OP++ Requirements

Module: Requirements Workflow

Author: Kim Horn
Date: 20 July 2014
Version: 5.92

Agenda

Overview
Artefacts: Vision and Requirements Document *
Usage and Function
Use Cases and Actors
Visualising Use Cases
Slices and Stories
Journeys and User Scenarios
Informational Requirements
Non Functional Requirements and QOS
Constraints Glossary
Process and Business Use Cases
Service/SOA Use Cases

* The requirements process is covered in another module

The Problem V Solution Space

The Business Problem that initiates the change/project.

Requirements: Business Process, Rules, Business Architecture, Use Cases, Epics, Stories, System Qualities (Scalability Usability, etc) …….

A Solution to the Problem.

Architecture, Technical Requirements, Design, Code, SQL, Java, Hardware….

Functional Decomposition Only happens here.

Implementation of the Operational Requirements and Evolutionary Non-Functional Requirements into the future.

Requirements

IEEE standard (1990) defines the term requirement as:

- (A) A condition or capability needed by a user to solve a problem or achieve an objective.
- (B) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
- (C) A documented representation of a condition or capability as in definition (A) or (B).

Underpinning this definition is the notion that requirements describe a desired state of the world after a system has been constructed to interact with the world. The relevant part of the world is called a system's environment.

However, the environment can affect the behaviour of a system and the behaviour of a system can affect its environment. This compounds the difficulty in separating the Problem and Solution space.

Requirement or Solution

A common mistake during software development is to focus on solution formation before the problem is well understood and analysed, before the business problem is understood.

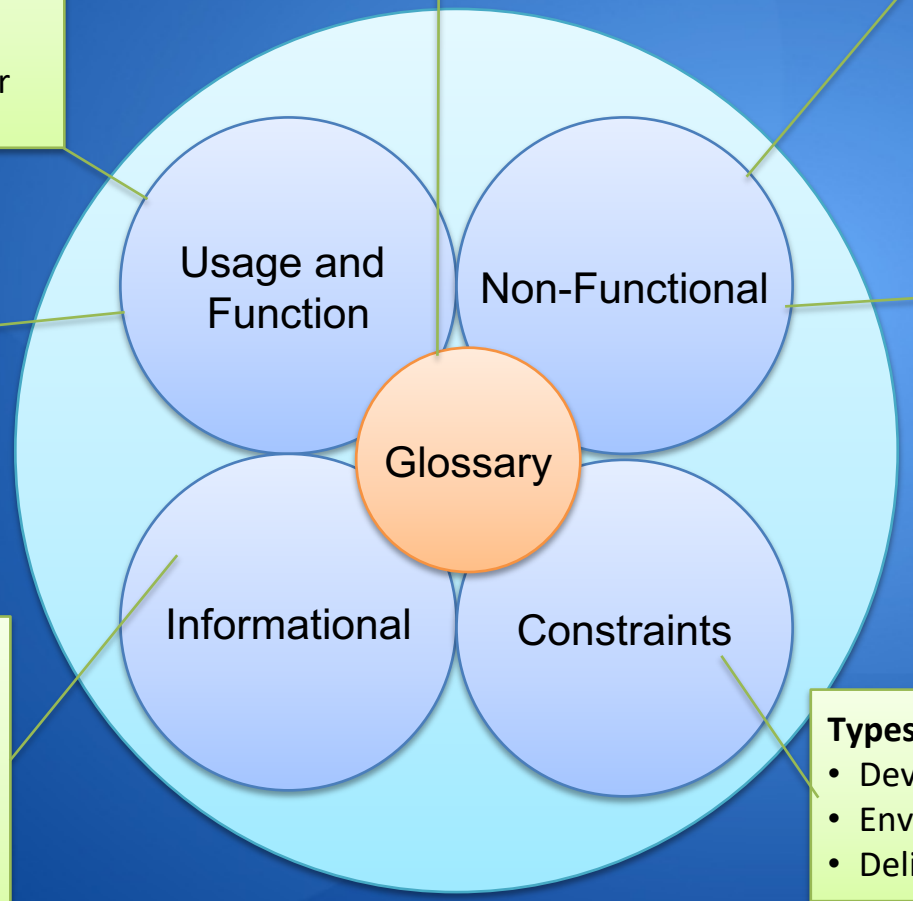
For example, in the domain of mobile phones a user may identify a requirement for "a sliding cover for the mobile phone".

Arguably, this requirement is a solution to the underlying problem which may be that they want to protect the phone against pressing keys accidentally or that they want to improve the aesthetic appeal of the phone.

It is critical to describe the problem space before the solution space. There is no one approach to requirements or model that covers all requirements. However, just using one, or using the wrong requirements model compounds this problem.

The techniques to be described in the requirements workflow cover a range of problem types using multiple models for different types of requirements.

Types Of Requirements



Includes:
- Actors, Roles and Personas
- Use Cases, Scenarios and Journeys
- Epics, Functions, User Stories.

Define the language and terminology used in the project domain

Includes:
- Performance
- Availability
- Scalability
- Security
- etc

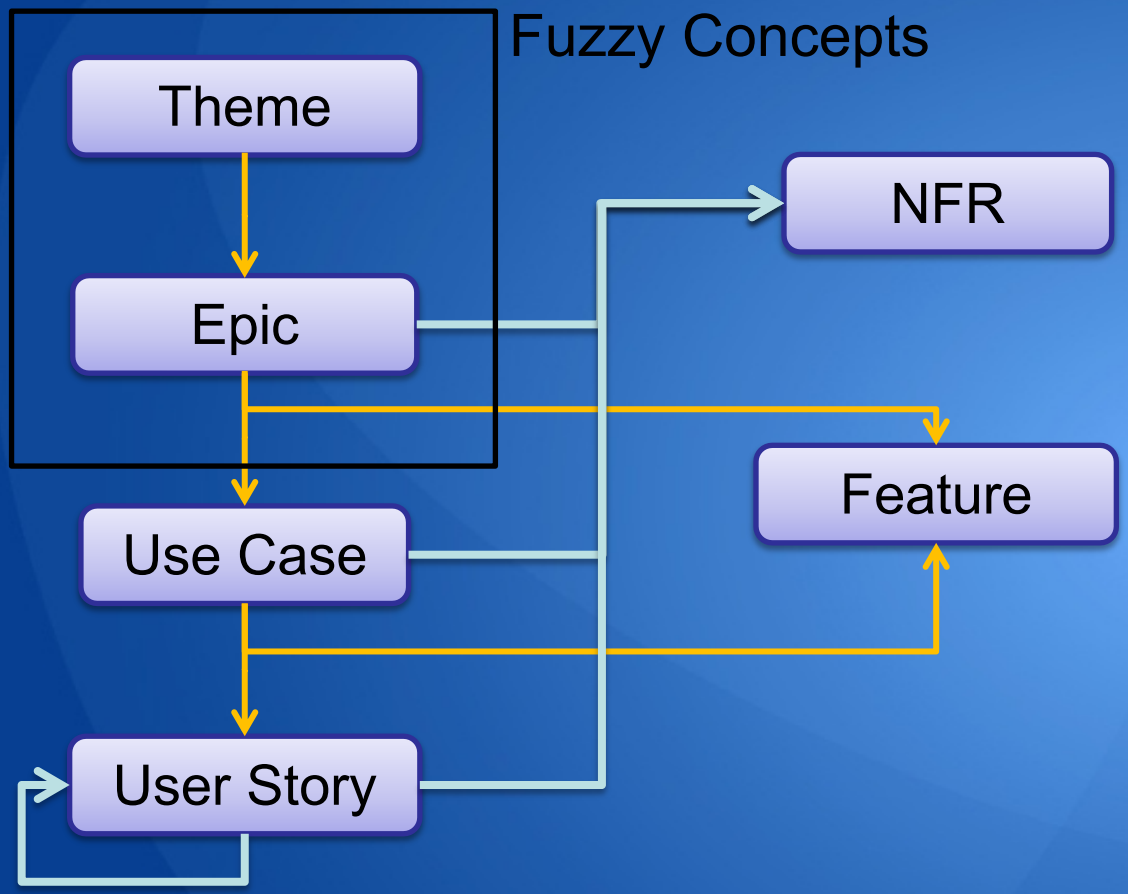
Types:
- Business
- Operational
- Infrastructure
- Integration

Types:
- Manifest
- Operational
- Developmental
- Evolutionary

Includes:
- Events
- Wire Frames
- Entity/Domain models
- Business Rules

Types:
- Development Process and Team
- Environment and Technology
- Delivery and Development

Types Of Requirements – High Level

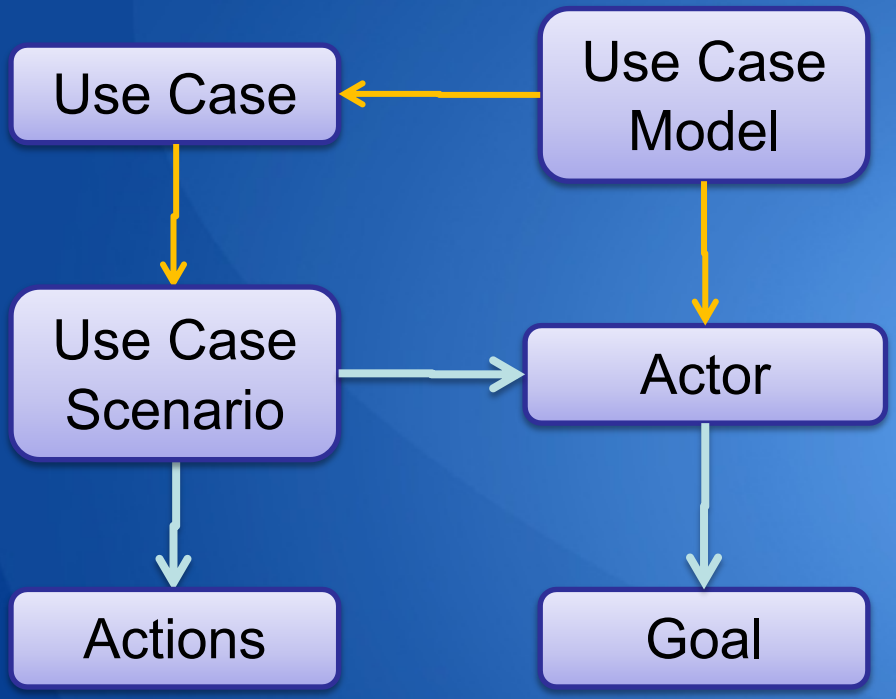


Epics and Themes are fuzzy concepts that are commonly used as placeholders for user stories; they have to be sliced before they can be implemented.

Use Cases help organise and aggregate User Stories and Features and other requirements, e.g. journeys, scenarios, NFRs.

Epics, Stories and Use Cases have more than Non-Functional Aspects – Traditionally NFRs have been viewed as distinct to Functional; but such a distinction is unrealistic and overly simplistic.

Types Of Requirements – Next Level



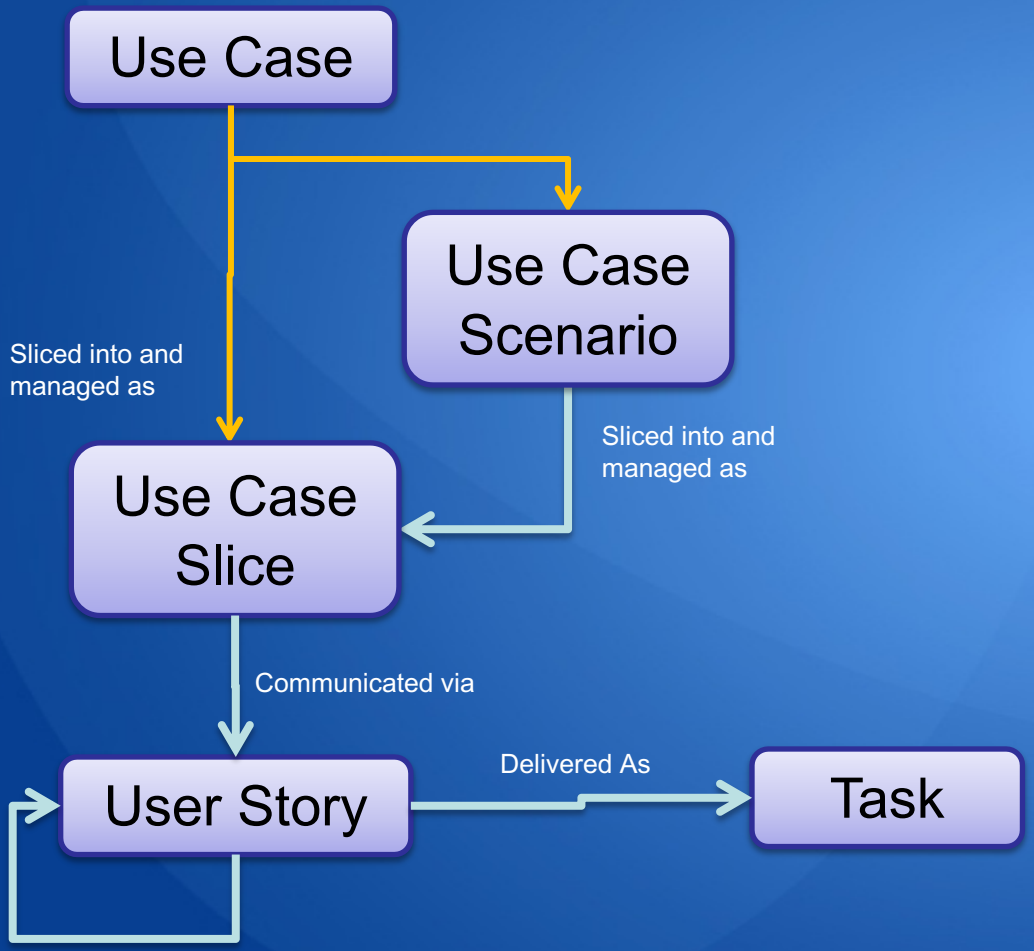
A Use Case Model is a graphical way to depict the context of a system by depicting the relationship between a number of Use Cases and Actors

A use case describes what the system must do to provide value to the stakeholders;

They describe the interactions between one or more Actors and the system in order to provide an observable result of value(Achieve a Goal) for the initiating Actor.

Use Cases are a collection of Use Case Scenarios. A Scenario describes a particular sequence of Actions by the Actors and responses by a system.

Types Of Requirements – Delivery

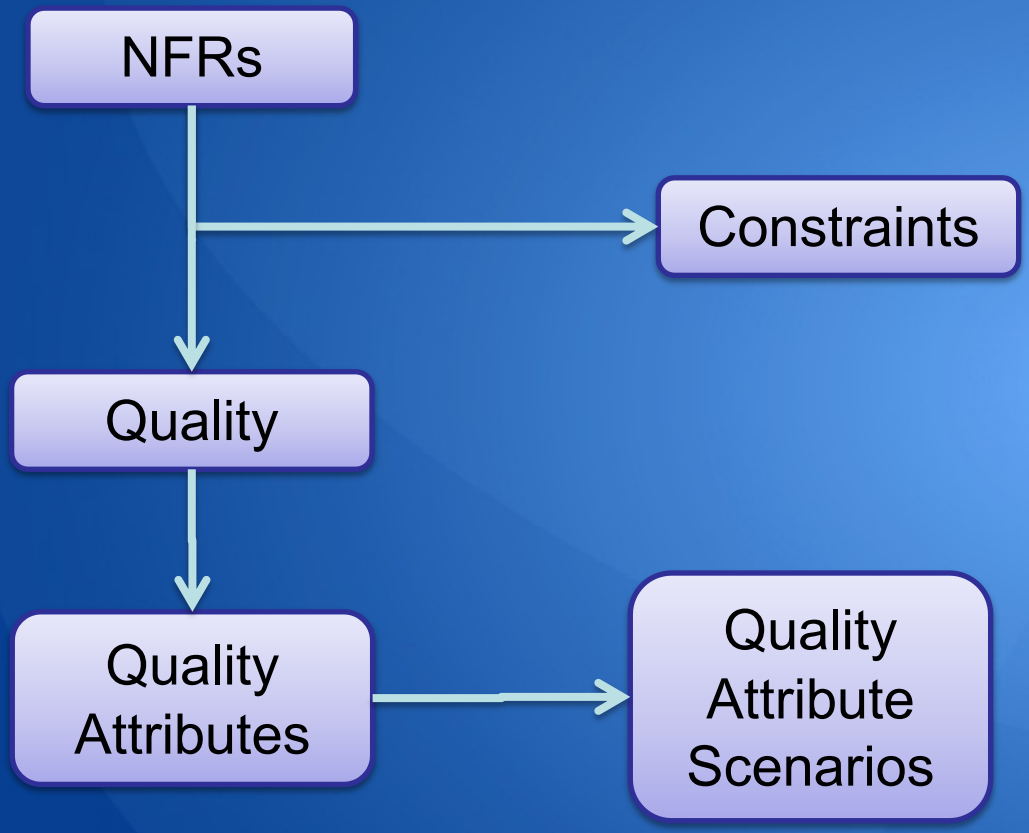


Use Cases and Scenarios can be broken into Slices.

A Slice is communicated as a User Story

A Story is delivered as a development Task

Types Of Requirements – NFRs



NFRs include:

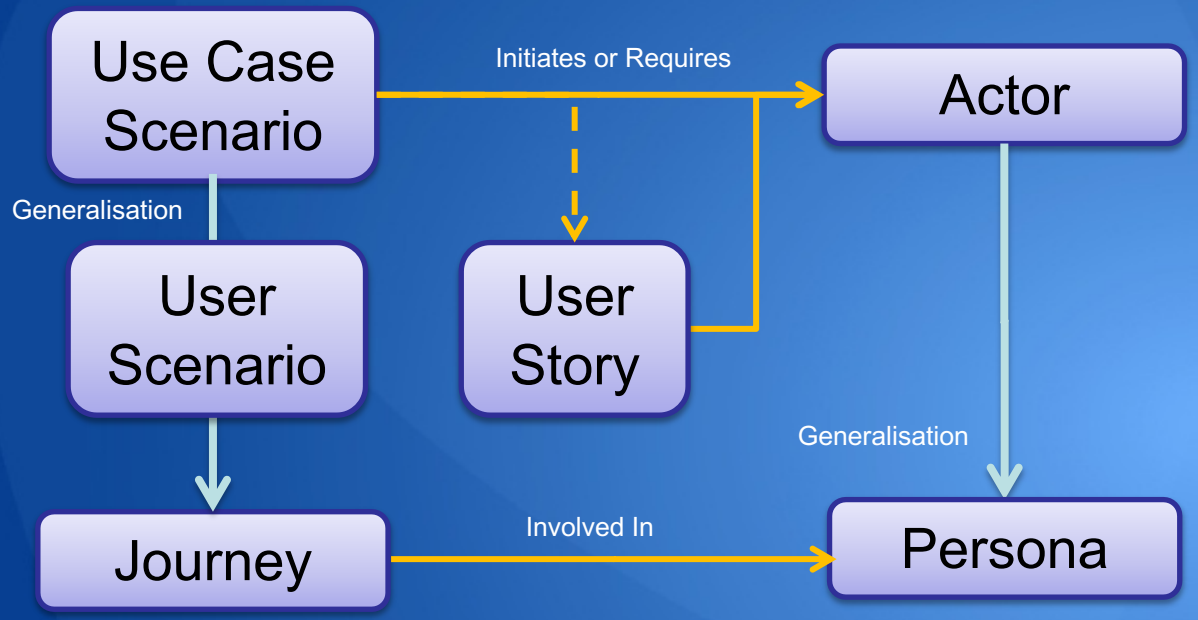
- Constraints
- Quality of Service

NFRs can be pervasive across the system or relate to specific Functional Requirements.

A system or a components quality is described by Quality Attributes.

Quality Attributes are involved in Scenarios that detail how they are achieved.

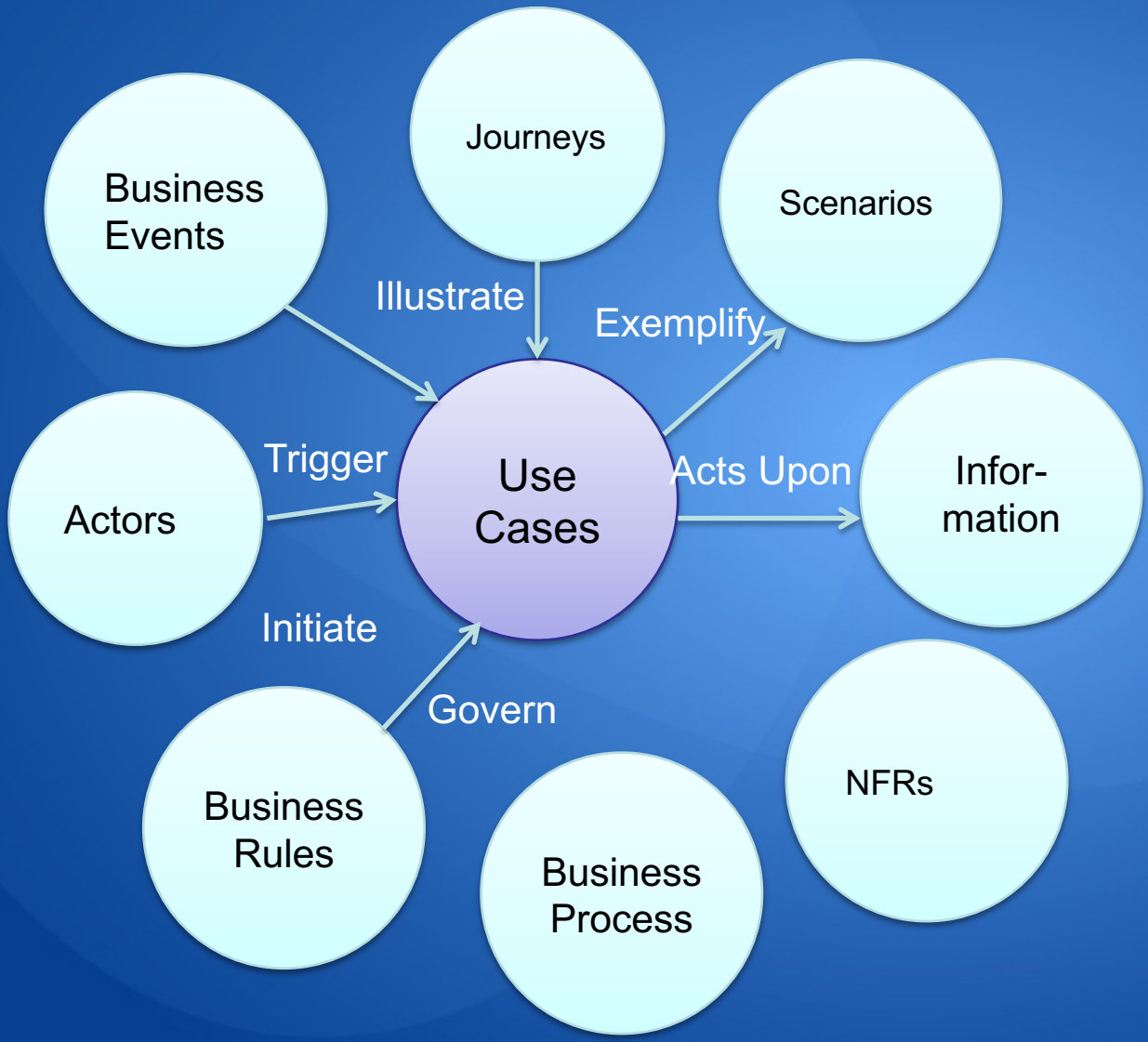
Journeys, Scenarios and Personas



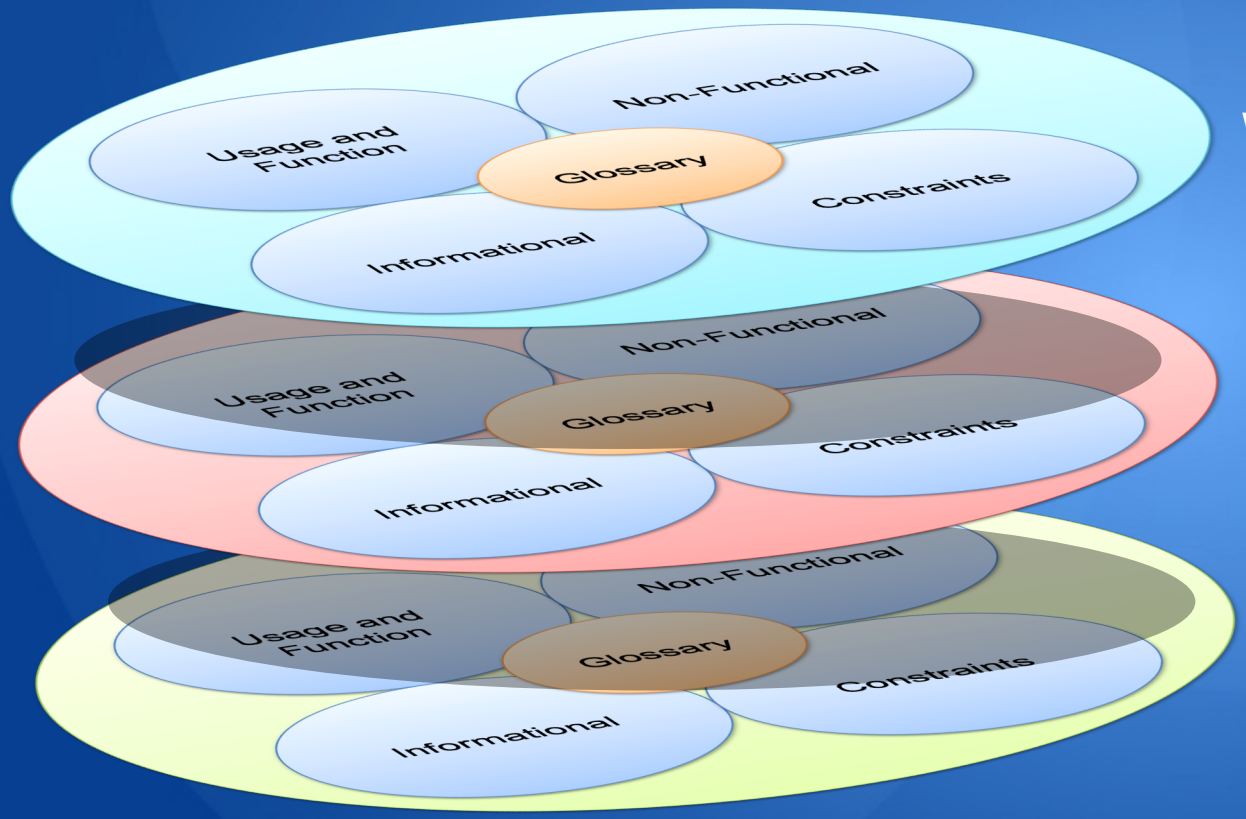
Journeys are a specific and highly detailed instance of a Use Case Scenario that focuses on a specific user's interaction.

