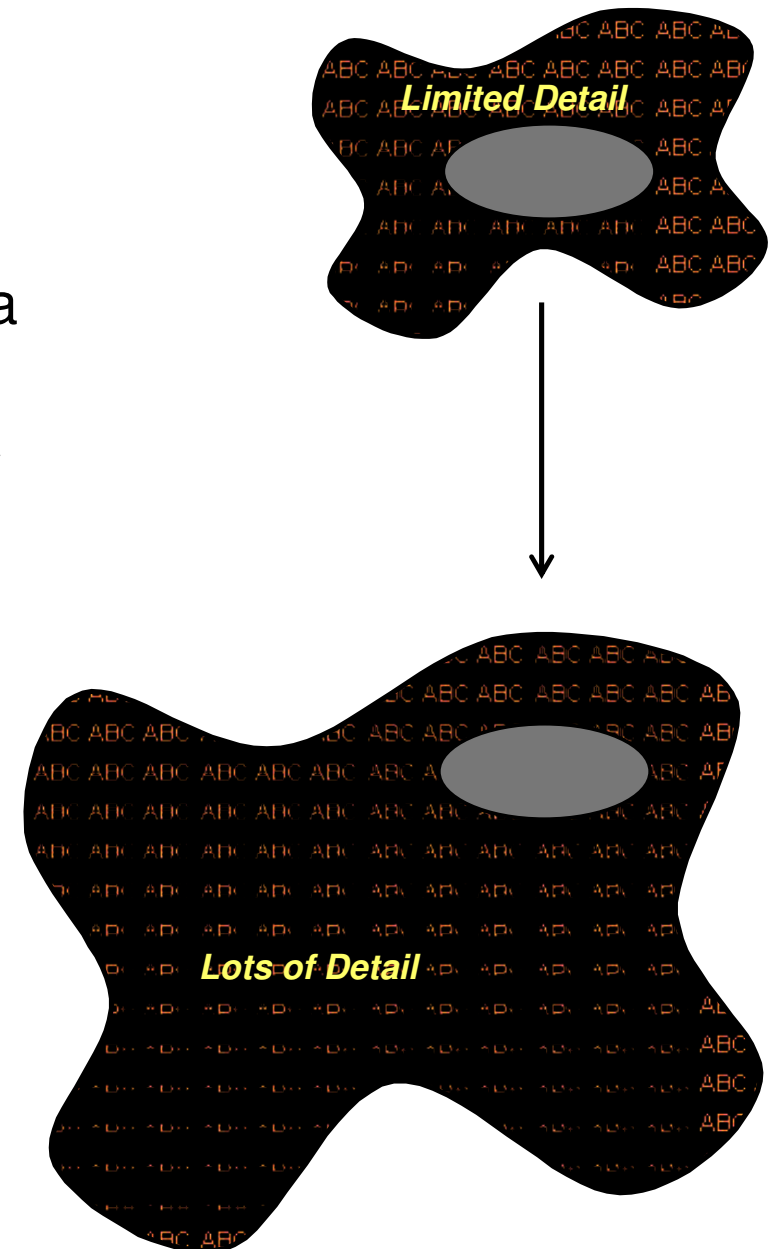
“The system does [xxx]; then the [Actor] does [yyy] in response”