- Personas are a specific identified instance of a Human Actor User that have a specific journey. Journeys don't usually cover other systems.
- Both Use Cases and Stories are enacted by Actors.
- 'User Scenarios' focus on a users goal and how they achieve that without describing what the system does. They are a halfway house between Use Case Scenarios and Journeys.

Models that feed off Use Cases



Levels Of Requirements



Business Level

Why the project is being undertaken ?

Solution Level

What users can do with the solution ?

System Level

What needs to be built ?

Business Level

- What is the Vision for the Project ?
- What are the Strategic Objectives (goals) this Initiative provides ?
- What capabilities does the business need to automate ?
- What is the Business Case ?
- What is the scope of the project ?
- What are the processes the business undertakes surrounding the system. These processes provide tasks for the users that require the system. Understanding this context helps bound and provide system or product requirements.
- If the project is delivering a product what is the market context ?
- What is the model of the business, and how will it operate with the system and what changes is required for the business ?
- What are the main features the business requires of the system or product ?
- What are the happenings of interest (events) that the business responds to ?

Solution or User Level

- What are the goals of the users of the system ?
- How will the system solve the business problem for the users ?
- What are the non-human or other system users and what do they require to operate ?
- What are the goals of other stakeholders (non-users) of the system, for example if it's a product then immediate users may not have a need yet ?
- What are the tasks these users need to accomplish ?
- What is the information, data, they may want ?
- What are the business rules required to automate tasks ?
- What do the operational users of the system need to operate and manage the system or product ?
- How does the system solve the business problem ?
- What are the qualities and levels of service required by users ?

System Level

- What are the technical needs of the system to meet the goals of the users ?
- What information is required of the system or product to allow it to be built and developed ?
- What are the external systems that the product or system needs to integrate with to complete users tasks, and how will they communicate ?
- What is the information (data) required from or sent to other systems ?
- What information is required by the system itself ?
- What are the technical and developmental constraints on the project ?
- What are the non-functional requirements that define the architecture ?
- What is required to operate the system ?
- What are the future requirements of the system and how will it evolve ?

Emergent Requirements

- Requirements ‘emerge’ during the software development lifecycle. The process of building a system will result in knowledge discovery, it is a collaborative and creative process; the requirements will be better understood as the system is delivered.
- Unfortunately humans are poor at predicting, or abstractly specifying, what they really want and the ultimate specification of the requirements is the working system that is tested against the real world by users and stakeholders.
- It is important to add that the working system is definitely not a proxy for good requirements, but the test-bed that proves they are correct.
- Initial requirements are only a starting point, a theory on what is wanted, and not what is actually wanted; not the end result.

OP++ considers the process of building a system as akin to the scientific testing of a theory. Theories incrementally evolve as they are tested against real world cases aimed to refute the theory*. Test Driven Development is a critical practise to achieve quality.

*Karl Popper, *Conjectures and Refutations: The Growth of Scientific Knowledge* (1963)

Emergent Requirements

Your initial problem definitions might take a bit of work.

Much of Agile - and Scrum in particular - is based on a fact of life called emergent requirements. Grandpa Harry used to say that the best laid plans of mice and men often go astray. You can't master plan a project and expect to follow the plan. Agile folks know that. However, most Agile folks think that means only that we discover new problems along the way. It isn't just that we discover new problems: the act of design actually creates new problems. More to the point, the act of design sometimes changes the very nature of the problem we have set out to solve. We must revisit the problem definition to refresh it now and then.

- **Lean Architecture: for Agile Software Development** By James O. Coplien

Artefacts

OP++ Requirements Workflow

- Set of activities that produce the following artifacts:
  - Business Opportunity
  - Context Diagram
  - Stakeholders
  - Use Cases, Stories, Features etc…
  - Informational Requirements
  - Service Level Requirements for Systemic Qualities
  - Constraints
  - Glossary
- These artifacts are captured in the following deliverables:
  - Vision Document – high level detail
  - Requirements Document – elaborates on Vision detail

Who Reads Requirements & Vision Docs

- Business Owner / Product Manager/ Product Owner.
- Funding Authority / Sign-off Authority
- Requirements Owner / System Users
- Project Manager
- Architect / Designer / Developer
- Test Analyst / QA
- Dev Ops
- Operator
- Maintainer

Vision Document

Goal:

Get all Stakeholders to agree on a common Problem Statement to initiate a project.

Includes:

- Business Opportunity
- Success Criteria, how do we know project success
- Proposed Solution and Alternatives
  - Pros, Cons
- Constraints – at a high level only
- Major: Risks, Issues, Assumptions, Dependencies
- Cost, Resourcing and Delivery

- Detail:
  - All items at a high level, detailed when project starts, across Requirements documents

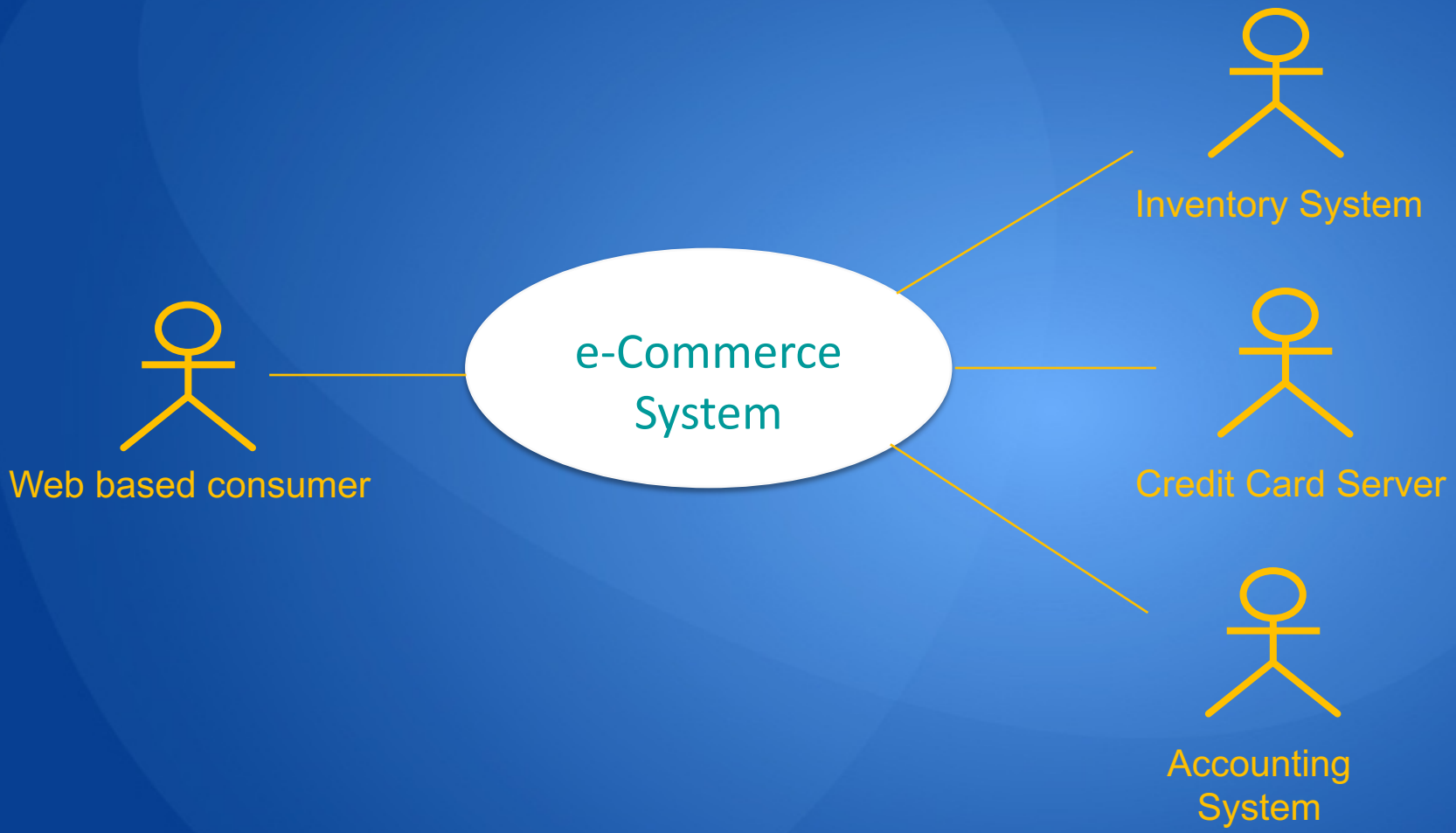
Business Opportunity

- Background
- Positioning
- System Context
- Primary Stakeholders
  - Funding authority
  - Sign-off authority
  - Requirements Owner
- Cost / Delivery Estimate
- Impact of Missed Opportunity

What is a Context Diagram?

- Defines the system context and scope
- Documented using a UML Use Case Model diagram
- Summarises the most important touch points with which the proposed system will interact:
  - Humans
  - External systems
- Protocols employed to support this interaction
- Format:
  - Left: who requires functionality from the system
  - Right: who is interacted with to provide functionality

Context Diagram - Example



Proposed Solution

- **Primary Features**
  - Essential, High-Value, Follow-on

- **Primary Systemic Qualities (Top 3)**
  - Performance, Throughput and Scalability
  - Reliability and availability
  - Security
  - Manageability
  - Serviceability
  - Usability
  - Etc.

A Solution Architecture is not provided yet; its too premature. The solution is cast in business terms via Features, and Systemic Qualities.

Constraints

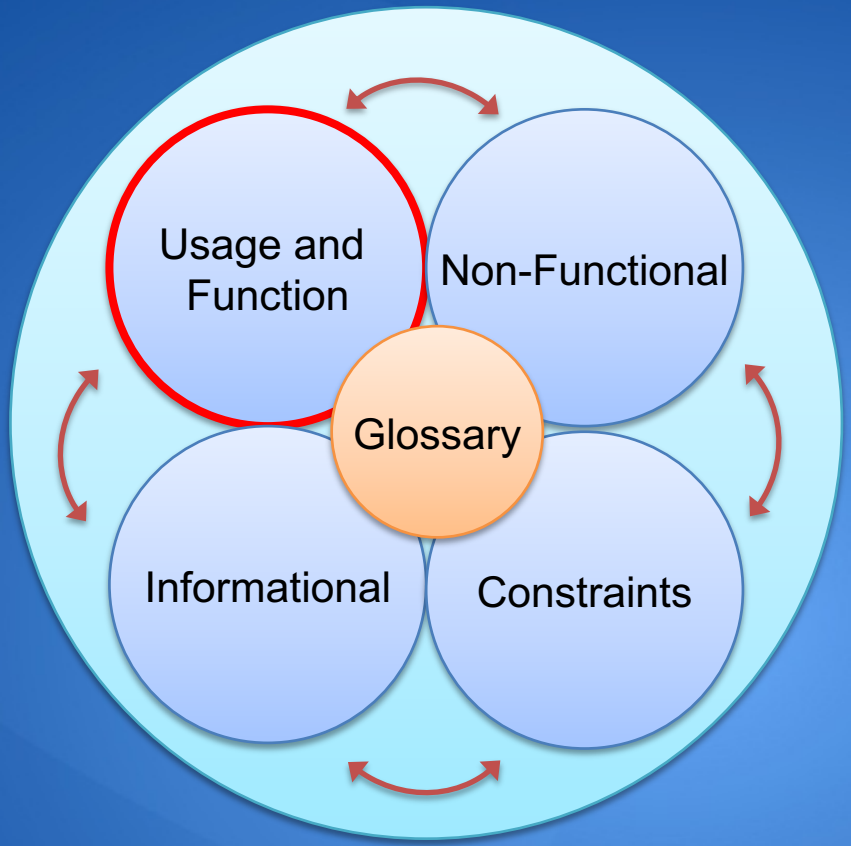
- Development Process and Team
  - Describe the manner in which a project may be formed and operated in order to build the proposed system
- Environment and Technology
  - Describe the environment in which this system will be built and operated
- Delivery and Deployment
  - Describe criteria the system must meet in order to provide a complete solution

Discussed in more detail later on……

Requirements Document

- Constraints
- Service Level Requirements (NFRs)
- Domain Model and Business Rules Requirements
- Business Process
- Operational Process
- Glossary

2 Types of Functional Requirements



Usage and Function

Traditional Functional Specification

Traditional Functional Specification is a list of Features, e.g.

"The system shall statements……."

1) "The system shall utilise function keys"
2) "It is mandatory that the system shall validate all text entry"
3) "The system shall log payments to the account receivable system."

.....

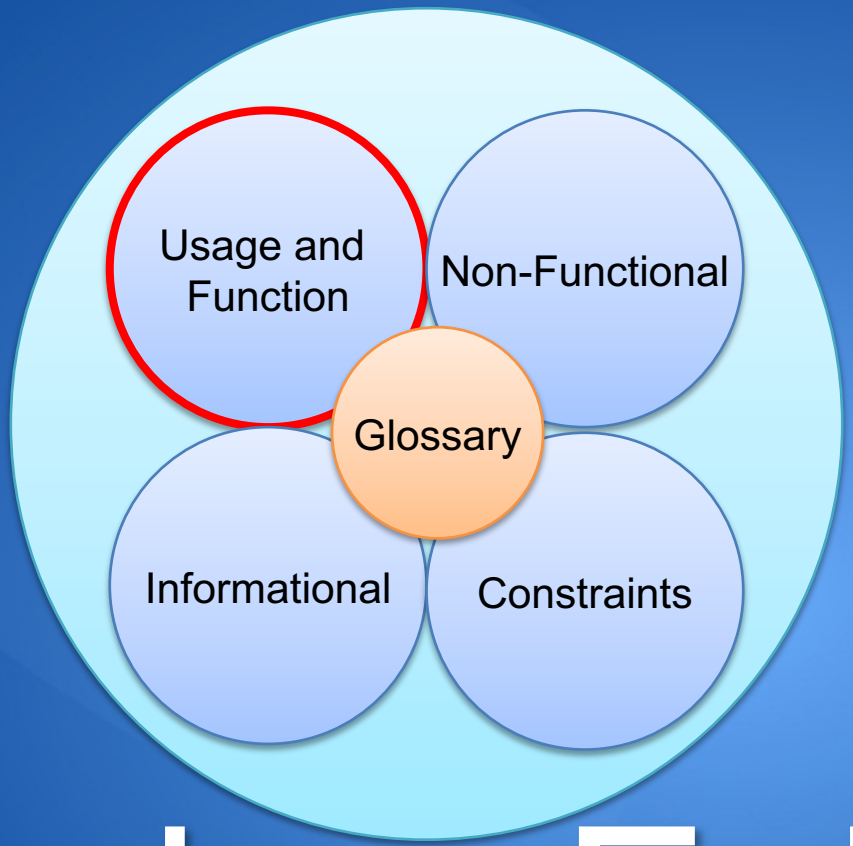
563) "The system shall scale"

Questions that inevitably follow to clarify the uncertainty…..

- What is the context of the statement ?
- Who is initiating the action ?
- Who wants the requirement and why ?
- Why do we need these ?
- How do they provide business value ?
- When do they occur ?
- Who, what triggers them?
- What happens when account receivable system is not available ?

Functional Specification Issues:

- Lack context of their business process:
  - Appear unrelated and difficult to comprehend; vague and arbitrary
  - Do not relate to business happenings that initiate them.
  - Difficult to find, discover, easy to forget functions.
- Hard for business users to prioritise and judge their value.
- Do not relate to / drive development / test process.
- Do not provide units of activity for Project Management.
  - Do not facilitate change management, which ones can we add, drop, change.
  - Can't be developed iteratively.
  - No way to know if they are complete.
- Difficult to know which ones are significant to architecture, thus cannot justify architecture.
- As OO is not functional they are not well suited to OO development!



Stories, Epics, Themes

User Story

Something a stakeholder wants.
Mike Cohen : They are descriptions of a feature told from the perspective of the person that desires that new capability.

Used to be a simple unstructured description (like a functional spec statement) ; now they are commonly templated as:

As a	I want to	So that
Type of User (Actor)	Some Goal (Features)	Some Reason / Benefit
Who	What	Why

- Stories can be split into smaller stories, size is variable
- Also include conditions of satisfaction; high-level acceptance tests that will be true after the user story is complete

An Example Story Template

Title (one line describing the story)

Narrative:

As a [role]
I want [feature]
So that [benefit]

Acceptance Criteria: (presented as Scenarios*)

Scenario 1: Title

Given [context]
And [some more context]...
When [event]
Then [outcome]
And [another outcome]...

*User Scenarios discussed later

Scenario 2: ...

Adding Detail – Mike Cohen

Consider the user story example:

"As a vice president of marketing, I want to select a holiday season to be used when reviewing the performance of past advertising campaigns so that I can identify profitable ones. "

Detail could be added to that user story by adding the following conditions of satisfaction:

- Make sure it works with major retail holidays: Christmas, Easter, Mother's Day, Father's Day, Labor Day, New Year's Day.
- Support holidays that span two calendar years (none span three).
- Holiday seasons can be set from one holiday to the next (such as Thanksgiving to Christmas).
- Holiday seasons can be set to be a number of days prior to the holiday

Non-Functional User Stories

- **As a** customer, I want to be able to run your product on all versions of Windows from Windows 95 on.
- As the CTO, I want the system to use our existing orders database rather than create a new one, so that we don't have one more database to maintain.
- As a user, I want the site to be available 99.999 percent of the time I try to access it, so that I don't get frustrated and find another site to use.
- As someone who speaks a Latin-based language, **I might want** to run your software someday.
- As a user, I want the driving directions to be the best 90 percent of the time, and reasonable 99 percent of the time.

Example from Mike Cohen

Good User Stories – Mike Cohen

Six criteria (INVEST):

- Independent: stories should ideally be independent of the other stories, so as to facilitate prioritization.
- Negotiable: stories are not contracts written in stone. They form the basis of conversations, during which the story may be adjusted, or even removed.
- Valuable to users or purchasers: sometimes the purchaser may have different needs from the users. E.g. a company may have requirements for consistency and cost-effectiveness that are of no interest to the users of the software.
- Estimable: it must be possible to estimate roughly how much time and effort would be involved in fulfilling the story.
- Small: If the story is too big or too small, it cannot be used for planning purposes.
- Testable: the story must be written in such a way that it can be tested. For example, a story to the effect that “the user must find the software easy to use” cannot be tested, whereas “the user must get a response within five seconds” can be tested.

Poor Quality User Stories

- Describe a solution rather than abusiness problem;
- Are just technical tasks or describe internal components of the system, e.g. a product within the solution, fields in a data base.
- Context of the story unclear;
- Provide no clear value to any stakeholder;
- The ‘As A’ user is not the user that gains the value;
- The mysterious user; use ‘User’ but you cannot work out which user;
- Use ‘Product Owner wants”, but they are not a user;
- Have not been shared with the whole team; Has the whole team been involved in the ‘conversation';
- Cannot be objectively tested or verified;
- Simply describe a feature;
- Are large, vague and complex;
- Too many acceptance criteria (e.g. > 5);

Good Acceptance Criteria

- State an intent not a solution, e.g.
  - Good: “The user can choose an account”
  - Bad: “The user can select the account from a drop-down”
  - Very Bad “The account details are stored as records in the system database, with an Account_ID and Timestamp field”
- Check the problem and business requirements not the design or solution.
- Are independent of implementation, e.g.
  - The phrasing would be the same regardless of being implemented on web, mobile or a voice activated system
- Are relatively high level (not every detail needs to be in writing). System details and design are never described.
- Do not restate the story narrative, don’t include new stories or contain entire workflows.
- Have one user type that matches the story user. All the criteria apply to the story user type. The user is specific and not generic, there are not different acceptance tests (scenarios) for each user sub-type.

Who is the ‘As A’ and Why ‘So That’ ?

A common problem with user stories is that behaviour is often attributed to the wrong Actor, one that does not interact with the system or achieves no benefit.

- As a user I want to enter my details into the system so that I can receive marketing communications.

However, the user has no requirements but really the stakeholder does:

- As a Marketing manager I want the user of my system to enter their details so that they can receive marketing communications.

However is this really a good reason why the manager wants this, does it explain the business goal.

- As a Marketing manager I want the user of my system to enter their details so they can receive marketing communications that will improve the upsell of our products.

Do you really ‘Want to’ ?

However the marketing manager is not a user, so is this a ‘user story’ ?
The ‘requirement’ actually appears elsewhere in the business process, where the manager is a user, gains benefit, and is the owner of the business value:

- As a Marketing Manager I want to be able to send marketing communications to customers to improve the upsell of our products.

A consequence of this is users are required to somehow enter all their details even when they don’t want to. To achieve the goal the user may not ‘want’ to fill out the form, but has to, so is the ‘user’ story:

- As a user ***I have*** to enter my details **(even though I don’t want to)** into the system so that I can receive marketing communications.

Above is assuming the user has a goal to get marketing communications but is happy with a poor experience. However it is hard to attribute this goal to a user ?

Issues with Stories (as an approach)

- Arbitrary size level; constrained by team, e.g. “no story can be expected to take more than 10 days of development work”.
- Can be split add infinitum.
- Difficult to not capture the solution rather than the problem , e.g. *As a user I want a phone with a sliding cover so that I do not accidently bump the keys*. This is a solution.
  - Acceptance criteria often detail and test a solution not the problem;
- They are not an objective record of requirements:
  - ***Oral stories*** are more valuable than ***written stories***.
  - The most useful stories are ***minimalist in form.***
- Many stories loose context and become a random list of features:
  - Hard to understand overlap, or coverage of the domain.
- Difficult to capture usage, workflow or sequential human interactions as with web pages or graphical user interfaces.
- Difficult to scale and communicate, as they require a ‘conversation’.
- Conversation be objectively tested or verified;

User Stories are not Requirements

A user story is an invitation to a conversation.

Ron Jeffries Stated:

“The requirement itself is communicated from customer to programmers through conversation: an exchange of thoughts, opinions, and feelings. This conversation takes place over time, particularly when the story is estimated (usually during release planning), and again at the iteration planning meeting when the story is scheduled for implementation. “

Mike Cohen Stated:
“Stories are not intended to document user needs. The story is a means to having a conversation.”

User Stories are not Stories

- A User Story has no plot line, unlike a use case, it has no steps.
- A use case follows a sequence of steps:
  - It tells the story of someone using the system to accomplish a goal.
  - It has more than one step.
- A user story is sometimes the title of one use case scenario.
  - A use case is the contents of multiple scenarios, it is more abstract
- A Use case records the decisions of a conversation.

Issues with Stories – Continued

- Often difficult to determine who the user is, and if they really are the owners of the stories goal.
- Difficult to scale and communicate, as they require a ‘conversation’
- Conversation cannot be objectively tested or verified;
- Adding Scenarios to Acceptance tests means stories are really no different to use cases.
- Acceptance criteria duplicates context across many stories, and the overlap is not clear.
- Can be used to, force fit, and cover every type of requirement
- NFR-User Stories are hard to place on backlog as not independent:
  - Hard to know what Functional Stories they impact.
  - Or are these NFRs part of Acceptance tests ?
  - They do not represent discrete functional behaviours that produce a valuable result.

- The difference from “Features” or ‘Traditional Func Specs’ is very subtle. Share many of the issues with Traditional Functional Spec.

Multiplicative Complexity

In Agile approaches Stories and Features are used to manage work sequentially; however, often they are not independent;

- Some features result in additive complexity, they can be added easily, independently of other features;
- Other features result in multiplicative complexity; each new feature is not independent and requires larger amounts of re-work of prior features.
- Dependencies will arise across Stories and this across items on the backlog;

Features interact and understanding one feature and implementing it, means this work has to be undone ad re-done to develop the next feature, and so on.

Jackson-Zave Distinction

Some requirements describe the domain;
Other requirements describe the ‘machine’ or system;

User stories confuse the two of these.

Epics

- Mike Cohen: Simply a Large Story that will be broken down into smaller stories;
- Epics are a high level description of a Business Need;
- Epics can have two types:
  - Business
  - Architectural

- Our use of “Epic” is a business digestible statement that collects and categorises Use Cases and Stories.

Themes

- Mike Cohen: A collection of User Stories
- They are key units for funding and strategic Investment;
- Cross systems, applications and not yet initiative or project based. Apply to IT as a whole.

- Our use of ”Theme” is that they are key value propositions that drive vision and used for high level prioritisation and funding.

Features

IEEE standard (1990) defines the term feature as:

- (A) A distinguishing characteristic of a software item (for example, performance, portability, or functionality)
- (B) A software characteristic specified or implied by requirements documentation (for example, functionality, performance, attributes, or design constraints).

- In English it is “a distinctive attribute or aspect of something”
- Features can be viewed as aspects of the system that are orthogonal to usage or workflow; Thus User Stories cannot capture usage or workflow.
- A feature is a distinct element of functionality which can provide capabilities to the business. Features are important because they allow a customer to more readily express their needs.
- In the development of a product line they are often used to distinguish between products and establish product configurations.
- Features express requirements in terms of the solution.
- Are an orthogonal aspect to stories and epics, not hierarchical.

Examples

Theme: Web customer management.

Epic: Allow the customer to manage its own account via the Web

Feature: Edit the customer account information via the web portal

User story: As bank clerk, I want to be able to modify the customer information so that I can keep it up to date.

User story: As a customer, I want to pay with the most popular credits cards, so I not inconvenienced.

Feature (as a constraint) support the GOV-TAX-02 XML API of the government.

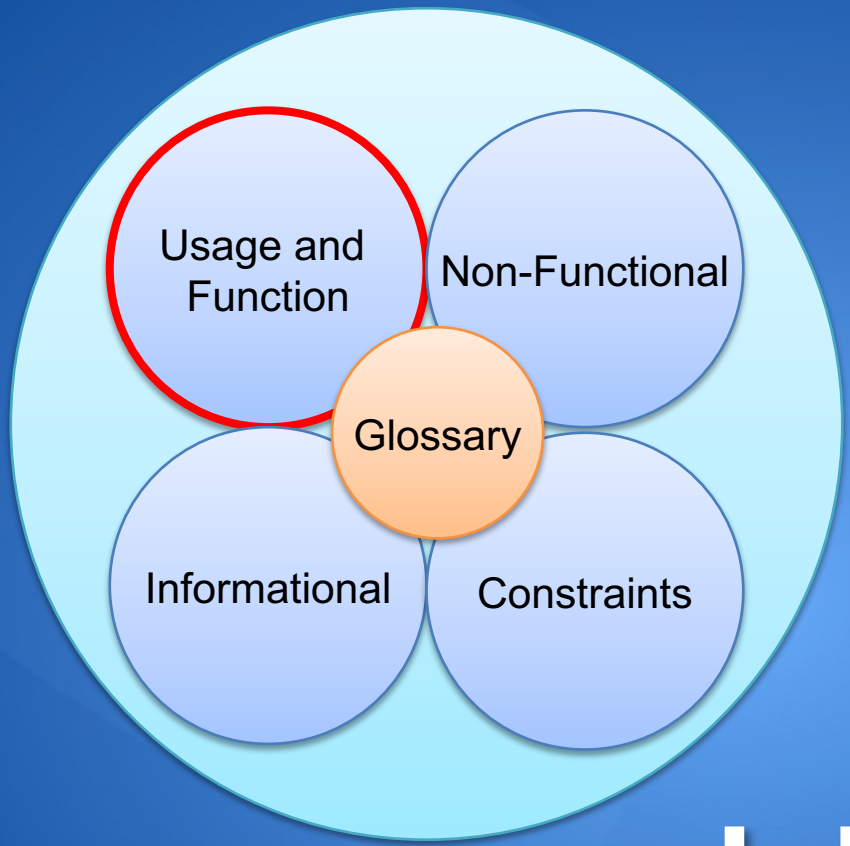
Summary

- In popular usage and across different methodologies, including the many Agiles; Theme, Epic, Story, and Feature have loose, varied, arbitrary, and confusing meanings;
- These items often, unfortunately, specify a solution;
- OP++ makes the assumption that an objective and documentable form of requirements is desirable rather than a conversation starter;
- The approach is to create definitions (constraints) that help use these terms in a collectively understood manner.

OP++ Requirements Items, include:

- Features are used to describe to the business, the tick box of characteristics for, the vision of a system.
- Use Cases capture the functionality and are sliced into User Stories for development.
- Non Functional Requirements capture the business and architectural SLRs. In many cases covered by legal contract.
- Constraints are used at the business vision and system level.
- Informational Requirements are as critical as User stories, to gain coverage of stakeholder needs.

- Themes and Epics are not used as they are too vague. Business Uses Cases better cover this level.



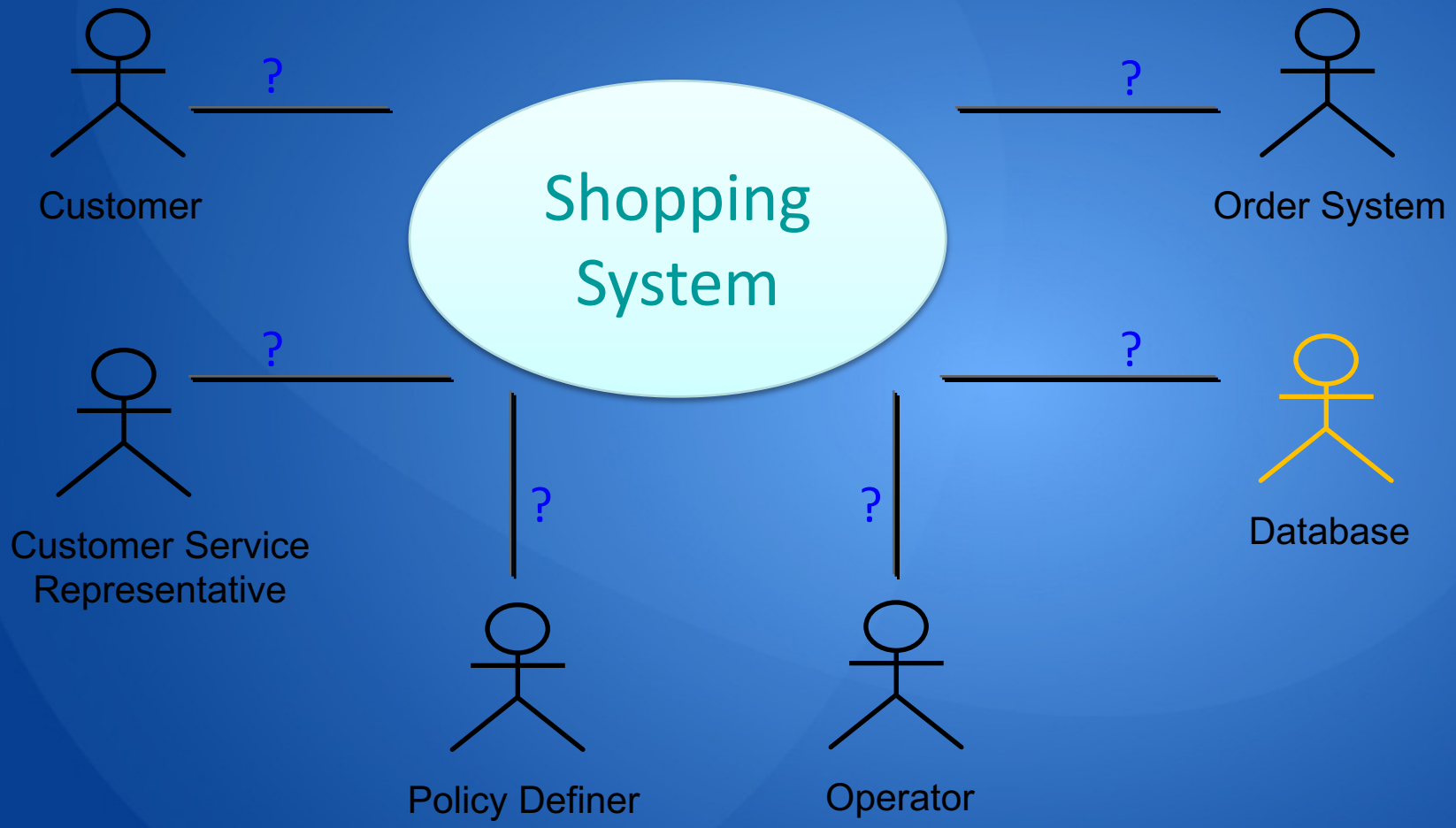
Actors and Use Cases

What is an Actor?

- Actor is someone or *something external* to the system that *directly* interacts with it
- Actors define the boundaries of the system to help narrow down scope

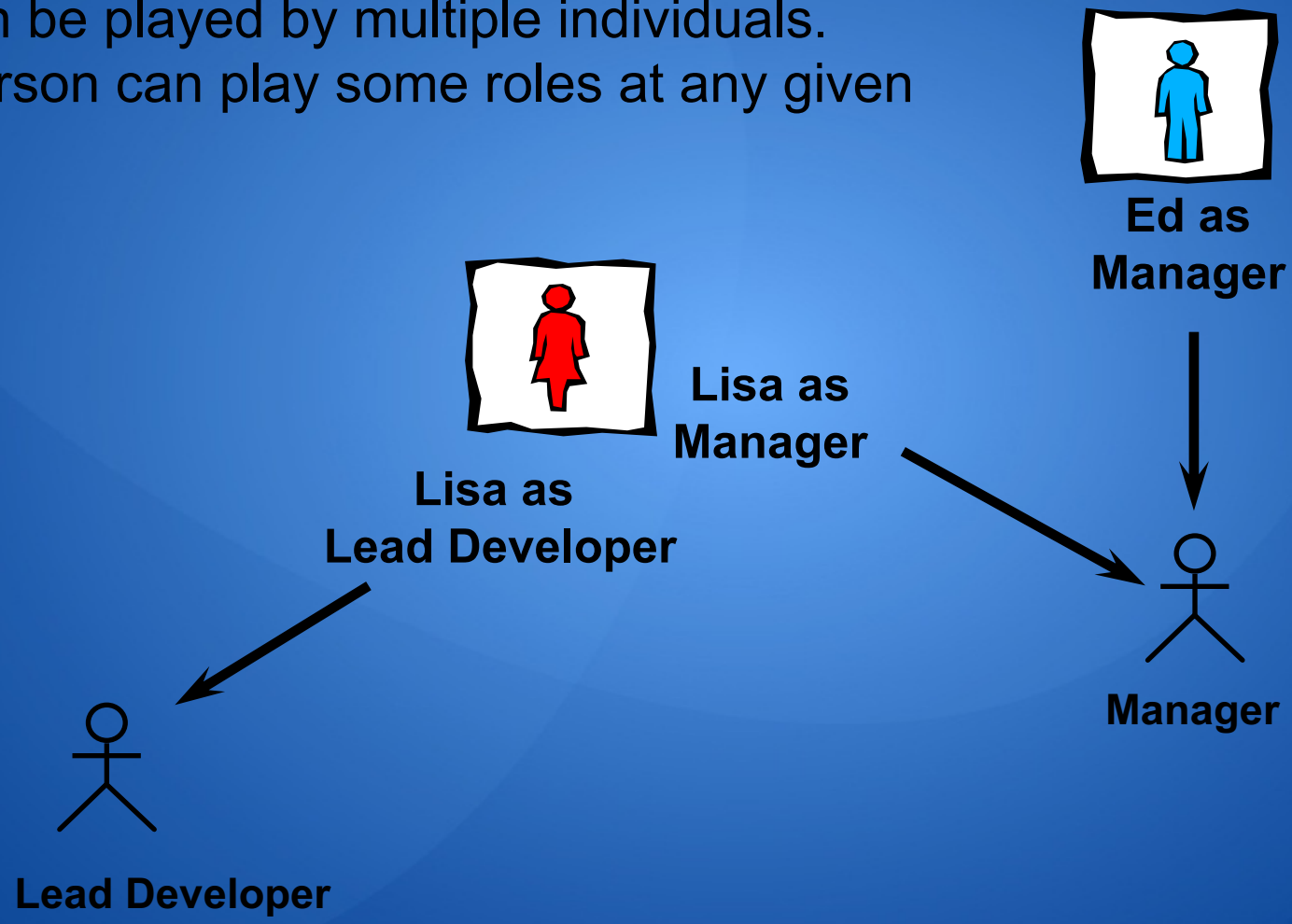
Actor

Identifying System Boundaries



Users (Personas) vs. Roles

- One person can play multiple roles.
- One role can be played by multiple individuals.
- Only one person can play some roles at any given time.



Actor Categories

EVENTS

Actors may be:

- **Active** : inbound actors that the system exists to serve; they initiate communications with system
- **Passive** : outbound actors that the system cannot exist without; they respond to requests that the system initiates.
- **Both** active and passive at the same time

Types of Actors

Actors		Active	Passive	Both
Human	Business	✓		
	Operational	✓		
External System		✓	✓	✓
Time		✓		

- Operational Actors - those who keep the system running
- Business Actors - those who benefit from using the system

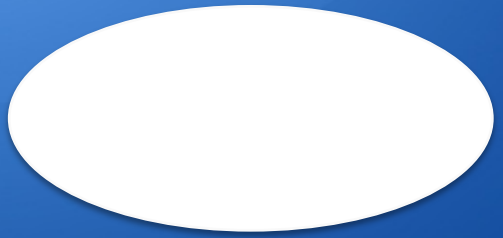
Documenting Actors

Actor list – enumerate all the actors, for each:

- Actor Name
- Description:
  - Skill level
  - Physical relation to the system network, especially internal use vs. external use
    - Protocols that must be supported
  - Frequency of use

What is a Use Case?

- A Use Case is a modeling technique that:
  - Captures dialogs (a sequence of messages) between a system and its environment (actors)
  - Defines a sequence of actions a system performs
  - Encompasses a complete chunk of functionality
  - Yields a result of *meaningful* value to an actor; achieves a goal; Walk Up and Away Test
  - Consists of a set of scenarios - a real-world sequence of operations that describes how an element works in real-time.
  - Includes at least one active actor
  - Provides test cases



Use Case

Benefits of Use Cases

- Facilitate coverage of business processes
- Demonstrate value of “features” to business and users
- Readable by all stakeholders
- Drive Process with Use Cases:
  - Lead directly to test cases (coverage and verification)
  - Are the units for iterations
  - Used as units of estimation
- Facilitate Project Management
  - Units for change management.
    - Independent groupings of functionality
    - Add/drop/change functionality; in meaningful groups.
  - Basis for planning around iterations.
- Architecture is justified by selected set of “significant” use cases.
- Focus work efforts on end results:
  - Can be developed Iteratively (and continuously);
  - Business value of work easily determined;
  - Avoid undue emphasis on intermediate results;
  - Avoid scope creep by defining system boundary;
  - Avoid premature design by specifying what the system should do rather than how to do it.

User Story V Use Case

"A user story is synonymous with “feature” as used in the 1990s, a marker for what is to be built, fine-grained enough to fit into modern iteration/sprint periods. A use case provides a contextual view of what is to be built, serving to bind the organization together, among other things. They have different properties, suiting different purposes. Use each according to its properties. “

Alistair Cockburn

"A user story is the title of one scenario, whereas a use case is the contents of multiple scenarios.“ Thus, where I have oft been quoted as saying, “A user story is a promise for a conversation”, ... that begs the question “about what?” ... which is answered, “about the scenario for which this is the title!”

Alistair Cockburn

My one liner is that a story is a promise to have a conversation and a use case is the record of the conversation. If you think you need one.

Jim Standley

Very nice.

Alistair

In UP Use Cases Drive the Process

"…the use case model…..describes the complete functionality of the system. This replaces the traditional functional specification.

A func. spec. can be said to answer the question. What is the system supposed to do? The use case strategy can be characterised by adding three words to the end of this question: *for each user?"*

Use-case driven means the development process follows a flow – it proceeds through a series of workflows that derive from the use case."

Use case are specified, use cases are designed, and at the end use cases are the source from which the testers construct test cases.

While it is true that use cases drive the process….they are developed in tandem with the system architecture….the system architecture influences the selection of the use cases.

Jacobsen et al "The Unified Software Dev. Process", p 5.

Main Types of Use Cases

- **Functional Use Cases :**
  - **Business Use Cases -** support the business actors, describe process and workflow.
  - **Operational Use Cases -** support the operational actors process and workflow, considered important to manage or keep the business running.
- **Non-Functional Use Cases :–** describe how to carry out or test a non-functional requirement. (discussed later)
- **Special Use Cases:**
  - **Infrastructure –** extend the above use cases to capture certain types of non-functional requirements, e.g. security, logging, auditing.
  - **Integration –** capture interactions when both actors are systems, also describe processes involved, orchestrations.

Application Use Cases - Examples

- Business Use Cases:
  - Check Order Status
  - Place Order
  - Maintain Shopping Cart
- Operational Use Cases:
  - Install and Configure System
  - Troubleshoot System
  - Start and re-start a System
  - Monitor System Alerts
  - Import Customer Data
  - Maintain User Accounts
  - Age Logins
  - Backup or Restore system
  - Status Enquiry
  - Version Upgrades

Operational Use Cases

- Cover Use Cases that are considered important by the business to operate and manage the Business Use Cases; where the goal is not so much a business function but a NFR goal.
- Describe how Generic Non-Functional Requirements are actioned.
- Need to consider the current Standards (Constraints) in the Data Centre:
  - Monitoring systems.
  - Start/Stop Procedures
  - User Administration Systems and Procedures.
  - Backup and Restore Procedures.
  - Fault and Diagnosis.
  - Uptime Reporting.

Finding Use Cases

EVENTS

Type of benefit		Examples
Information	Sent from an actor	•Send a message •Set preferences
	Received by an actor	•Search for supplier info •Send fault alarm (i.e. system sends alarm to user)
Transaction completed / changing the system state		•Finalize pending order •Restart server
Process Initiated	External to the system	•Pay bill (to 3rd Party) •Control device
	Scheduled activity within system	•Dump nightly batch file •Run system diagnostic (initiated by timer)

Documenting Use Cases

- **Start Here:** Develop a high level use case list, of:
  - use case names
  - with brief descriptions (one or two sentences)
- **Then:** Select use cases to focus first iteration of work on.
  - Develop these use cases details, e.g. action steps
  - Use discretion on detail, try to be lean.
- **Then:** Develop use case model
  - May group use cases in packages
- Then develop specific User Journeys to test business/marketing sensitive cases.

Breadth First Before Depth (Detail Iteratively)

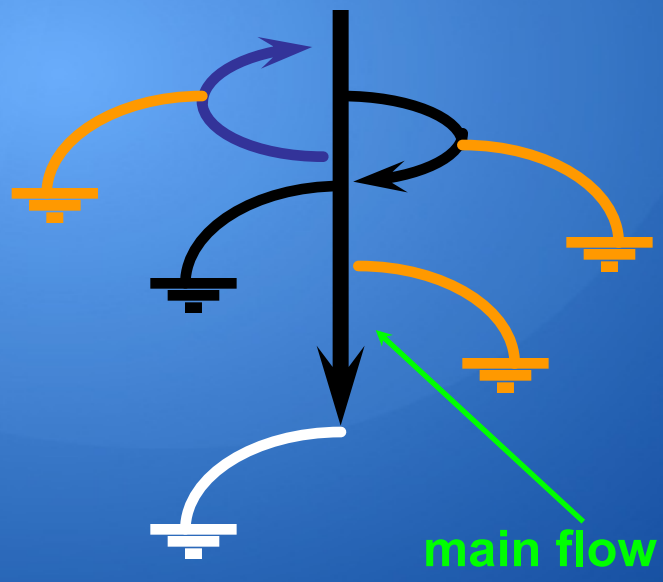
Use Case Details – Template

Attribute	Description (use discretion in selecting attributes to use)
ID	An Identifier, UC-<project number>-[B|O]-<unique identifier>. B = Business. O = Operational
Name	A short, descriptive name usually verb-noun
Description	A short summary description of the use case, 1-2 sentences
Trigger	A business or operational event, condition or stimulus that starts the use case.
Actors	The source of the trigger. Only the ones initiating the use case (active)
Goal	A description of the goal of the Use case; what the actor is trying to achieve
Actions: Main Flow	The normal sequence of actions; the “happy path”
Actions: Alternate Flows	Exception and variation paths, alternate actions
Pre-conditions	What must be true prior to starting the use case
Post-conditions	What becomes true after use case ends
Special Requirements	Link to generic NFRs or NFR Use Cases
Time Frame	Required now or a future state and evolutionary goal
Priority	Business priority (high, medium or low)

Use Case Flow of Events

Each of below is a ‘Scenario’, a use case is a collection of different flows - scenarios:

- One main flow
- Several *alternate flows:*
  - Regular variants
  - Rare or unusual cases
  - **Exceptional flows**



Scenario - a real-world sequence of actions (operations) that describes how an element works in a real situation.