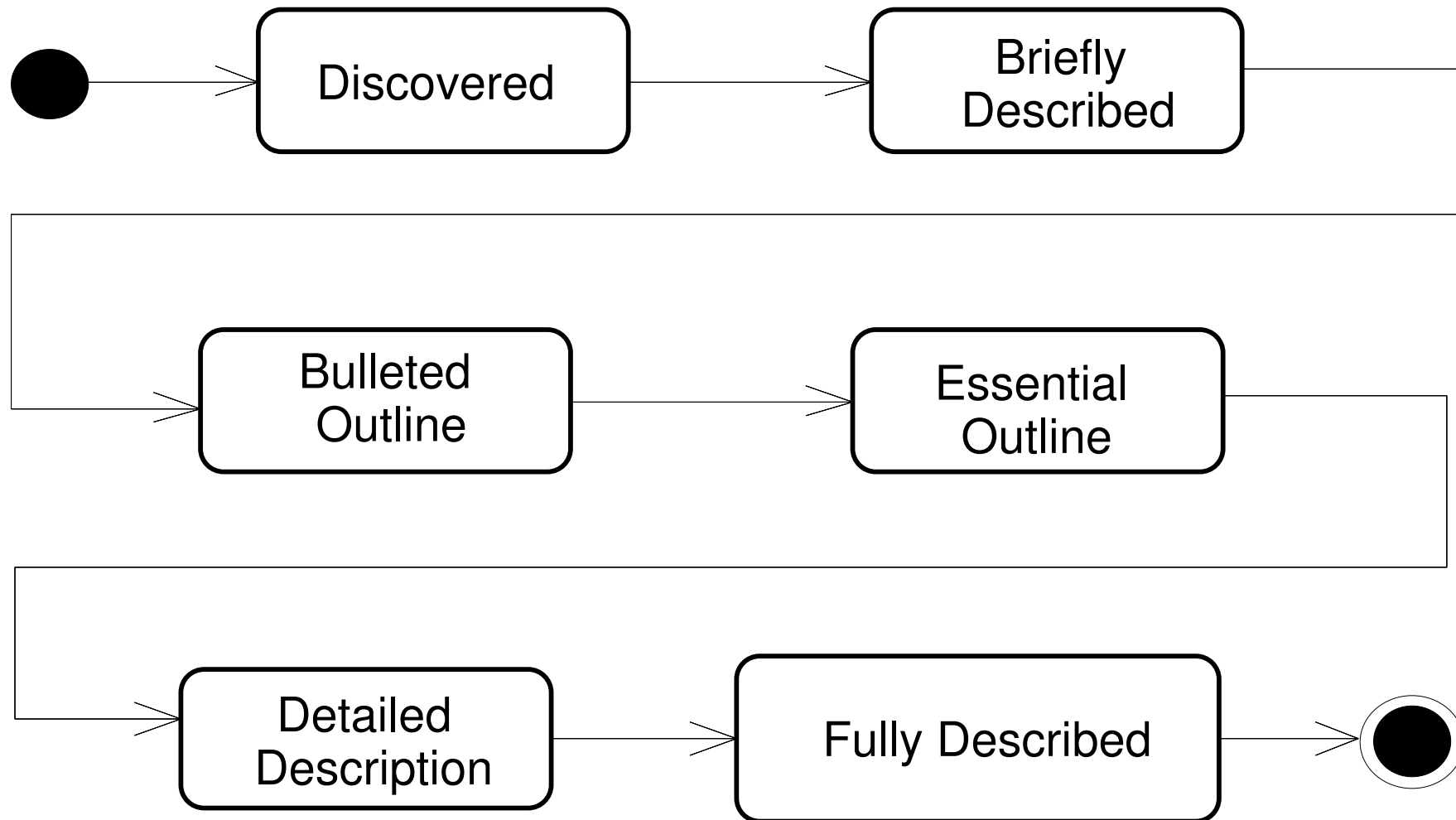
“The use case [xxx] ends when [yyy]....”

Make a Conscious Decision about the Depth of Detail

- Use Cases can be considered a technique for reducing risk that people might misunderstand what the system should do
- The amount of detail you need is dependant on a number of factors including:
 - The level of legal / contractual / safety / regulatory requirements
 - The amount of domain expertise you have in the team
 - The need to record complex decision making
 - The detail required to support testing activities