Documenting Use Case Flow of Events

- The main flow should be readable from top to bottom;
- Start off by describing how an actor starts the use case;
- End by describing how the use case ends;
- Generally alternate between user and system action or between system interaction across different actors;
- Enumerate scenarios under separate sub-headers in the alternate flow section;
- Write from the point of view of the actor and in the active voice
- Focus on "WHAT" the behaviour requirements are, not "HOW" to achieve them;
- May reference other use cases;
- Indicate the passive external actors the system may be interacting with;
- Avoid describing data items in detail;
- Avoid ties to the user interface design or channel device.

Use Case - **Print TSBs**

1 Print TSB's

Allow technician to print TSB's and Screen images to a local printer.

1.1 Actors

1. Auto technician

1.2 Main Flow of Events

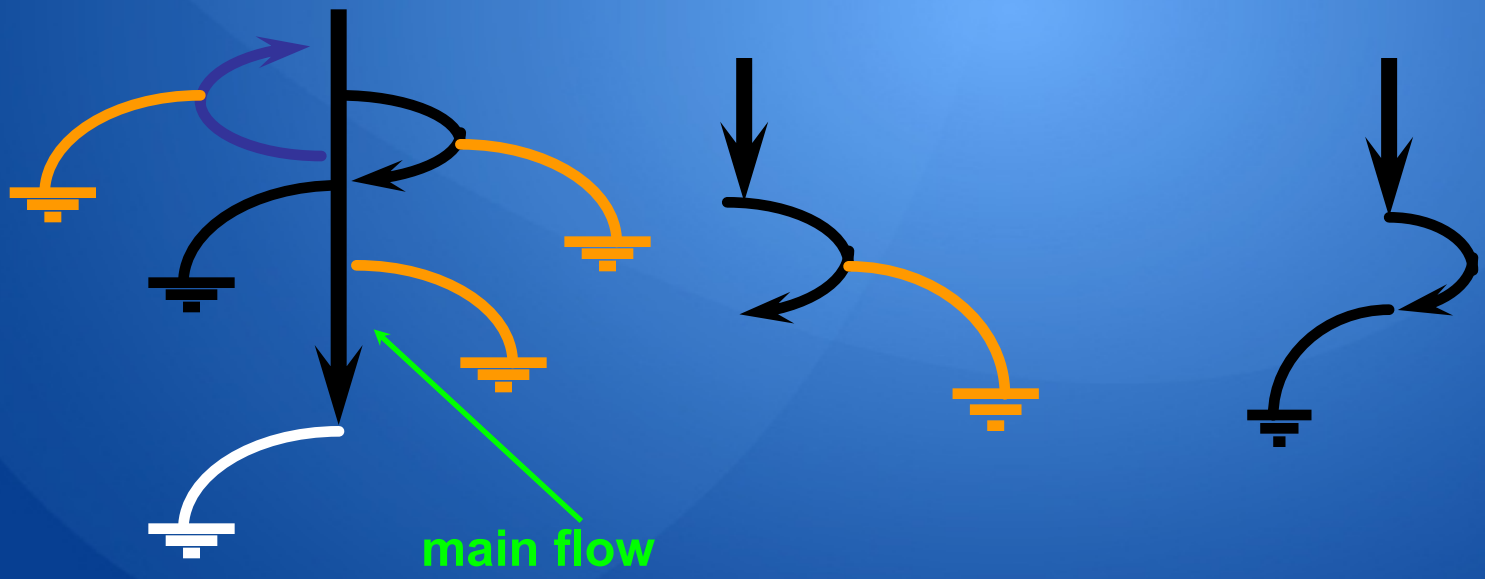
1. The technician selects the Print Document function.
2. The system displays a "Print" dialog.
3. The technician selects the desired print function from the "print" dialog.
4. The technician processes the print request and exits the "Print" dialog.

1.3 Alternate Flows

3a. The technician cancels the print request and exits the "Print" dialog.

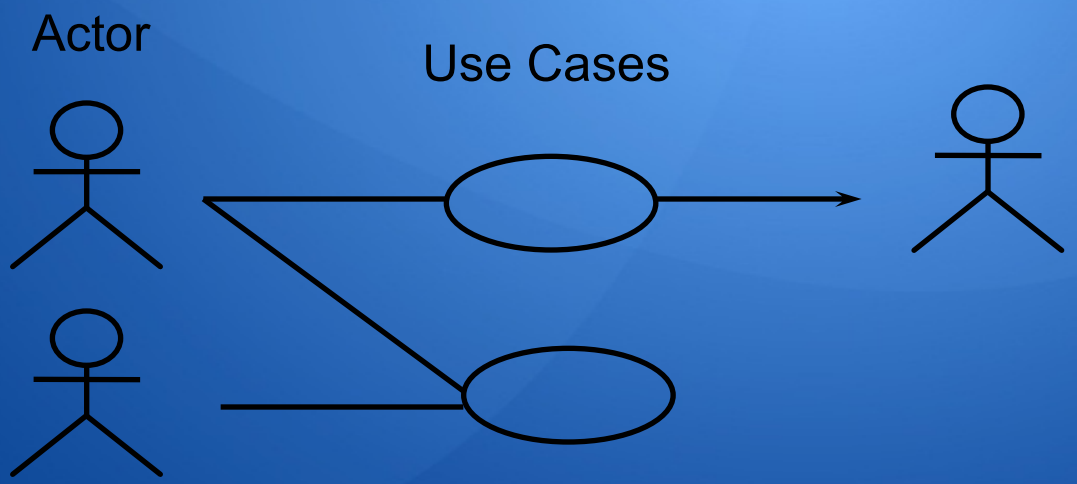
What is a Use Case Scenario ?

- A scenario is an instance of a use case or one of its alternate flows
- Concrete example
  - e.g. “Sue logs into the system and chooses value X …”

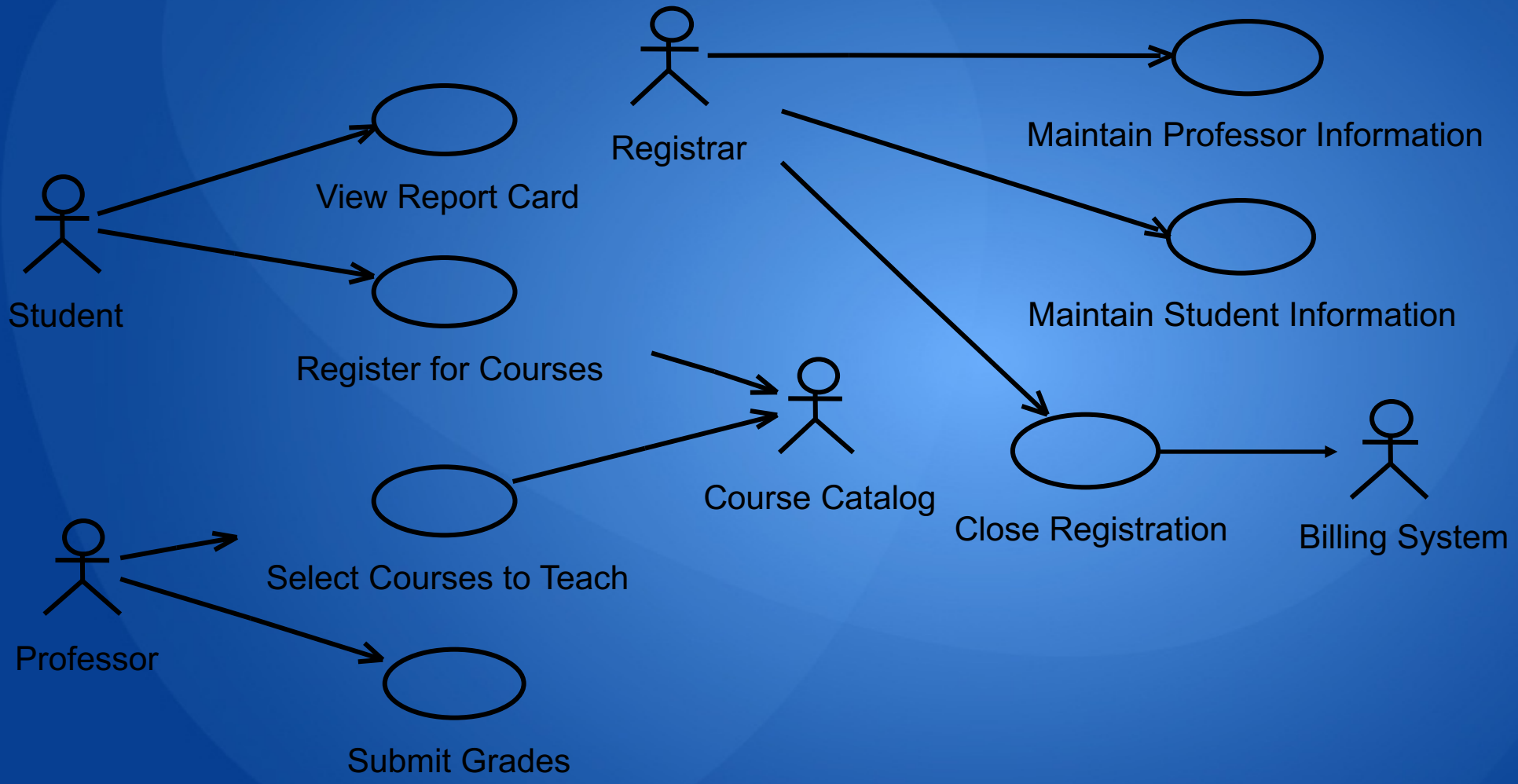


Use Case Model

- In some cases, the relationship between actors and use cases tends towards many-to-many, in other words, several actors are related to each use case;
- Considers each actor's perspective;
- Shows active actors that initiate use cases;
- Exhibits complexities in the system;
- Optional way to illustrate use cases and relationship with actors.

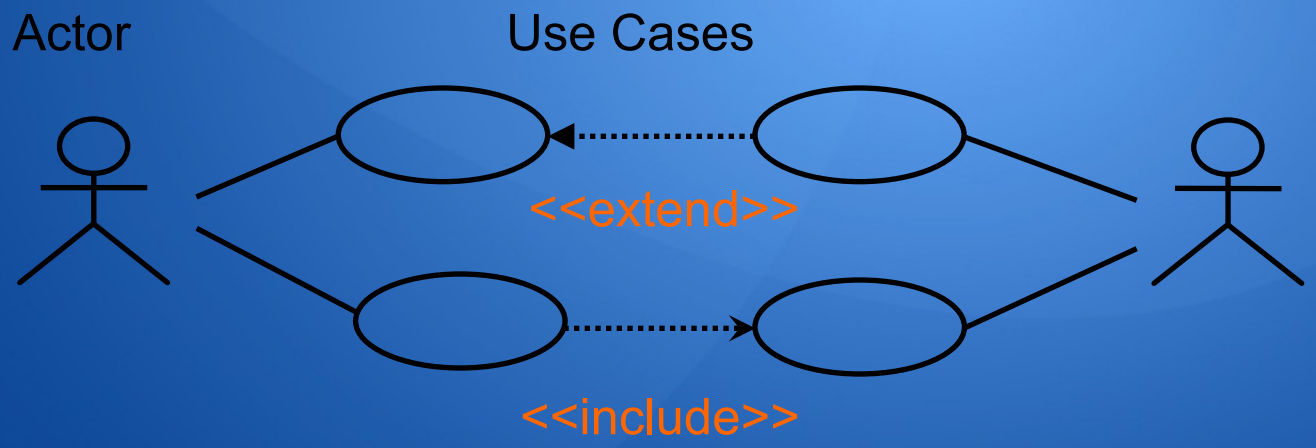


Example UC Model – (TBD)



Use Case Model Relationships

- Use cases can have relationships between them:
  - Extend
  - Include
- Model shows these relationships with stereotype <<…..>>



Use Case Relationships

- Included Use Cases:
  - Are a *big* step in the flow of a use case that is common to more than one use case;
  - The main use case is not complete without this use case;
  - Document the included use case details separately.
- Extending Use Cases:
  - Are additional or extra steps that may be taken when certain conditions are met e.g. “Item not in stock”;
  - Think of it as a *very* different and complex alternate flow of the base use case that *may interrupt a number of steps in a scenario*; As it affects a number of steps easier to understand both original flow and alternate flow if they are separated;
  - Suggests a Business Rule, a logic choice;
  - Document the extending use case details separately.

Alternate Flow / Extension

Main Flow

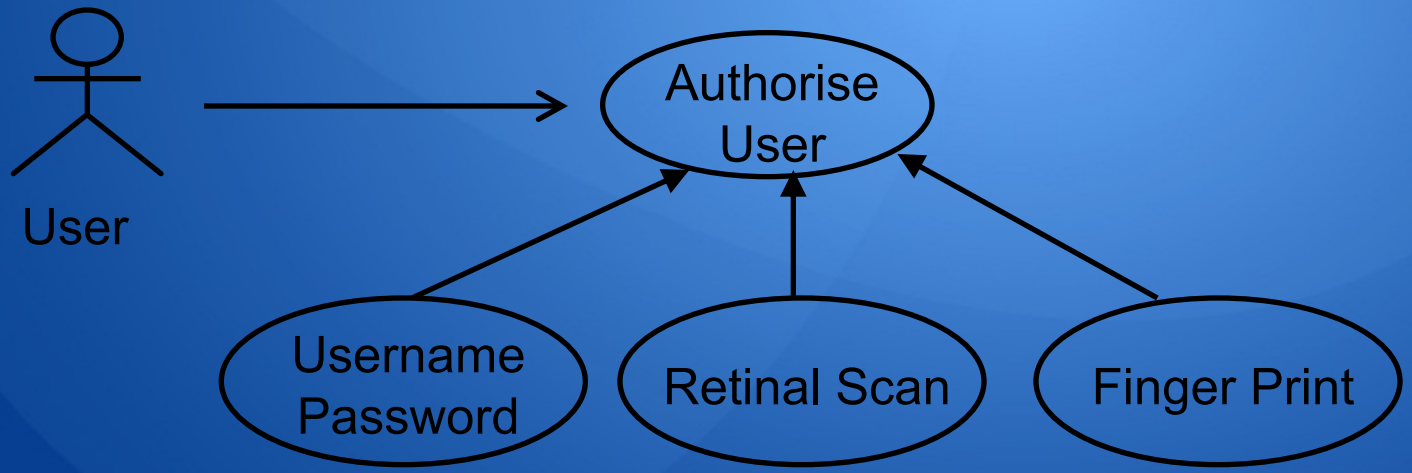
1. The customer inserts the card.
2. The system validates the card.
3. The customer enters the PIN.
4. The system validates the PIN.
5. The customer enters the desired amount of money.
6. …

Alternate Flow or Extension

4a. PIN has been incorrectly entered for the first or the second time:
4a1. The system logs the attempt.
4a2. The system notifies the customer.(Rejoin at 3)
* The customer cancels session (anytime)

Generalisation Relationship

- Having many individual use cases can scatter important behaviour
- Consider creating an Abstract Use Case and put each variant into a specialised use case.
- Employs a use case Generalisation Relationship



Use Case – **Browse TSBs**

2 Browse TSB's

The Technical Service Bulletin (TSB) is browsed by the user to access the latest technical reports and service information for a vehicle. TSBs include the specific description of the problem, the date of issue, vehicle code and the subject. .

2.1 Actors

1. Auto technician

2.2 Main Flow of Events

1. Select TSB tab.
2. <include> Browse TOC use case for Table of Contents navigation
3. Selecting a TSB from the list of TSBs within a "Category" opens up the TSB document.
4. The user browses the TSB document.

2.3 Alternate Flows

2.3.1 <u>Selecting a hyperlink within a TSB document</u>

4a. Opens up a related Graphic image. (Refer to "View Image" use case.)

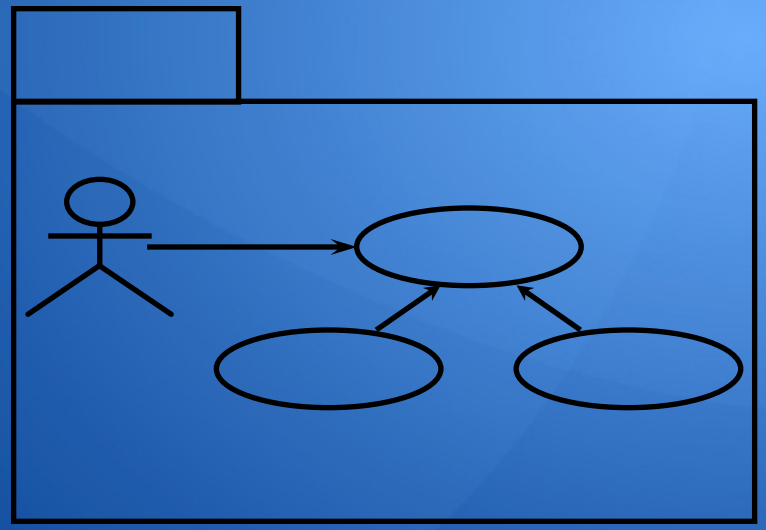
4b. .....

CRUD Use Cases

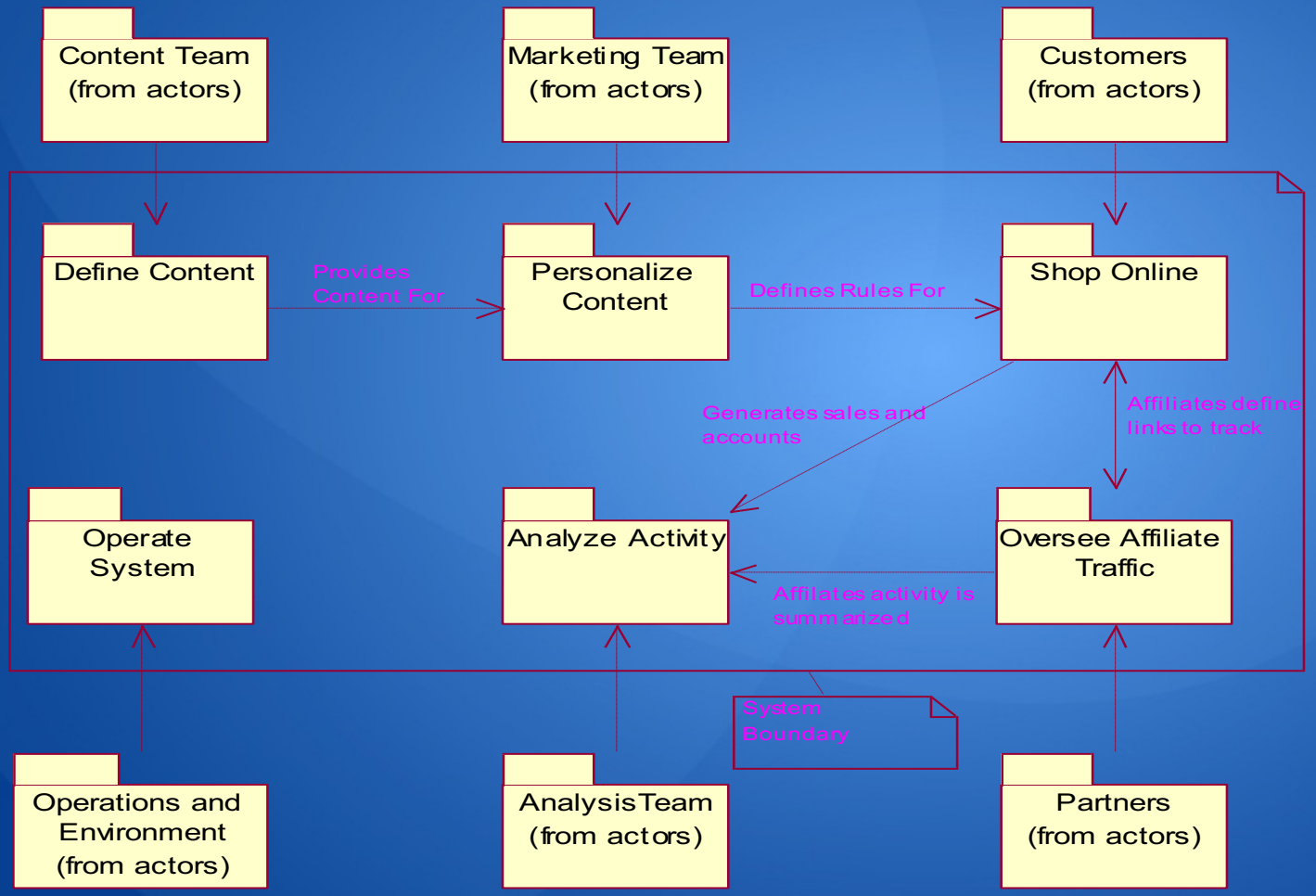
- CRUD = Create, Read, Update, Delete
- CRUD use cases are generally combined in one use case called “Maintain”
  - e.g. “Maintain Order” encompasses creating, reading, updating and deleting the order
  - Other features may also be encompassed in the “Maintain” use case such as copying and finding
  - Document features as alternate flows or part of the main flow of the “Maintain” use case
- Always start off with a ”Maintain” use case, consider alternate flows for each operation of CRUD only if required.
- Separate a feature into its own use case
  - if started by a different actor
  - if you discover a low-level included or extending use case

Use Case Packages

- Help in organizing the use cases of a large system by functionality category
- A use case package contains use cases, actors and possibly other use case packages



Example UC Packaging



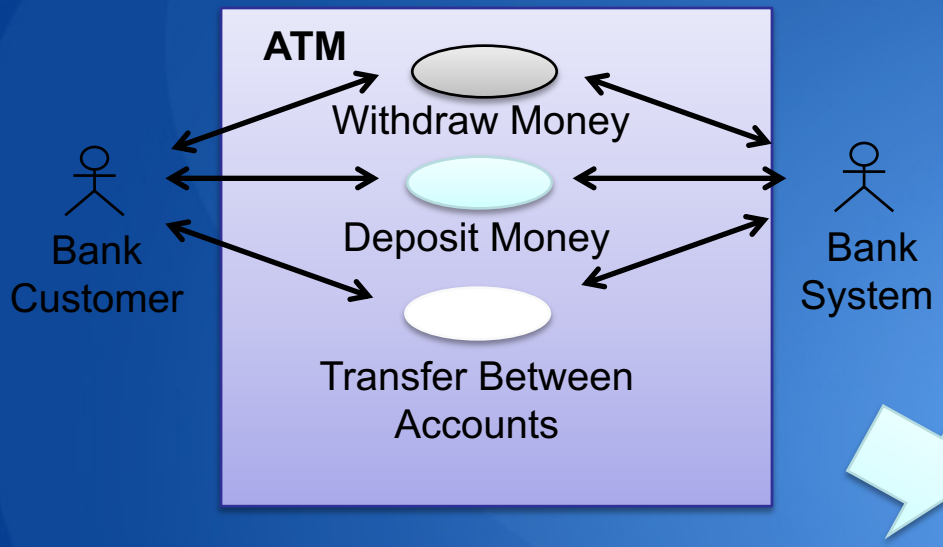
Common Use Case Problems

- Inconsistent or incoherent naming;
- Missing or poorly-worded summaries;
- Not treating the system as a black box in the flow of events;
- Incorporating design elements into use case descriptions;
- Overuse of use case relationships;
- Overuse of pre and post conditions;
- Aiming granularity too low; having too many very small use cases;
- Trying to make up for poor domain models with excessive verbosity in use case descriptions;
- Substituting screen shots for poor use case descriptions;
- Cross-cutting concerns – see extensions & infrastructure use cases
- Overuse of interaction diagrams in use cases.

Use Case Variability

- Each use case flow describes a particular variation; but how much variation is there ?
- Need a systematic way to determine variation otherwise you will miss requirements.
- Examples, how many customer types, what are the different channels system will support;
- Alternate flows can be organised by use case variables, a section for each one.
- Variables are important as system extensions may be based on them. e.g. new customer type, new channel.

Use Case Variability Example

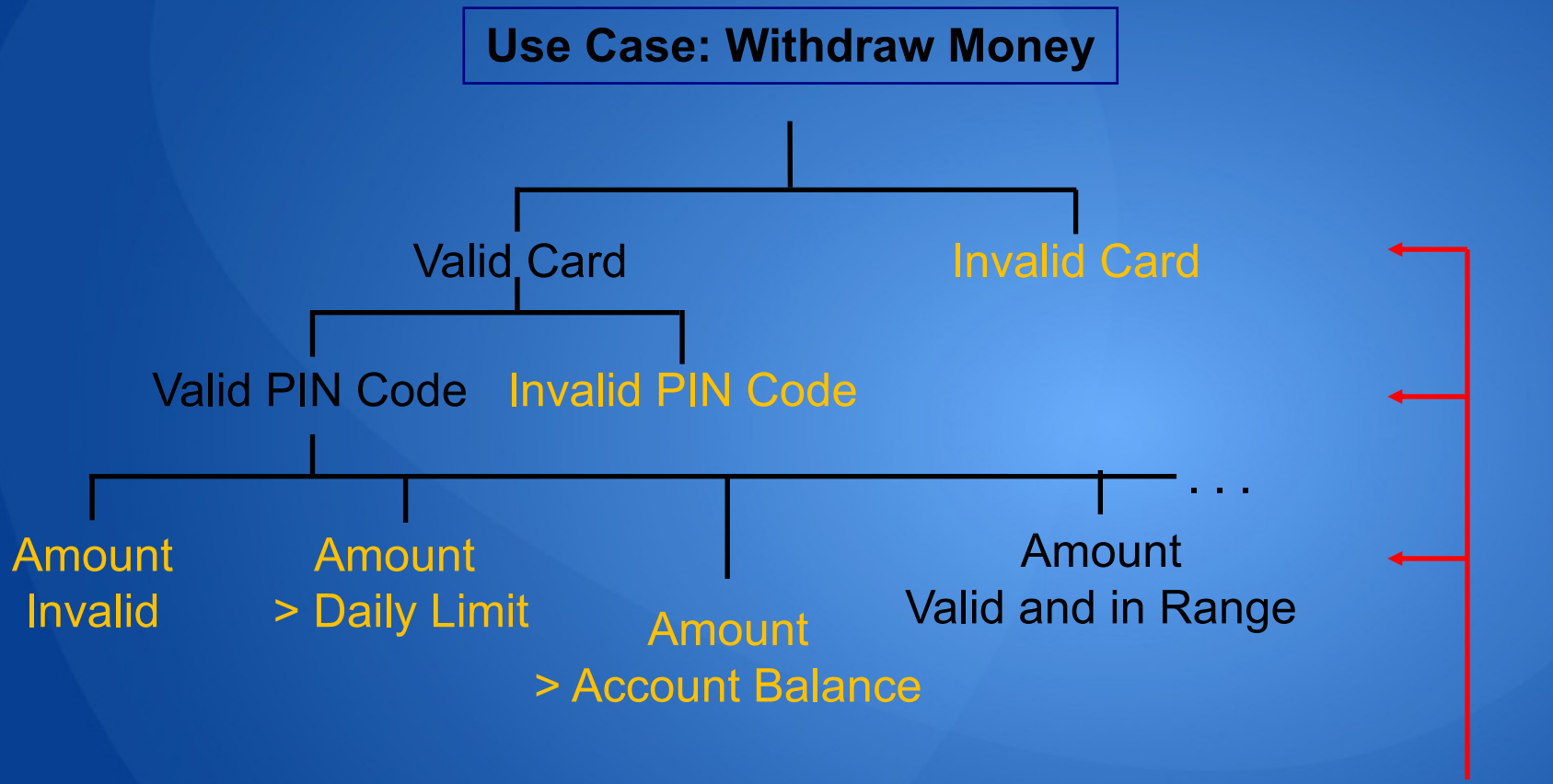


Use Case Scenarios

Cash Withdrawal of a pre-set amount

Cash Withdrawal of custom amount

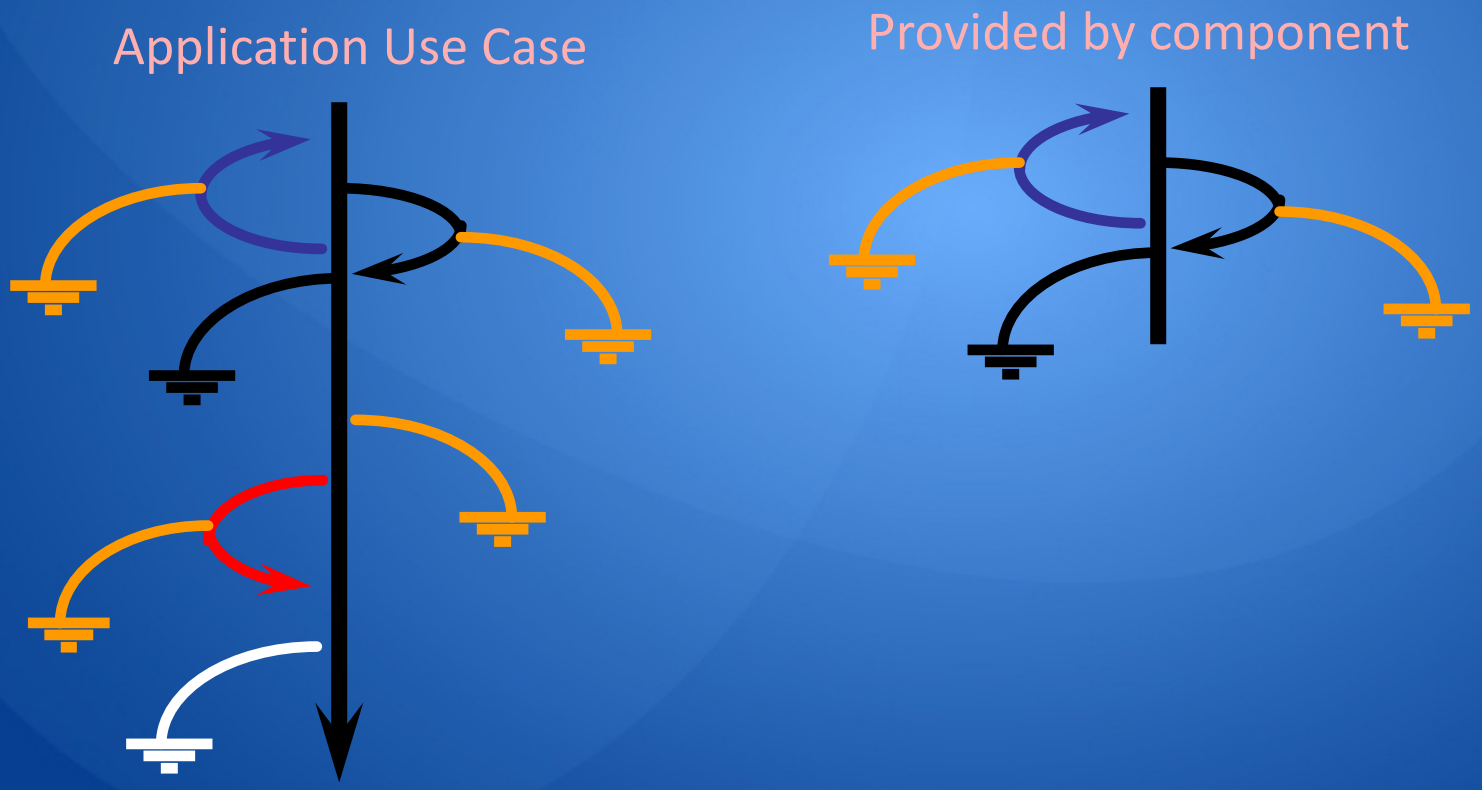
Variables help find new scenarios



Orange - Values not considered before

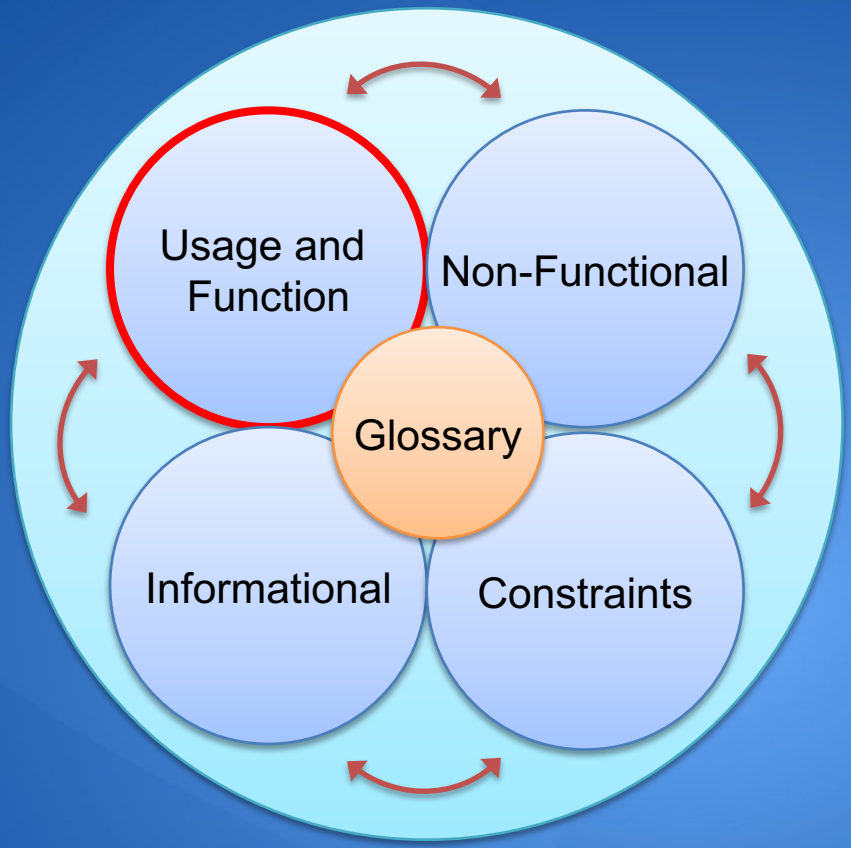
Use Cases for Components or Services

- A Component / Service (e.g. purchased product) has use cases that should map onto/over Application use cases.



Mitigating Functional Decomposition

Problem	Mitigation
CRUD elements are dealt with in separate use cases.	Make CRUD functions alternate flows in one manage user use case.
Use cases deal with specific concrete users.	The users should be generalised to actors, and then the use cases should be generalised where they deal with same functionality. User Journeys deal with Specific users.
Use cases deal with specific functions and don't by themselves complete a single transaction	Re-write the basic flow, merging current use cases, to complete an end-to-end scenario.
Use cases are locked to specific system or user interface components or screens	Systems should be treated as black boxes and references removed. Remove references to UI technology and specific screens.
Use cases have no alternate flows	Deviation to a main flow are treated as alternate flows not separate use cases. Treat single use cases with no alternate flows as suspect and review.



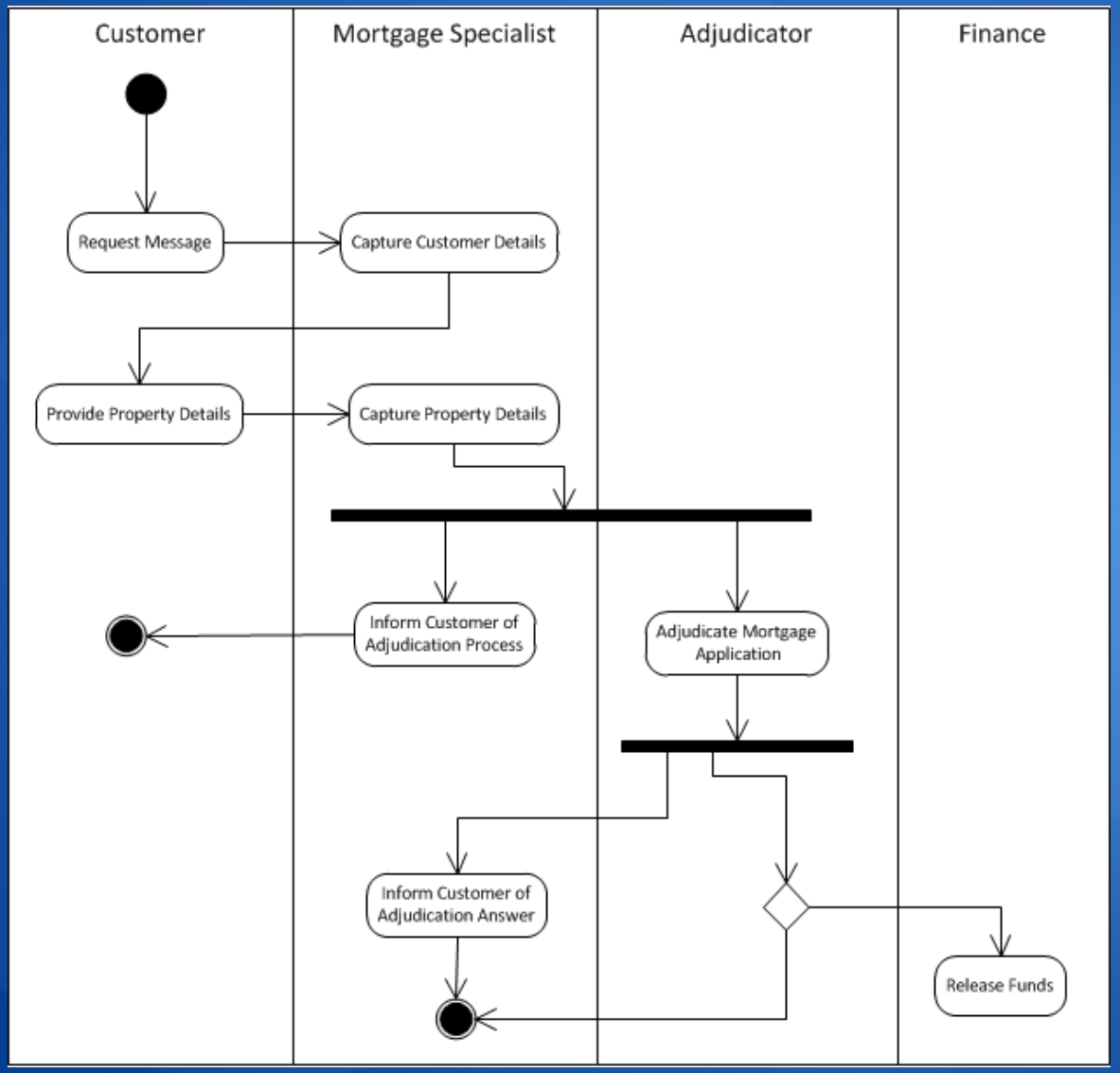
Visualising

Techniques to Visualise Use Cases

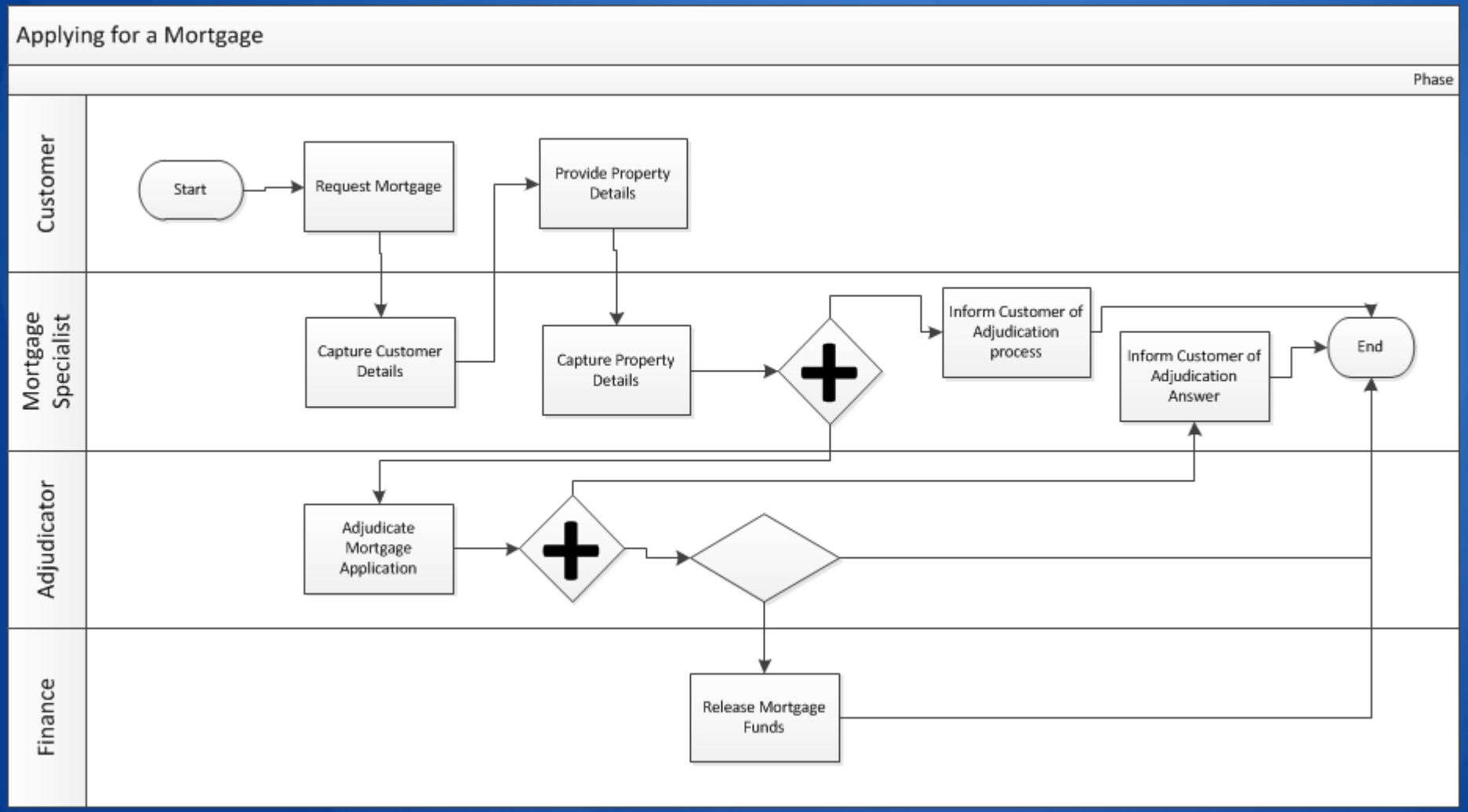
- Standard UML techniques:
  - Context Diagrams
  - Activity Diagrams - can add swim lanes
  - Interaction diagrams – use Swim lanes
  - Collaboration diagrams
- BPMN process diagrams

UML and BPMN require significant material to cover properly and this material is only a note.

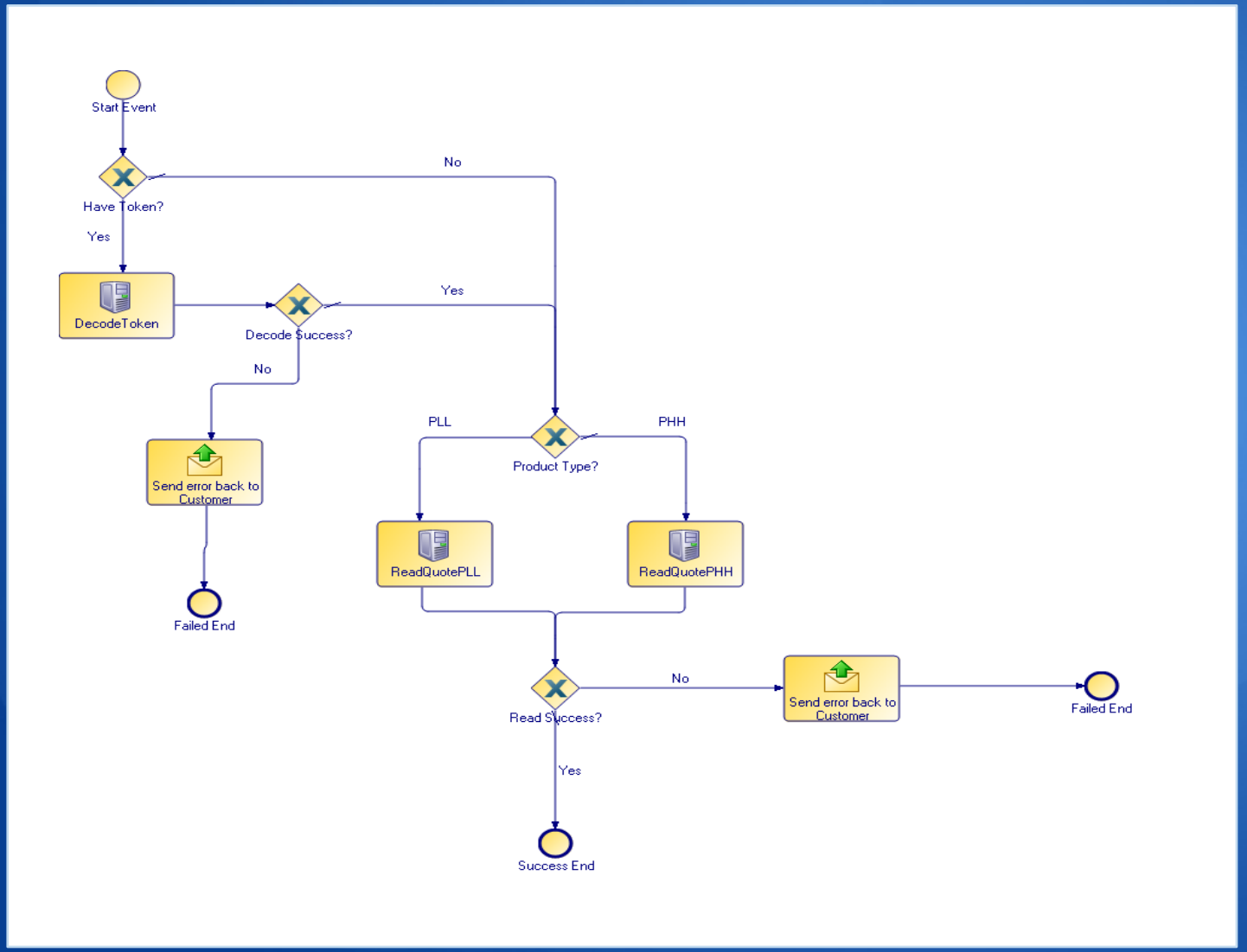
Activity Diagram



BPMN Process Flow



BPMN example – single actor flow



BPMN V's UML

- BPMN is the richest language for communicating processes:
  - Only modelling language that has time triggers, delays and messages.
- BPMN is aimed at modelling processes, but UML provides a variety of visual models with different purposes, e.g. collaboration, state, sequence.
- Both have swim lanes but BPMN has idea of a Pool; a collection of swim lanes.