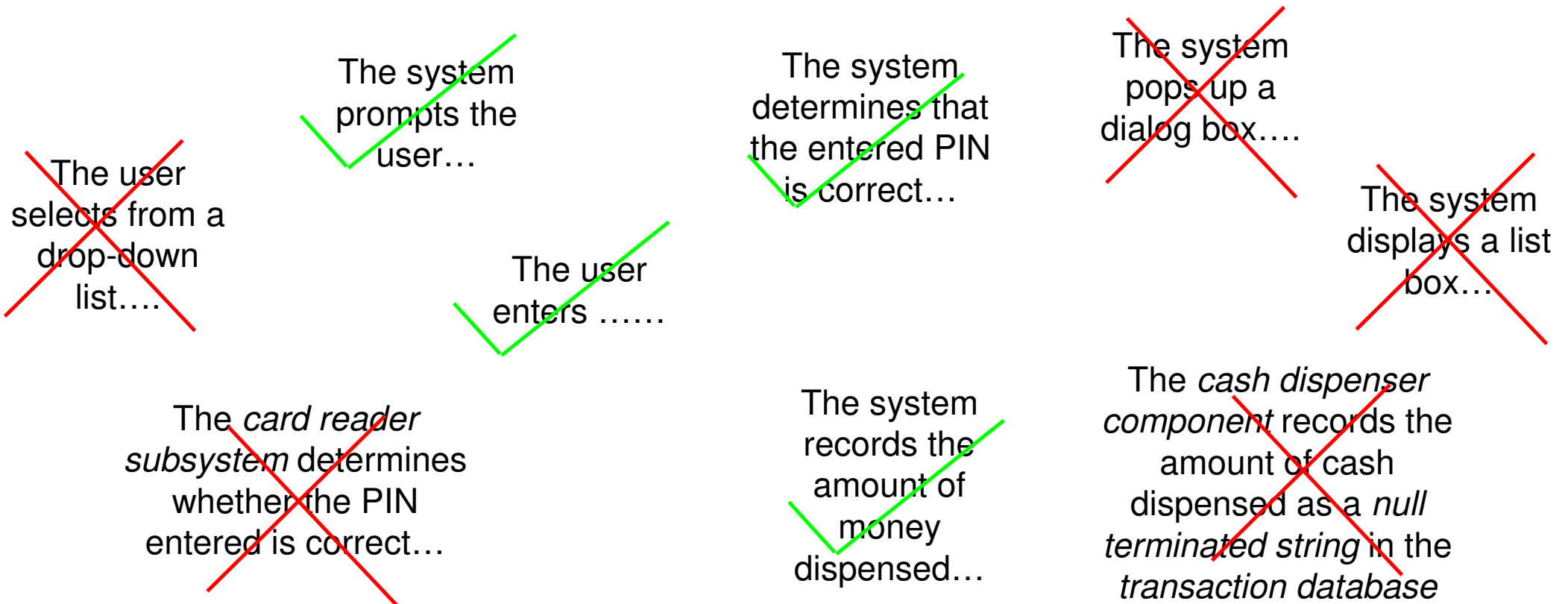
Sometimes an Outline is Enough



Source: Use Case Modeling, Kurt Bittner & Ian Spence, Addison Wesley 2002

Describe what Happens when the Actors / System Interact

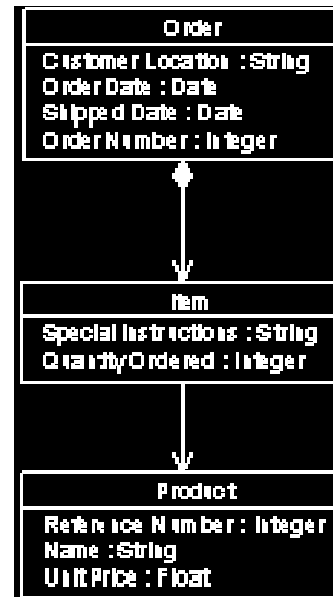
- You will have to describe what the system does
- Do not describe how the user interface or internal components collaborate to do what the system does



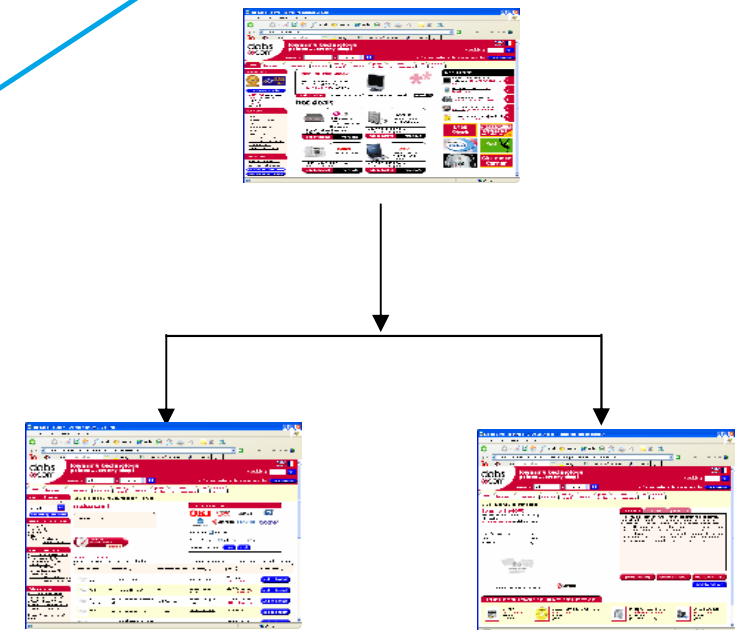
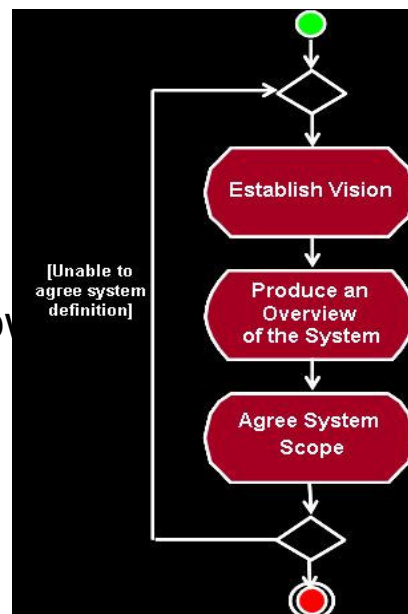
Don't rely on Just Text