Two Layers of Use Cases

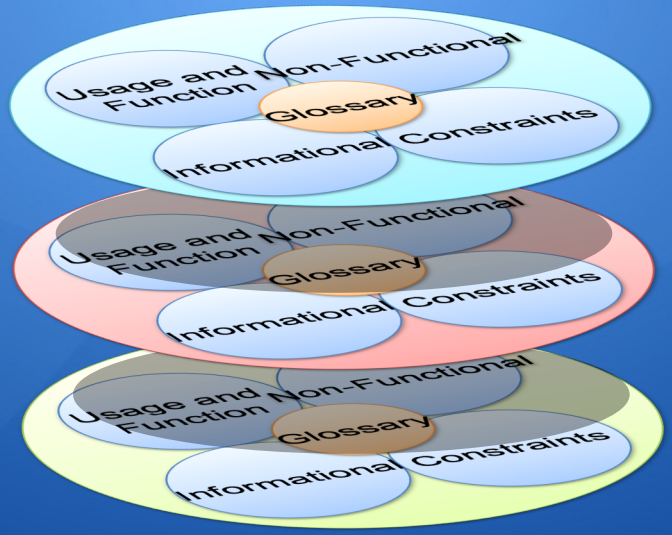
Activity Diagrams can be used to structure the use cases for a system and provide a single high level overview.

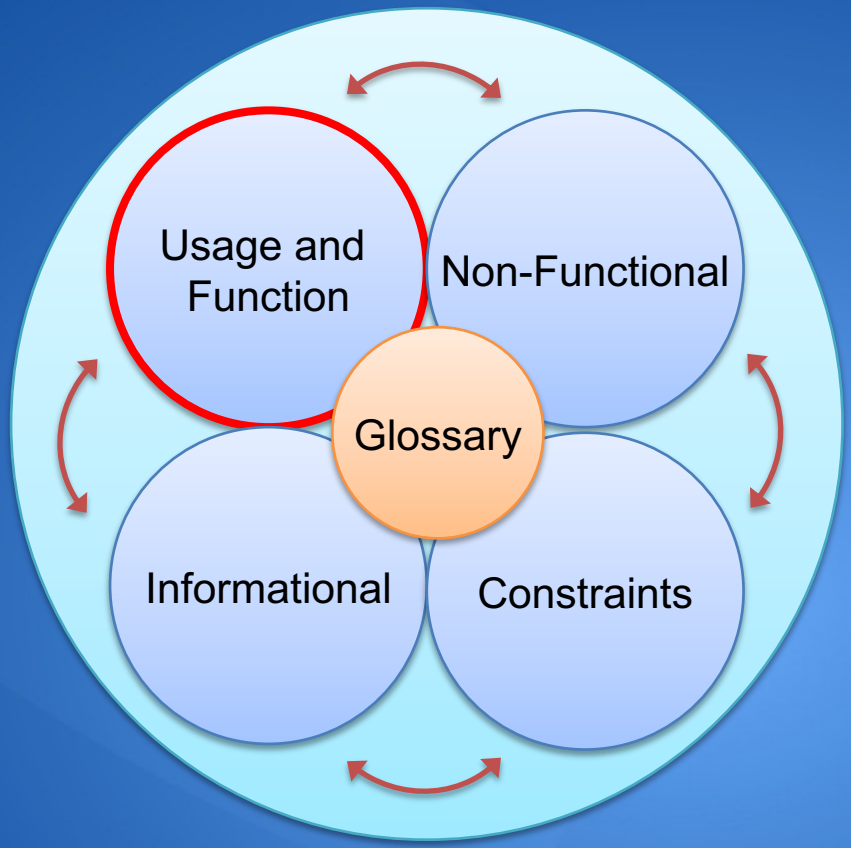
- Business level: The business process surrounding the system can be described end to end with activity diagrams;
- The items of activity are the lower level use cases (Solution and System)
- Highlights the recursive nature of process and use cases.

Business Level

Solution Level

System Level




Usage and Function
Non-Functional
Glossary
Informational
Constraints

Slices

Use Cases and Stories

- Use Cases can be viewed as:
  - generalised collections of stories
  - relating to categories of users not individuals
- A use case is a set of stories relating to two main subjects:
  - Users points of view (left side)
  - Systems points of view (right side)

Use Case Slice and Story

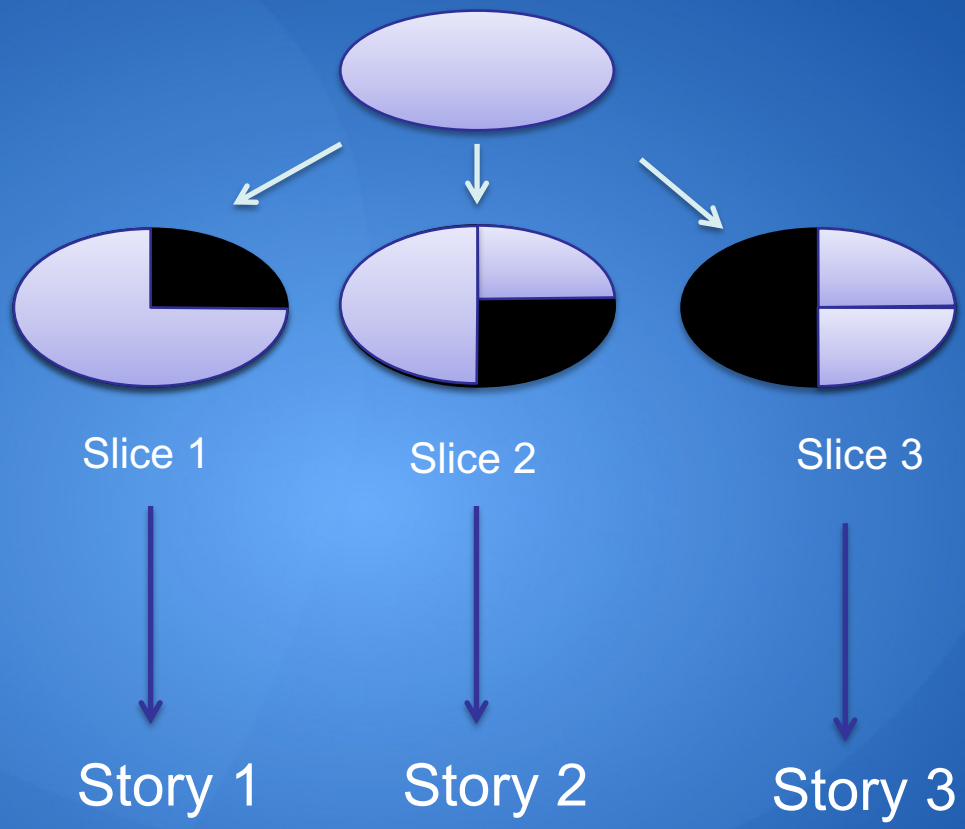
- A slice takes the story from its identification through its realisation, implementation and test allowing the story to be executed.
- A use-case slice is all the work that goes together with a particular story.
- Each story and thus its slice is designed to be a proper backlog element, and realized within an iteration or a sprint.

Slicing a Use Case

Slice by:

- Actors, roles, personas
- Each scenario
- Collections of actions
- Special requirements
- NFRs (e.g. scale from 10 users to 100)
- Etc

- Each slice forms a communicable story

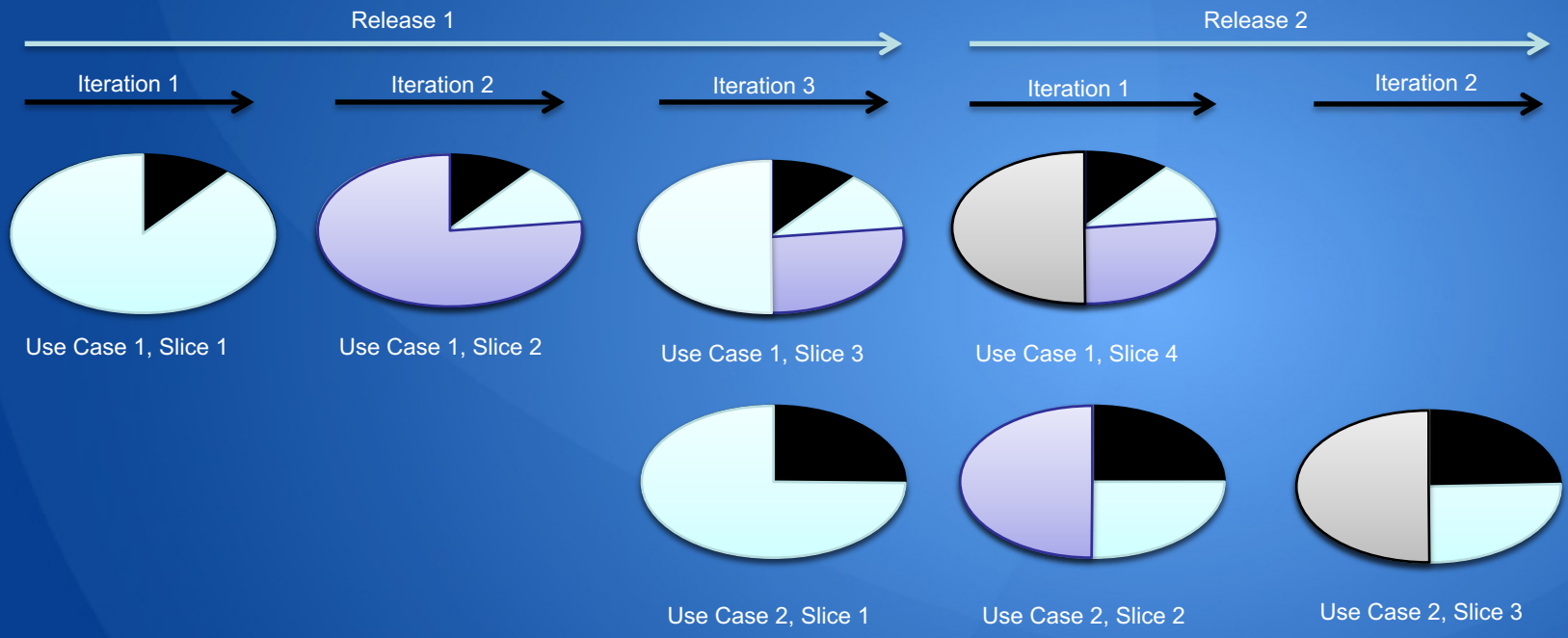


Use Case Slices

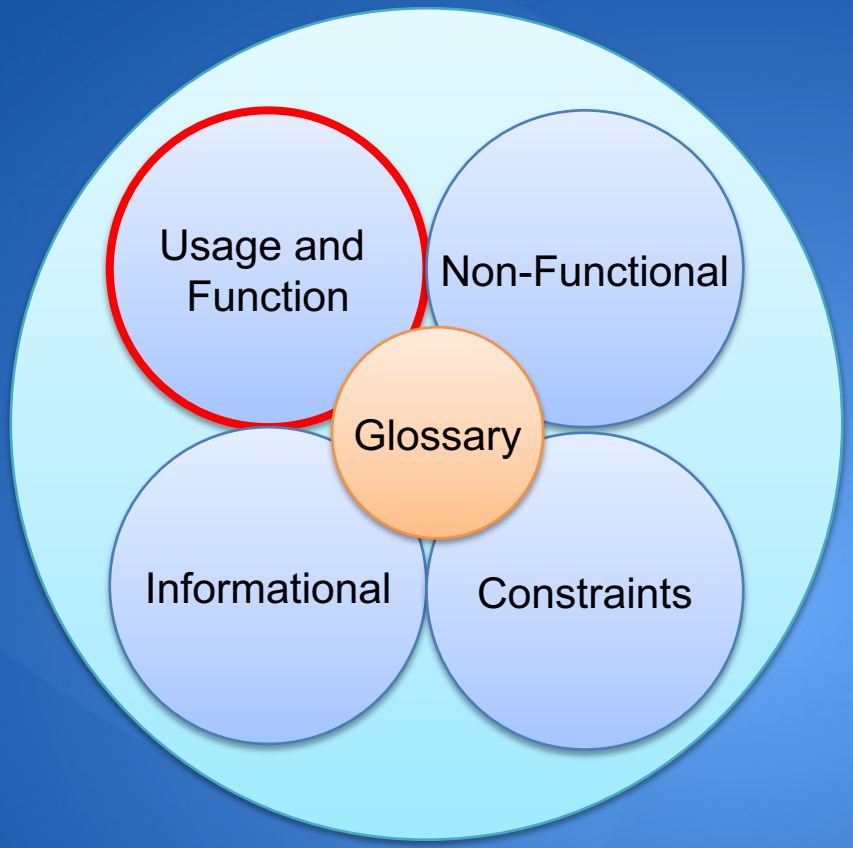
- A Slice includes:
  - a story or use case
  - its analysis counterpart,
  - its design,
  - its code
  - and its test.
- The use-case strategy (starting from a use-case model) makes it significantly easier, compared to the traditional user story strategy, to identify the right user stories to work with and to understand how the ones selected make up the whole system.

Iterative Delivery of Use cases

Use Cases can be delivered continuously or across Iterations and Releases



Each slice is managed across iterations



User Scenarios and Journeys

Personas

A persona is a representation of a particular audience segment for a design, system (website / product) or service. It captures their motivations, frustrations and the “essence” of who they are.

Personas are based on real users. E.g. manager, receptionist, mail clerk, and a visitor. Each persona is named and their background detailed.

They help you understand who will actually be using your website, service or product and therefore can be used to make key design and functionality decisions during the UX process.

Personas are handy to pull out when you are trying to communicate what the user experience should be like or is like for an existing system.

Personas

Detail Includes:

- A person's goals
- A person's motivations for using the system or product.
- A person's current pain points or frustrations
- Relevant demographic data
- A quote that captures their attitude in general, or towards the system or product
- A short bio about their background
- A person's technical ability along with which devices they use and how often
- Other brands or websites they may like
- A picture

User Journeys

User journeys are developed to more closely consider not only the people who use your system or product but the exact steps or interactions they take to complete a particular task.

They can be used to demonstrate the way personas (users) :

- Currently interact with the system or product
- Could interact with it (to be system or product)

Detail usually includes:

- A title summarizing the journey.
- A series of steps in short, concise text. Per step:
  - The device and channel used
  - Changes to the current journey (if future state)
  - Benefits to the user and/or business
  - Any functionality being demonstrated
- An illustration of what's happening in the step (Journey Map)

‘User Scenarios’ – not ‘use case’ scenarios

User scenarios describe in detail what a user (a Persona) does on a website, business, product or service and specifically *why* they do it. *

Are a short story or narrative of a specific person with a certain motivation and a specific goal in mind; a thought experiment used to help you explore the solution or design.

A good user scenario includes all information that is relevant to the process the user undergoes in order to reach their goal, and nothing more.

- Who is the user I’m designing for ?
- What does this user want in this context, hat is their goal ?
- How does this user achieve their goals?
- Why does this user come to my site and not elsewhere?

* They do not cover what the system does

Why use User Scenarios ?

- Provide a vehicle for communication as well as a mechanism to explore design solutions;
- Help mediate the thinking and communication required in design;
- Concrete yet flexible enough to change and morph in detail as the project progresses;
- Help us understand the flow of experience and are tools for thinking about design (they help us reflect and reason);
- Present and situate solutions (‘real’ context with ‘real’ personas – although made up);
- Identify potential problems;
- Easy to understand by all stakeholders as they are story-like;
- Can provide rich descriptions of use in context which can spark ideas and help inform design;
- Determine if the design or solution is appropriate;
- Raise social factors;
- Describe a users multi-channel experiences;
- Can be viewed as test cases for use cases;
- Help flesh out user related Non-Functional Requirements.

Differences with Use Cases Scenarios

User Scenarios typically only include the user actions and not the systems responses, and thus do not describe what the system will do; **they are not system or solution requirements.**

Use Cases are are more general description of a problem than Journeys and User Scenarios, that are very specific and detailed.

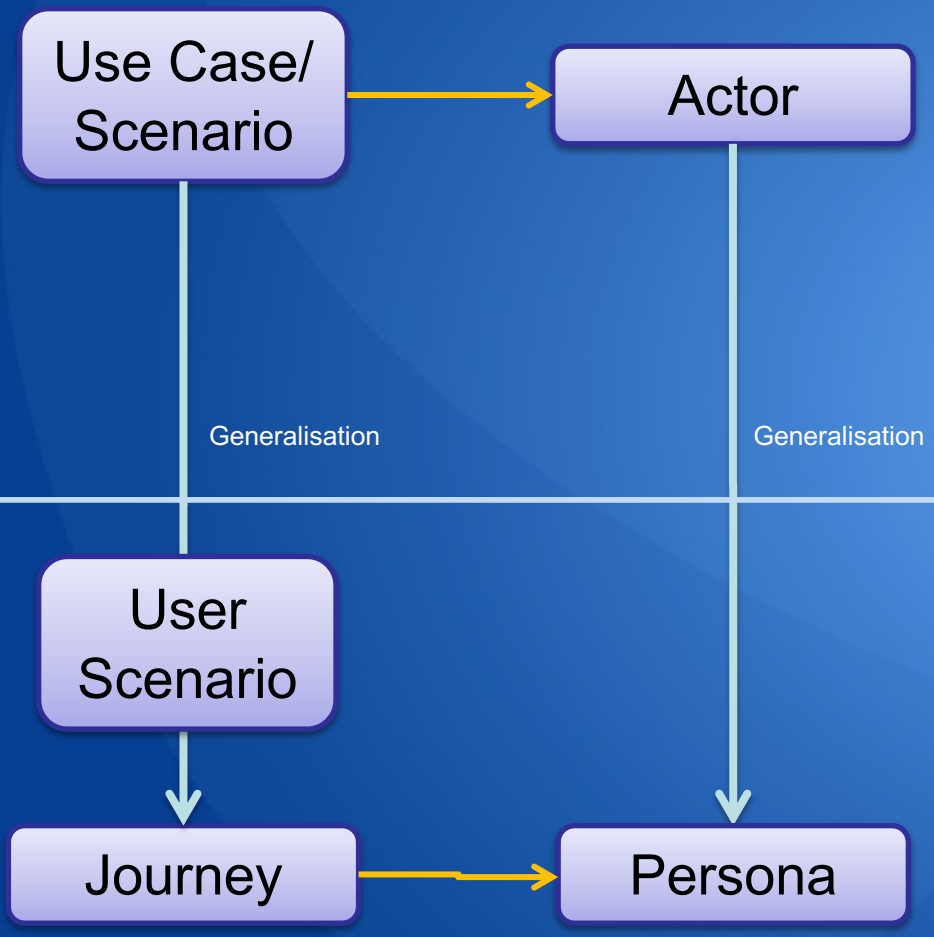
User Scenarios and Journeys describe a single path of flow whereas use cases typically describe several paths – use cases collect and organise all relevant scenarios.

The Actors of a Use Case are a type, whereas with Journeys and User Scenarios they are specific instances, described in detail.

User Scenarios and Journeys are often used to design, check and test the specific solution and mechanism, e.g. a Website form that works on mobile phone.

A use case should not detail the solution, and a common principal is to exclude GUI details, they should work across devices. Use cases focus on the business process and not the specific solution mechanism.

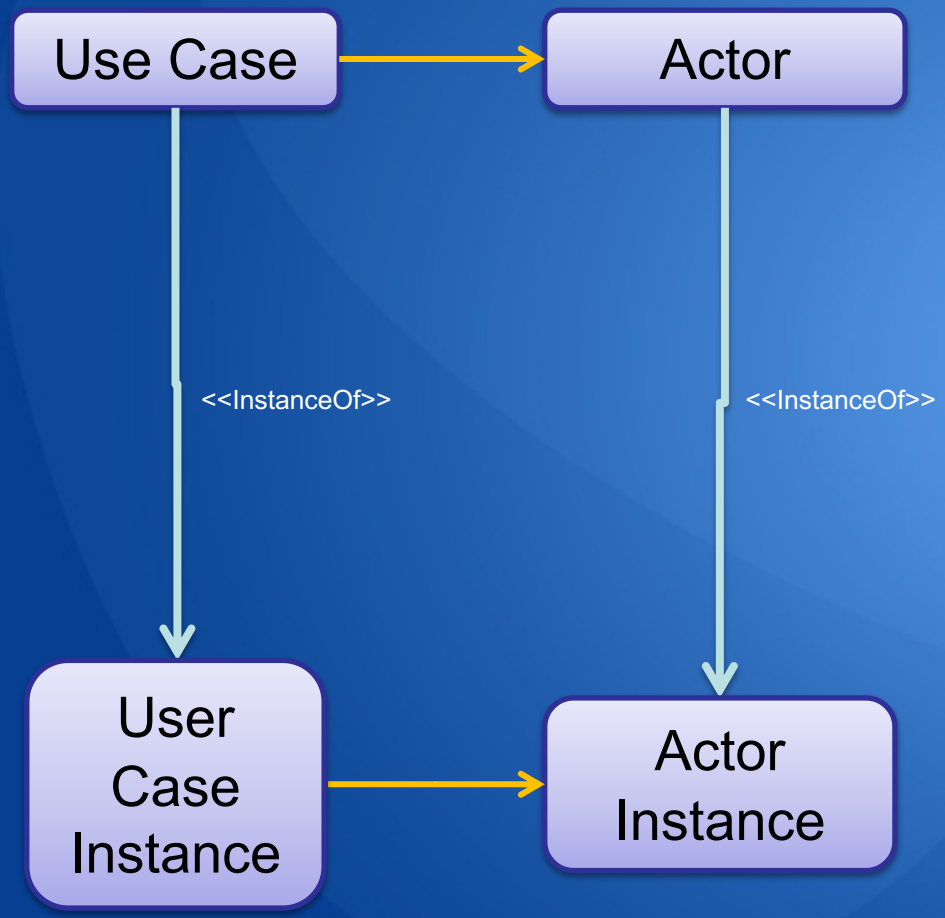
General to Specific - Issues



General: may require new use cases or new use case scenarios; to cover more specific cases – discovered by Journeys or User Scenarios, that introduce new specific variables or flows.

Specific: may be too specific and miss many cases that are covered generally by a use case or introduce irrelevant detail.

Use Case Instance (Alternative)

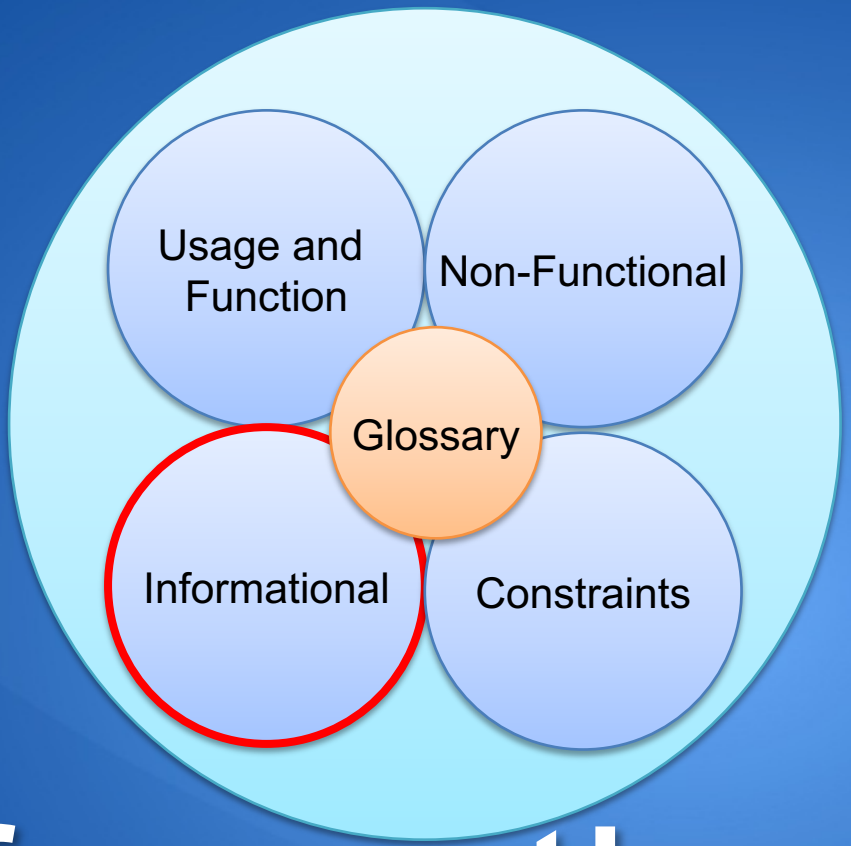


A more Object Oriented alternative approach to **User Scenarios** and **Journeys** is to create specific Instances of Use Cases, actually Use Case Scenarios and Actors.

- Use <<InstanceOf>> relationship
- Actor Instance = Persona
- Use Case Instance = User Scenario & Journey

Uses:

- Draw out and validate rqmts
- Another way to explore detail, and exceptions
- Transition to test cases



Informational Requirements

Information

Two Main Categories

- Entity/Domain models
- Business Rules

Other types of information models:

- Wire frames (UI simulations etc) – not discussed here.

Domain Model

- Describes the interactions and some of the business rules of the *key* entities managed or tracked by a system within a *business* domain
- Describes key entities in the business domain. Details:
  - Identity
  - Structure
  - Relationships between them
  - Rules governing them and their relationships
- Purely conceptual
- Understandable by Requirements Owners
- **Critical for requirements coverage and completeness**

Key Entities

- Pieces of business domain information that the system knows about;
- Key entities are those the system cannot do without;
- Non-key entities are necessary for completeness; and
- Key entities are generally the nouns in the use cases

Finding The Key Entities

- Focus on the actors' needs, what they do, what their goals are and what they expect the system to know;
- Each entity must be of some use to some actor;
- Each entity requires sets of use cases to make them useful to a business process;
- Actors can be entities if they are managed or tracked by the system;
- A state diagram allows a single entity to substitute for what might otherwise seem to be different entities;
- Entity examples:
  - Actor Customer may expect the order management system to know about Shipments
  - Actor Technician may expect the vehicle diagnostic system to know about Vehicle Service History

The Role of a DM

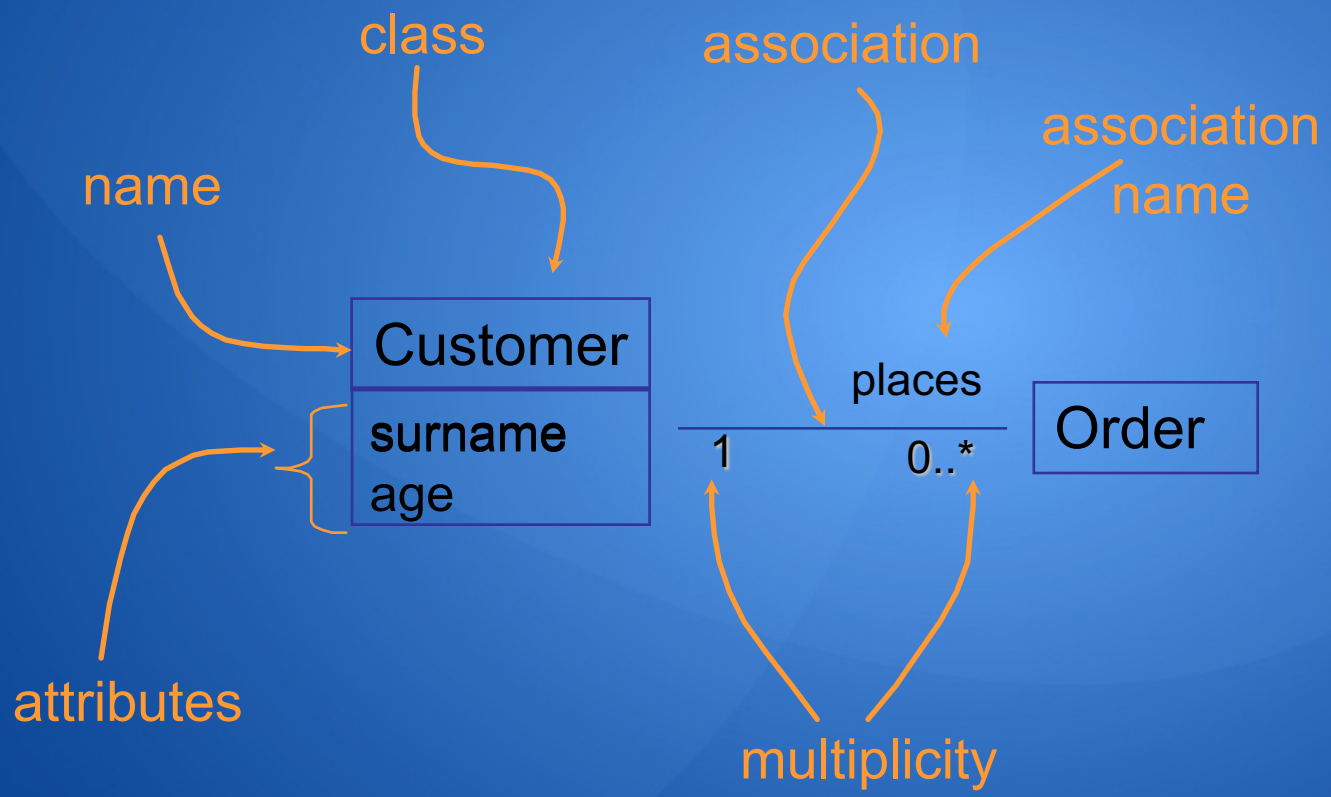
- Information model separate from functional uses (use cases)
- Reinforces functional requirements and validates use cases
- Captures specific kinds of business rules
- Helps identify variables in use cases
  - domain classes
  - attributes
  - associations

Documenting The DM

- Can start out as just a list of the key entities
  - Consists of the names of the key entities and a brief description of each
- Represented using a UML class diagram
  - Independent of implementation
  - Many of the implementation-oriented constructs do not apply (no methods, not all attribute types)

DM – Class Diagram

- Group large, complex diagrams in packages



DM – Attributes & Methods

- Add only logical attributes that are meaningful to the business
- Do not add implementation attributes such as arrays
- Attributes are represented in a business format:
  - Example:"Sponsor Name" and not "sponsorName“
- Generally, attributes that affect relationships or entities are important enough to show explicitly
- Types and default values are added only if they are not obvious and add value such as for a Date
- Methods do not belong in the DM
  - Note: Relationships between entities, which indicate the responsibilities of the entity, may lead to methods in design

DM - Relationships & Multiplicities

- Relationships significant to the business should be shown
- Relationships that explain how the system is designed should be avoided
- For every relationship, show:
  - Relationship name or entity roles if useful
  - Multiplicity
  - Type (inheritance, aggregation, association class, etc.)

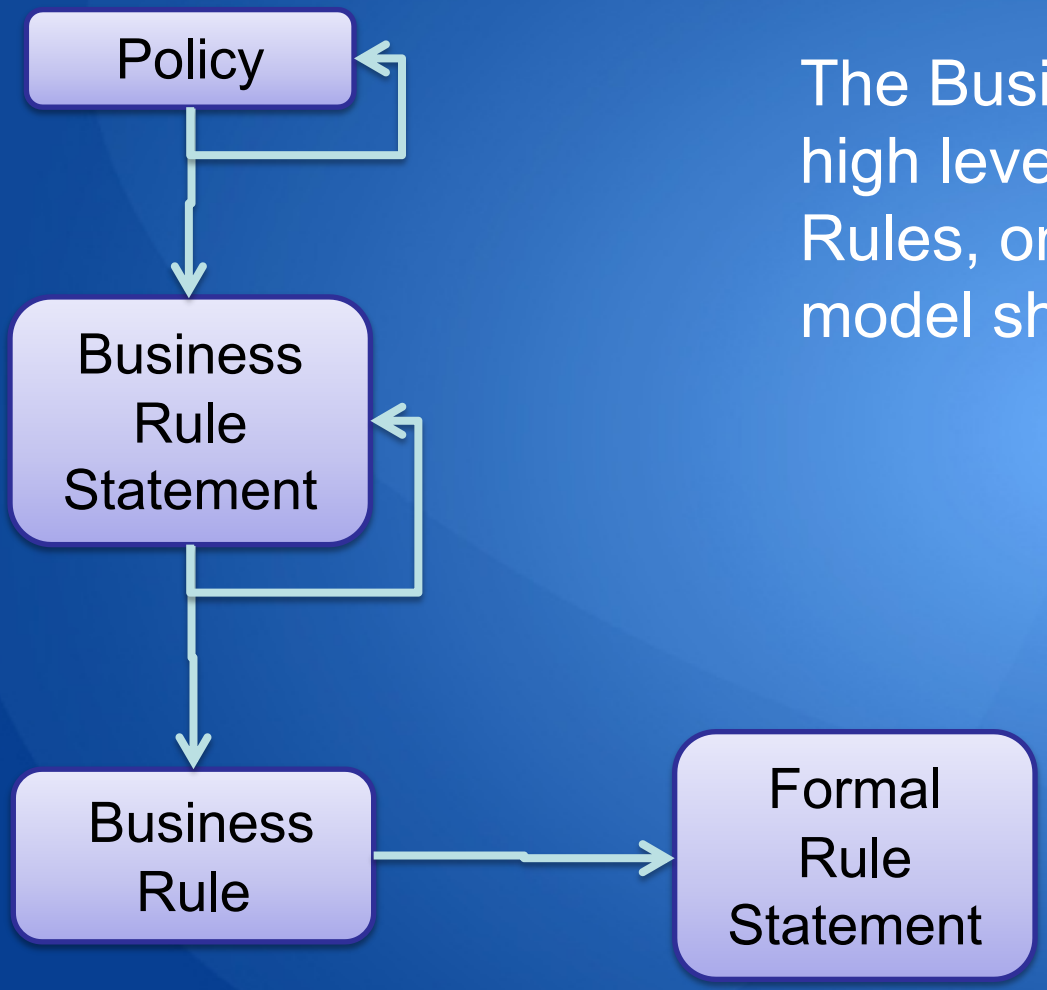
Business Rules

The Business Rules Group defines Business Rules as:

"a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business."

- Business rules are by definition atomic and only involve Boolean logic (true/false). They are distinct to other kinds of rules, e.g. inference rules, do not involve many-valued logics (e.g. true/false/unknown), uncertainty, fuzzy logic or conflict resolution.
- Business rules may be used to manage choice in a workflow or business process.

Business Rule Model (Partial)



The Business Rules Group high level model of Business Rules, only small part of model shown

Rules and Policy

- POLICY is a general statement of direction for an enterprise. Each POLICY may be *composed of* more detailed POLICIES.
- A BUSINESS RULE STATEMENT is a declarative statement of structure or constraint which the business places upon itself or has placed upon it.
  - May be based on POLICY
  - Each may be *related to* one or more other BUSINESS RULE STATEMENTS.
- A BUSINESS RULE STATEMENT, in turn, may be the *source of* one or more ATOMIC BUSINESS RULES.
- Unlike a BUSINESS RULE STATEMENT a BUSINESS RULE cannot be further decomposed.
- A FORMAL RULE STATEMENT is an expression of a BUSINESS RULE in a specific formal grammar.

Types of Business Rules

3 General Types of Business Rules:

- A STRUCTURAL ASSERTION — a defined concept or a statement of a fact that expresses some aspect of the structure of an enterprise. This encompasses both terms and the facts assembled from these terms.
- An ACTION ASSERTION — a statement of a constraint or condition that limits or controls the actions of the enterprise.
- A DERIVATION — a statement of knowledge that is derived from other knowledge in the business.

STRUCTURAL ASSERTIONS

- A statement that something of importance to the business either exists as a concept of interest or exists in relationship to another thing of interest. It details a specific, static aspect of the business, expressing things known or how known things fit together.
- Frequently portrayed by entity/relationship or domain models.

- A TERM is a word or phrase which has a specific meaning for the business. The two types: BUSINESS or COMMON TERMS.
- May be classified as a TYPE (defining abstract categories of things like ‘car model,’ ‘walk-in rental,’ ‘customer,’ etc.) or as a LITERAL (describing instances of things, such as ‘General Motors,’ or ‘5000’).
- A type of TERM could be a SENSOR that represents the presence of something that detects and reports constantly changing values from the outside world. Another type could be CLOCK which reports the passage of time.

STRUCTURAL ASSERTIONS

- A FACT asserts an association between two or more TERMS. That is, it expresses a relationship between the TERMS. A FACT involves two or more TERMS, and a TERM may be in one or more FACTS.
- Three types of FACT:
  - an ATTRIBUTE of another TERM,
  - a GENERALIZATION (or, supertype) of one or more other TERMS (its subtypes), and
  - a PARTICIPATION (or relationship) between other TERMS
- Three general kinds PARTICIPATION :
  - An AGGREGATION is a ‘part of/composed of’ relationship.
  - A ROLE describes how one TERM serves as an actor (another TERM) through its interactions with its environment.
  - An ASSOCIATION is simply any other kind of relationship.

ACTION ASSERTION

- A statement that concerns some dynamic aspect of the business. It specifies constraints on the results that actions can produce. The constraints are described non-procedurally, in terms of the other atomic business rules.
- Where the STRUCTURAL ASSERTIONS describe possibilities, ACTION ASSERTIONS impose constraints — 'must' (or, 'should') or 'must not' (or, 'should not').
- Examples:
  - "A car must have a registration number.";
  - "A car cannot be handed over to the customer unless a provisional charge has been accepted against the customer's credit card.";
  - An order must have a line item.

DERIVATION

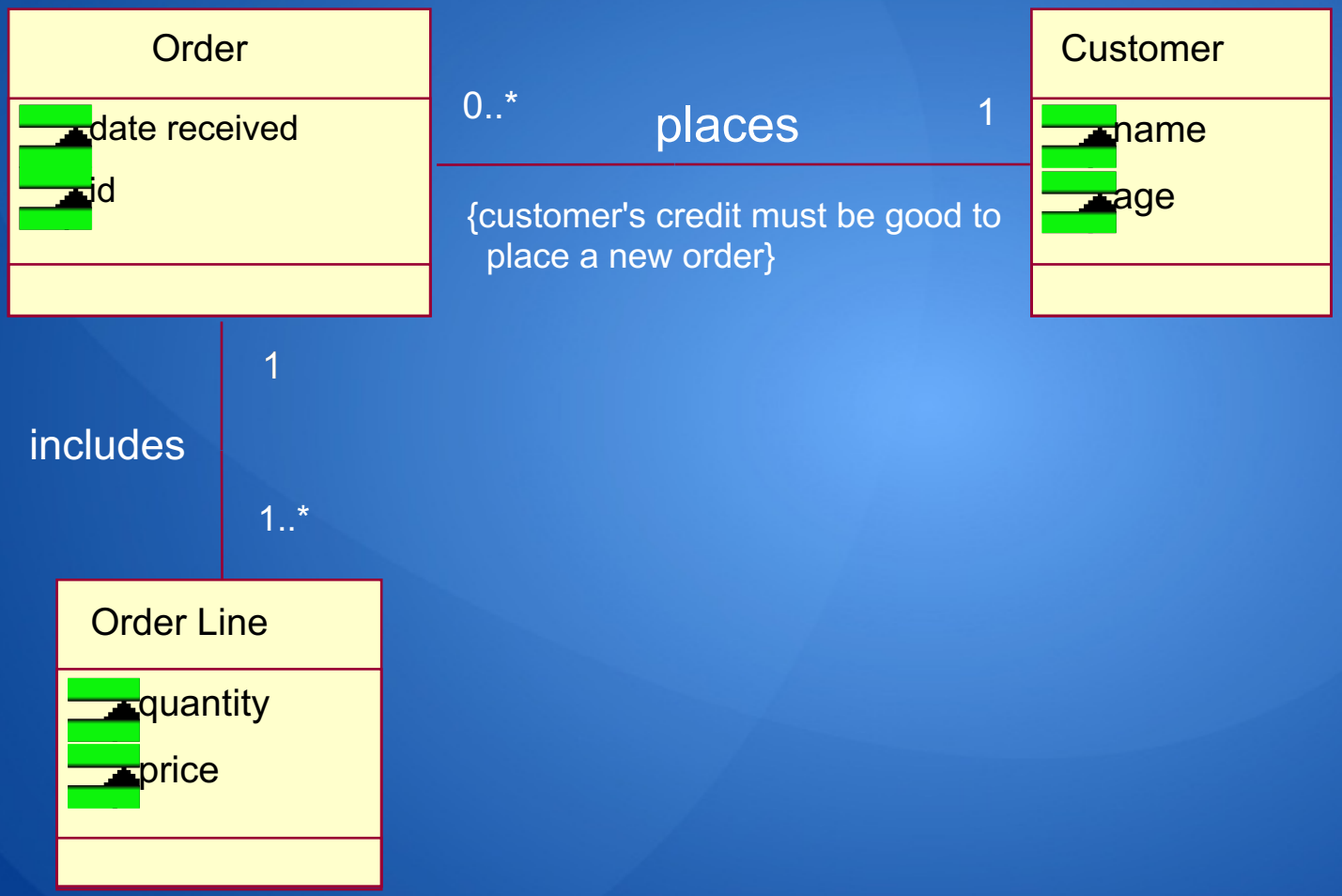
- A BASE FACT is a FACT that is a given in the world and is remembered (stored) in the system.
- A DERIVED FACT is created by an inference or a mathematical calculation from TERMS, FACTS, other DERIVATIONS, or even ACTION ASSERTIONS.
- May be either a MATHEMATICAL CALCULATION or an INFERENCE
- Example:
  - Calculation: The ‘Rental amount’ in RENTAL is ***calculated from*** the ‘Rental rate’ multiplied by the ‘number of days.’”
  - Inference: the TIME SHEET ENTRY’S ‘charge rate’ is in fact the corresponding PERSON’S ‘charge rate.’

DM - Business Rules

Some Business Rules can be captured in the Domain Model:

- Business rules governing relationships may be represented in the class diagram in curly brackets
  - When the number of these business rules becomes too large, they may be captured textually
  - Multiplicities are form of business rule, e.g. an order must have a line item.
- Business rules governing the attributes and the entities are captured textually
- Business rules related to behavior are captured within the use cases
- Complex and detailed business rules may be captured in a separate section of requirements document

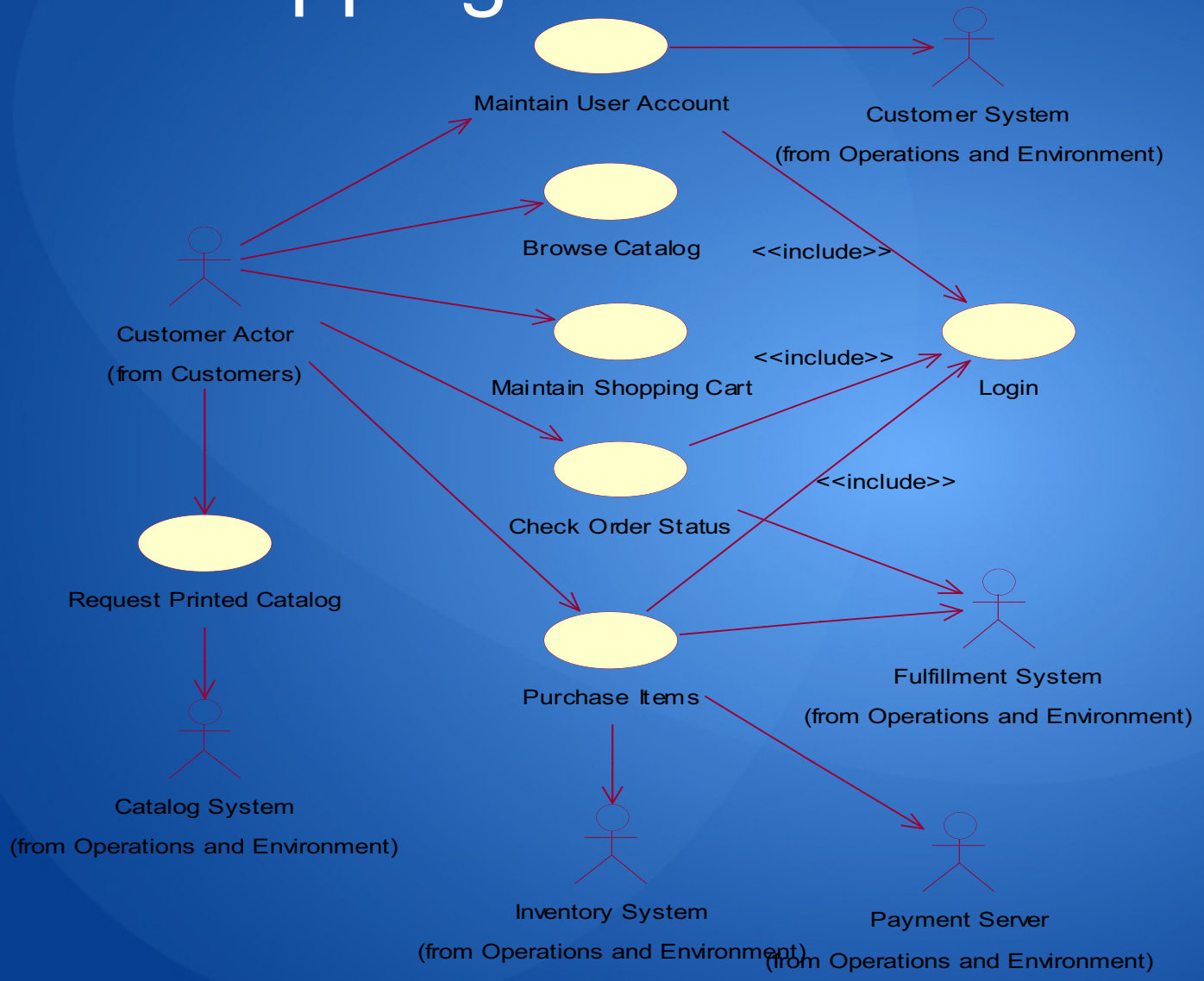
DM - Example


Order
date received
id
0..*
places
1
Customer
name
age
{customer's credit must be good to place a new order}
1
includes
1..*
Order Line
quantity
price