- Use the appropriate visual techniques to supplement and support your use cases

Class (Domain) models, help illuminate the use case by explaining complex concepts



Activity Diagrams, provide a way to visualize a complex flow of events



Use-Case storyboards, a series of screen shots from a prototype to depict the flow of events of the use case

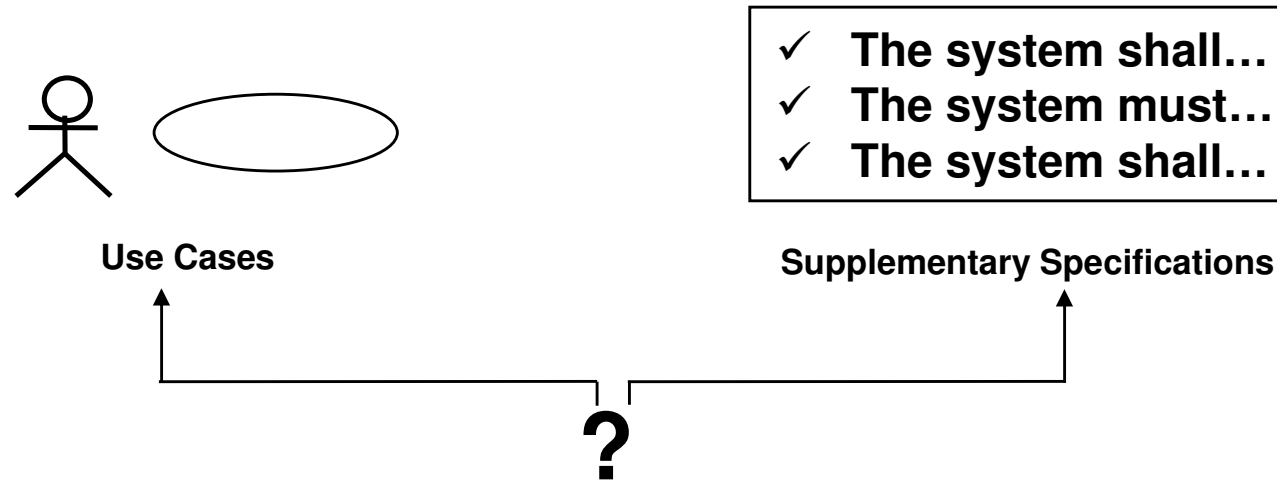
Don't Ask People to Describe Things They Don't Understand

- Don't force people to make up requirements
- Use other techniques to support your use cases
 - Observation
 - Business Models
- A picture is worth a thousand words
 - A prototype is an invaluable tool to describe a user interface, leaving the use case to describe what happens behind the scenes.
 - Great for generating user feedback
 - Great for presenting to stakeholders
- The prototype should evolve in parallel with the use case descriptions
 - Great for instantiating scenarios
 - Can be a good basis for usability testing

Use the Glossary and Domain Model to Capture Definitions

- Anytime you see a lengthy discussion or definition that serves mostly to explain background information, consider putting it into the glossary
- Enlightened use of a glossary will simplify the use-case descriptions by allowing the use case to focus on describing behavior not terminology.
- **Details Matter**
 - What is “customer information”, better define it as “Name, Address, Order History” and so on..

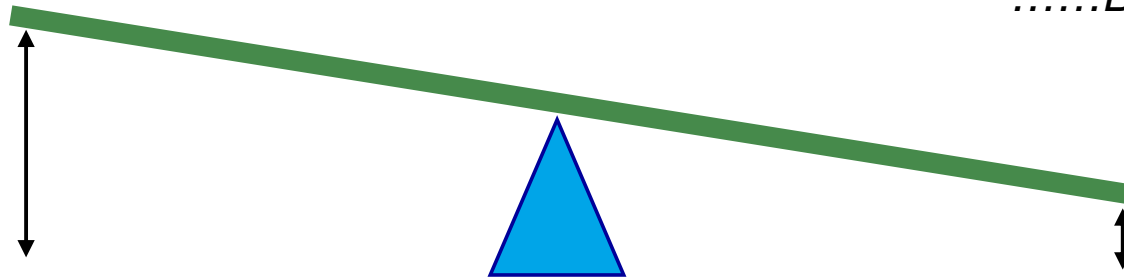
Don't Forget There Are Other Places to Put Things



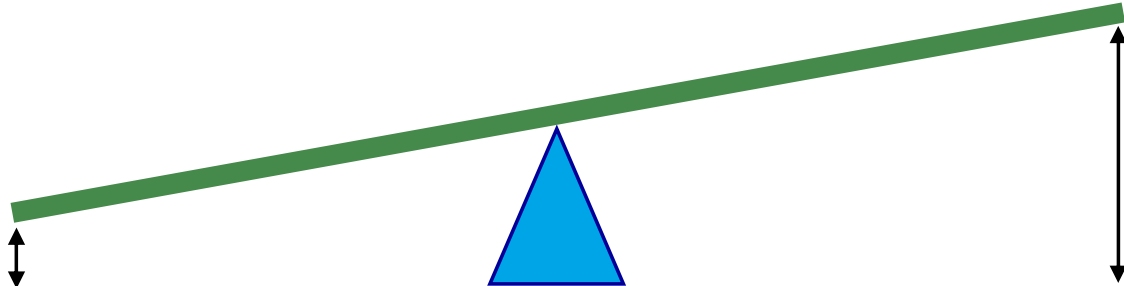
Which one to choose?

.....Balance is the key!

A large amount of actor interaction (ATM / UI) = majority of requirements in use cases

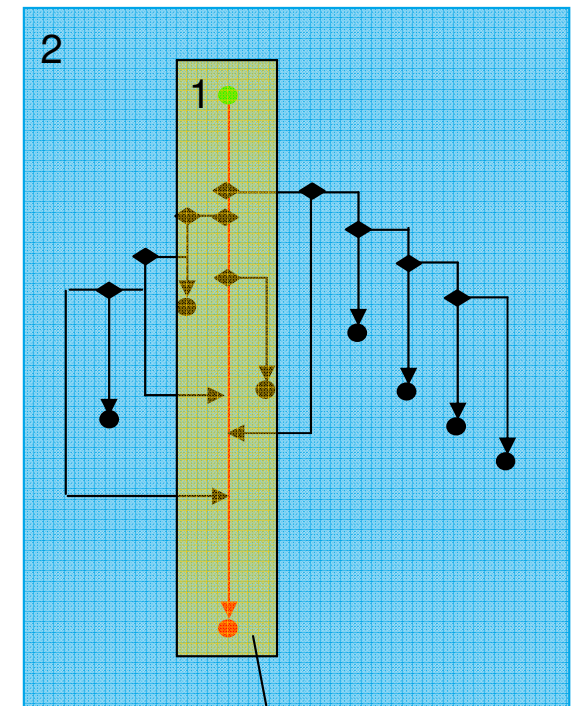


A small amount of actor interaction (*a compiler*) = majority of requirements in supplementary specification



Start from an outline – Start from the basic flow

- Good writing typically proceeds from an outline; use cases are no different:
 - Outlines will help clarify the purpose of the use case and help you think through each step
- As you evolve the use case from the outline, focus first on the basic flow
 - Evolve the alternative flows only when there is the framework of the basic flow upon which to build



Basic flow,
then
alternates

Don't Write Them On Your Own

- Involve other people in the creation of the use cases
 - Subject matter experts
 - Developers
 - Testers
- Always share and walkthrough the outlines before adding more detail

Write down the requirements

- Don't be scared of detail
- A use case with no detail truly is a “useless case”
- The trick is to include it without obscuring the use-case message.....

Agenda

- Useless Cases and Useless Models
- Avoiding Useless Models
- Avoiding Useless Cases
- Questions and Answers

Thank You

ispence@ivarjacobson.com