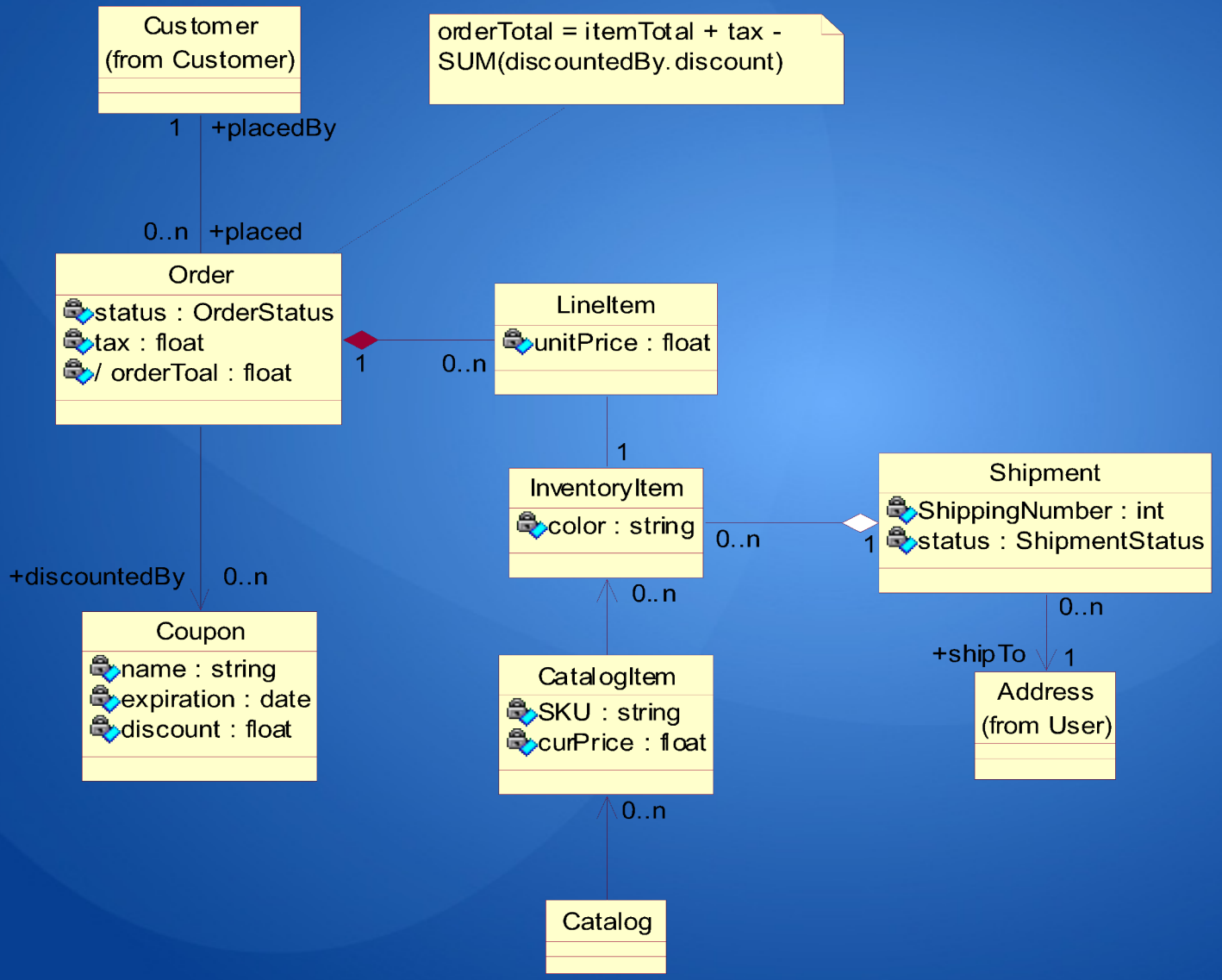
Domain Model vs Design Model

Domain Model	Design Model
Problem	Solution
Business	Technical
For Requirements Owners	For Developers
Conceptual	Specific
Non-behavioral	Behavioral
UML Class Diagram	UML Class Diagram
Non-programming Language Specific	Programming Language Specific

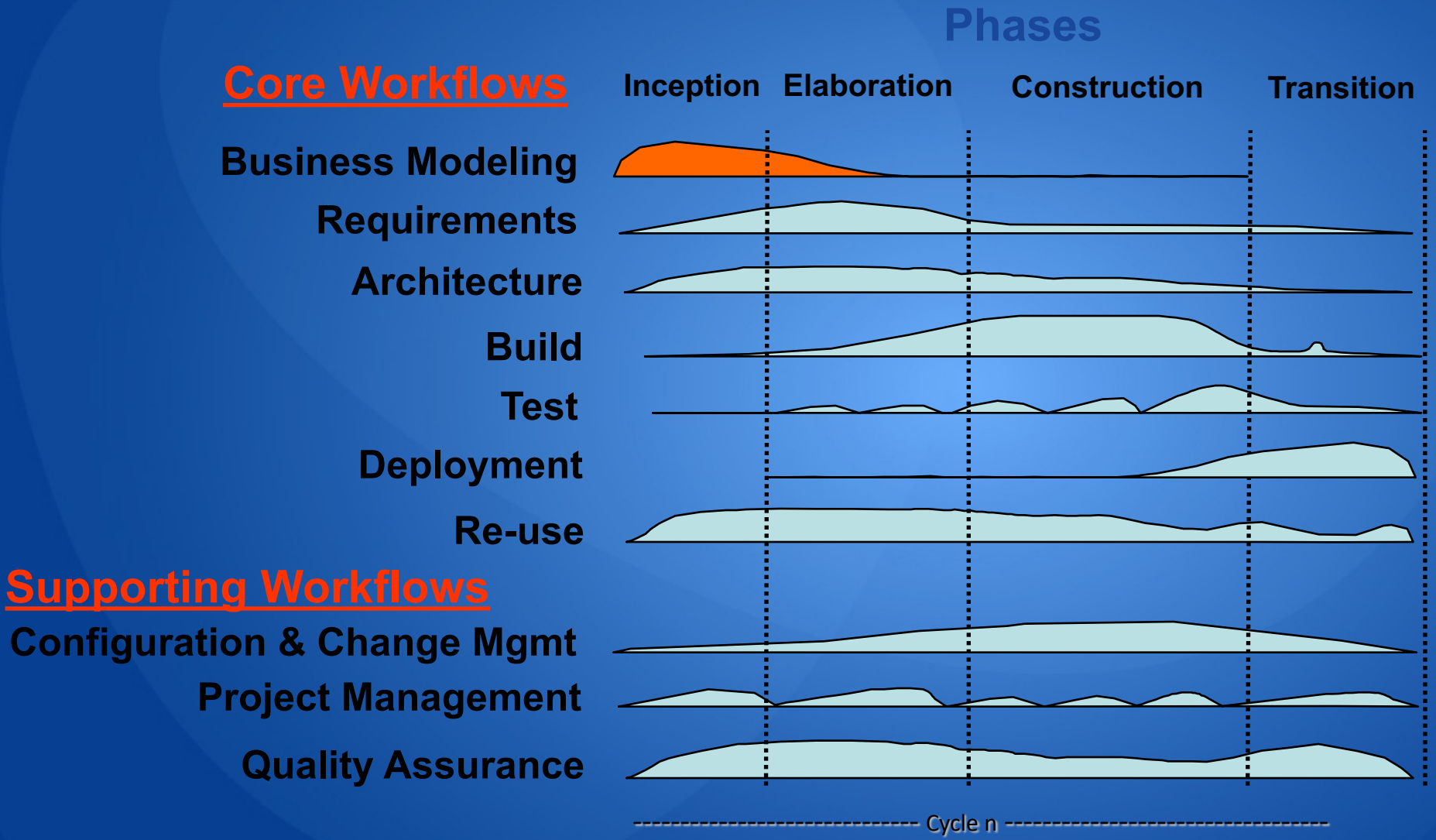
Online Shopping



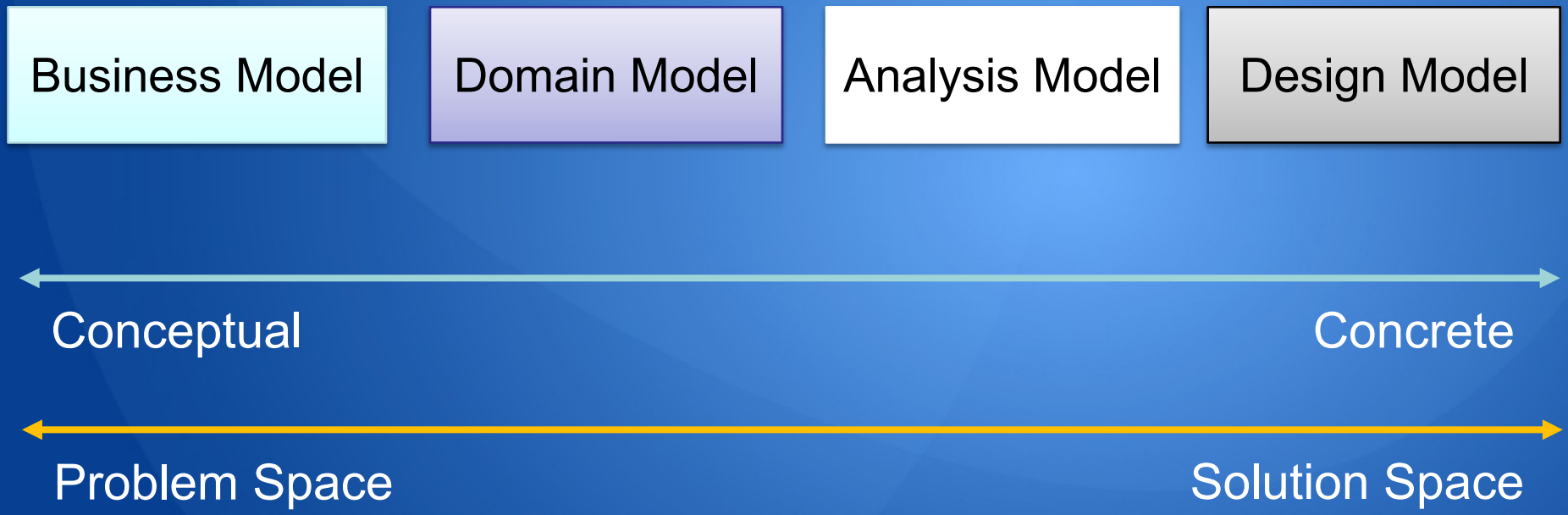
Shopping DOM



Business Modeling Workflows

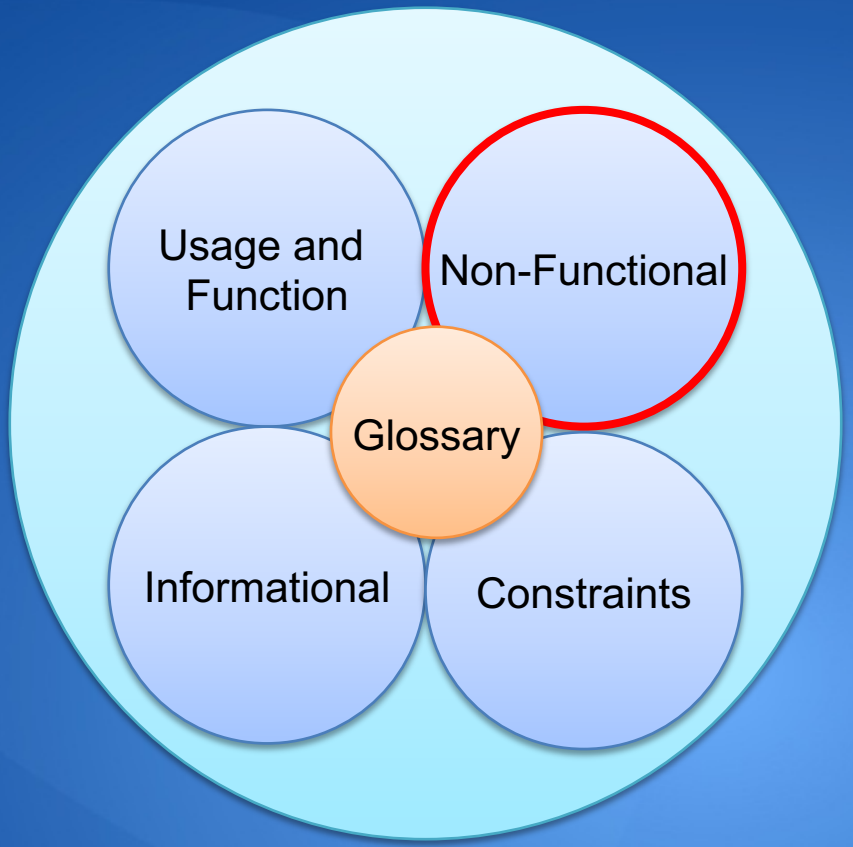


The Models in the Process



What is a Business Model ?

- Business modelling is a process to understand the business processes and structure of an organisation.
- Supports two kinds of UML models:
  - Use Cases (business processes)
  - Object Models (business domain entities)
- These models differ from information system domain models and use cases as now the system is the business
- Employ business specific extensions to UML.
- Provides Starting point to define ‘Services’



Qualities of Service

Non-Functional Requirements

- The International Institute of Business Analysis (IIBA) defines **Functional requirements** as “the product capabilities, or things that a product must do for its users.”
  - Functional requirements define how software behaves to meet user needs.
- The IIBA defines **Non Functional Requirements** (NFRs) as “the quality attributes, design and implementation constraints, and external interfaces which a product must have.”
  - Quality attributes are often affectionately called the “ilities” because the names of many of them end in “ility.”

Often Ambiguity Around Qualities

Functional requirements describe ‘what’ the users want, and value, but not ‘how well’ the functions perform.
Non Functional Requirement, the Systemic Qualities, capture the ‘how well’.

What is an SLR ?

Service Level Requirement (SLR):

- A requirement that specifies desired ‘service’ conditions on a functional requirement at or over a period of time.
- Often a broad statement from a customer to a service provider.
- They are orthogonal to the functional requirements.
- Provide the systemic qualities for a system.

What is a Systemic Quality (SQ)?

- A measurable or observable, and generally broad characteristic of a system.
- Criteria that can be used to judge the operation of a system.
- Broader in scope than a single use case, feature, story, a single stakeholder, or a single point in the lifecycle of the system.
  - Are cross-cutting; Systemic
- Describe a goal to be achieved.

Qualities and Customer Value

- Unlike Functional requirements, Architecture is often communicated as having no (or little) externally visible “customer value” and is most often seen as a cost.
- NFRs become ‘Invisible Features’, and are often forgotten.
- However, visible “customer value” is not enough to deliver a ‘quality’ solution.
- Given only Functional Requirements the result is solutions that do not meet the current and future needs of the organisation, e.g. are not available, too hard to use, do not scale to meet new customer demand, or are unreliable.
- Although not directly impacting the customer, these invisible features, end up impacting the customer indirectly but of more concern they will impact the organisation, and its reputation;
- In hindsight organisations often find them to be the most important requirements of a system.

Understand NFRs Early

"While we must acknowledge emergence in design and system development, a little planning can avoid much waste."

James Coplien.

- Refactoring Architecture, in most cases, requires expensive and time consuming re-builds. It is often too big a problem for refactoring.
- Therefore, some upfront planning is required, and this needs to be driven by good quality NFRs.
- Tackling these early on, can reduce development and operational costs, and also provide an accurate cost of a quality architecture.
- To do this, the business impact of NFRs needs to be understood by Stakeholders, so they appreciate their 'value', and that they are driven by (traceable to) business needs.

4 Main Types of Qualities:

- **Manifest** qualities support end user needs. In most cases they are objectively measurable and have impact at specific release points.
- **Operational** qualities impact those stakeholders that run, operate, monitor or manage the system.
- **Developmental** qualities describe desirable qualities of the system from the standpoint of those who will build it.
- **Evolutionary** qualities describe future goals of the system.

4 Types of Systemic Qualities

Type Of Quality	Who	When	How
Manifest	End user and business stakeholders	After initial release	Test, customer satisfaction, business objectives
Operational	Operations staff	Development operations, release-time, after release	Test, customer satisfaction
Developmental	Developers, PMs, etc.	During and throughout development	Ongoing productivity metrics
Evolutionary	Developers, business stakeholders	After initial release	As above

Specific Qualities by Type

Manifest	Performance
	Availability
	Usability
	Reliability
	Accessibility
	Mobility
Developmental	Buildability
	Testability
	Understandability
	Code Quality Measurability
	Conceptual Integrity
	Budgetability
	Planability
	Traceability
	Development Distributability
	Modularity / Packagability

Operational	Throughput
	Security
	Manageability
	Maintainability
	Serviceability
	Deployability
	Reproducibility
Evolutionary	Scalability / Elasticity
	Variability
	Flexibility
	Extensibility
	Reusability
	Portability
	Interoperability

The Jargon Word “interoperability”…..

Alternatives:

Compatibility – the ability of systems to cooperate meaningfully for some purpose and/or to coexist without interfering with one another. Compatible systems work together as-is, not requiring any integration effort beyond some trivial configuration to let them know that they are supposed to work together.

Integratability – the quality of possibly *incompatible* systems that makes it easier to adapt them or their exchanged data so that they will cooperate meaningfully for some purpose. Systems that are integratable can be made compatible with some nontrivial amount of effort, for example, by writing "glue code."

Composability – the quality of a collection of components that enables them to cooperate meaningfully for the purpose of assembling larger systems.

Substitutability – the quality of a group of things that makes it possible to replace one with another without adverse effects on things that depend on them.

Portability – the quality of a component that allows it to be substituted.

Availability SLR Questions

- What are the expected hours of operation?
  Does it vary by use case?
- Can the system become unavailable for scheduled maintenance?
  Any restrictions on the hours?
- What is the acceptable downtime on a failure?
- Are there times when downtime is more costly than others;
- Examples
  - Availability is required between 6 AM and midnight Mon-Sat. A 6 hour maintenance window exists between midnight and 6 AM and on Sundays.
  - If the connection is down to mainframe the User Interface should still function and allow changes to be submitted later.

Availability & Reliability Aspects

- Reliability:
  - Measure: Mean Time Between Failures (MTBF) or number of failures per week;
  - need to define a ‘failure’
- Recoverability:
  - Recovery from failure
  - Measure: – Mean Time To Repair: (MTTR)
  - Tactics: checkpoint restarts; auto-saves, etc.

Availability = (1 - MTTR/(MTBF + MTTR)) * 100%

Percentage Uptime	Percentage Downtime	Downtime Per Year	Downtime Per Week
98%	2%	7.3 days	3 hours, 22 minutes
99%	1%	3.65 days	1 hour, 41 minutes
99.8%	0.2%	17 hours, 30 minutes	20 minutes, 10 seconds
99.9%	0.1%	8 hours, 45 minutes	10 minutes, 5 seconds
99.95%	0.05%	4 hours, 23 minutes	5 minutes
99.99%	0.01%	52.5 minutes	1 minute
99.999%	0.001%	5.25 minutes	6 seconds
100.00%	0.0001%	31.5 seconds	0.6 seconds

Scalability / Elasticity SLR Questions

- What is number of concurrent users at deployment?
  - In 6 months? In 12 months? In 24 months?
- What is the total user base at deployment?
  - In 6 months? In 12 months? In 24 months?
- Does the system need to scale automatically or is this a manual operations process ?
- Examples
  - The proposed system must support up to 40 External Customers and 100 Internal simultaneously.
  - The proposed system must be able to handle double the approximately connections per day to support deployment to 8 more DC's in 6 months.

Performance SLR Questions

- What is the average desired response time or **latency** ?
  - Does this vary by use case?
  - Does this vary by timeframe?
- What is the maximum allowable response time? Does this vary by use case?
- How many user requests should the system handle at one time ?
- Examples
  - Item scans should not take more than 2 seconds during period of normal activity (during the hours of 8am and 6pm Monday to Friday).
  - The load profile is as follows:
    - 8000 concurrent users, 25,000 sessions/hour, 100 dynamic pages/sec.
    - Under this load, the system shall operate within these performance parameters for operational requests:
      - 99% of requests within 5 second response time
      - < 2 second average response time
      - 5 hours sustainable load

Usability

Usability is concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides. It can be broken down into the following areas:

- **Learning system features.** If the user is unfamiliar with a particular system or a particular aspect of it, what can the system do to make the task of learning easier?
- **Using a system efficiently.** What can the system do to make the user more efficient in its operation?
- **Minimizing the impact of errors.** What can the system do so that a user error has minimal impact?
- **Adapting the system to user needs.** How can the user (or the system itself) adapt to make the user's task easier? For Example personalisation.
- **Increasing confidence and satisfaction.** What does the system do to give the user confidence that the correct action is being taken?

Performance – Good/Bad Example

Bad:

"Response times will average 5 seconds or less"

Provide a usage profile and time characterization to make the goal more specific and meaningful.

Good:

"Response times for document searches will take no longer than 5 seconds between the hours of 8am and 11pm on a weekday"

What about Agility ?

Agility in software is an aggregation of multiple qualities:

- understandability – includes (debug-ability)
- extensibility,
- flexibility,
- portability,
- scalability,
- security,
- mobility,
- testability

However, hard to know what specifically the emphasis in on, what should be measured and tested.

Note this is different to using an ‘Agile’ delivery process. An Agile delivery process may deliver a solution that is not agile, and a non-Agile waterfall process could produce a solution that is Agile.

Systemic Qualities Interact

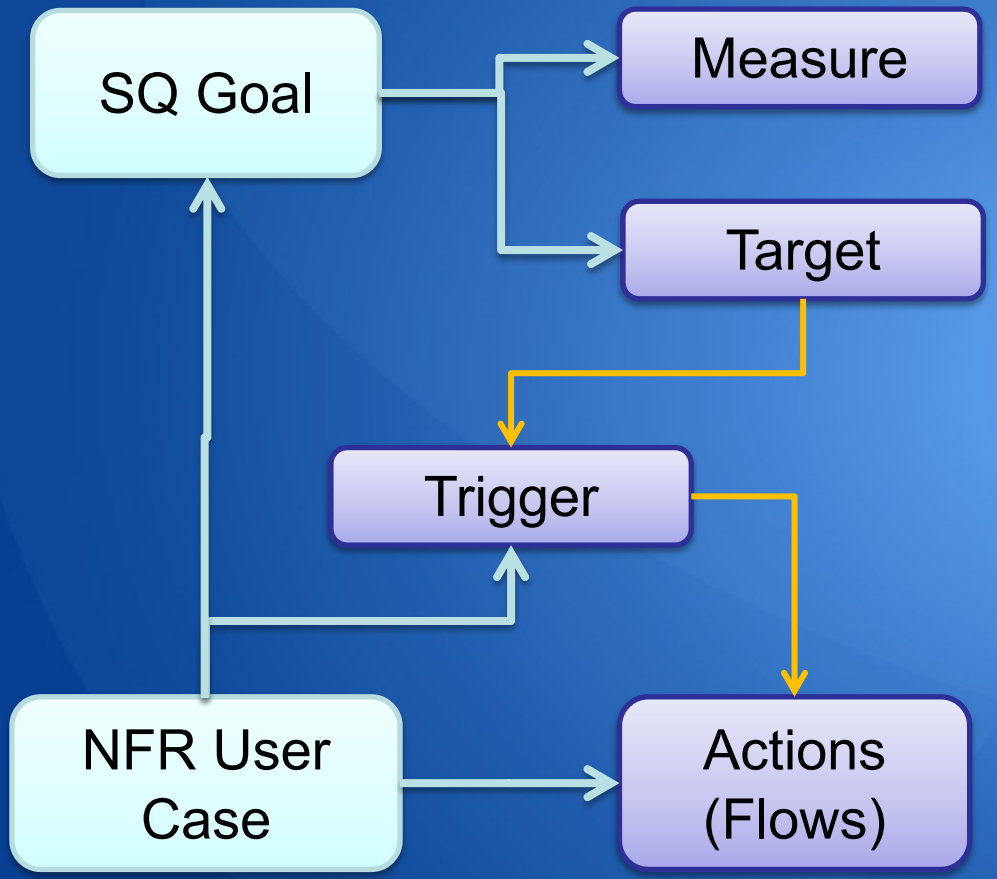
These (across) impact those below	Scalability	Serviceability	Availability	Manageability	Security
Scalability		+	-/+	+	-
Serviceability	+/-		-/+	+	-
Availability	+/-	+		+	-
Manageability	+	+	+/-		-/+
Security	+	+/-	-	+	

Generic NFRs are Not Enough !

Generic statements about Service Levels or Systemic Qualities only capture the goal but do not describe how they are achieved by the organisation; what is required.

- Activities, Actions, Processes and workflows need to be implemented to achieve these goals;
  - Often called ‘operational’ activities.
  - These could be manual or automated.
  - Involve different actors to those specifying the goal, e.g. operators and maintainers Vs business stakeholder.
- Use cases and scenarios can be used to describe these requirements.
- Describing these use cases allows the impact of NFRs to be understood, assessed, costed and traded-off.

NFR Use Cases have associated SQs

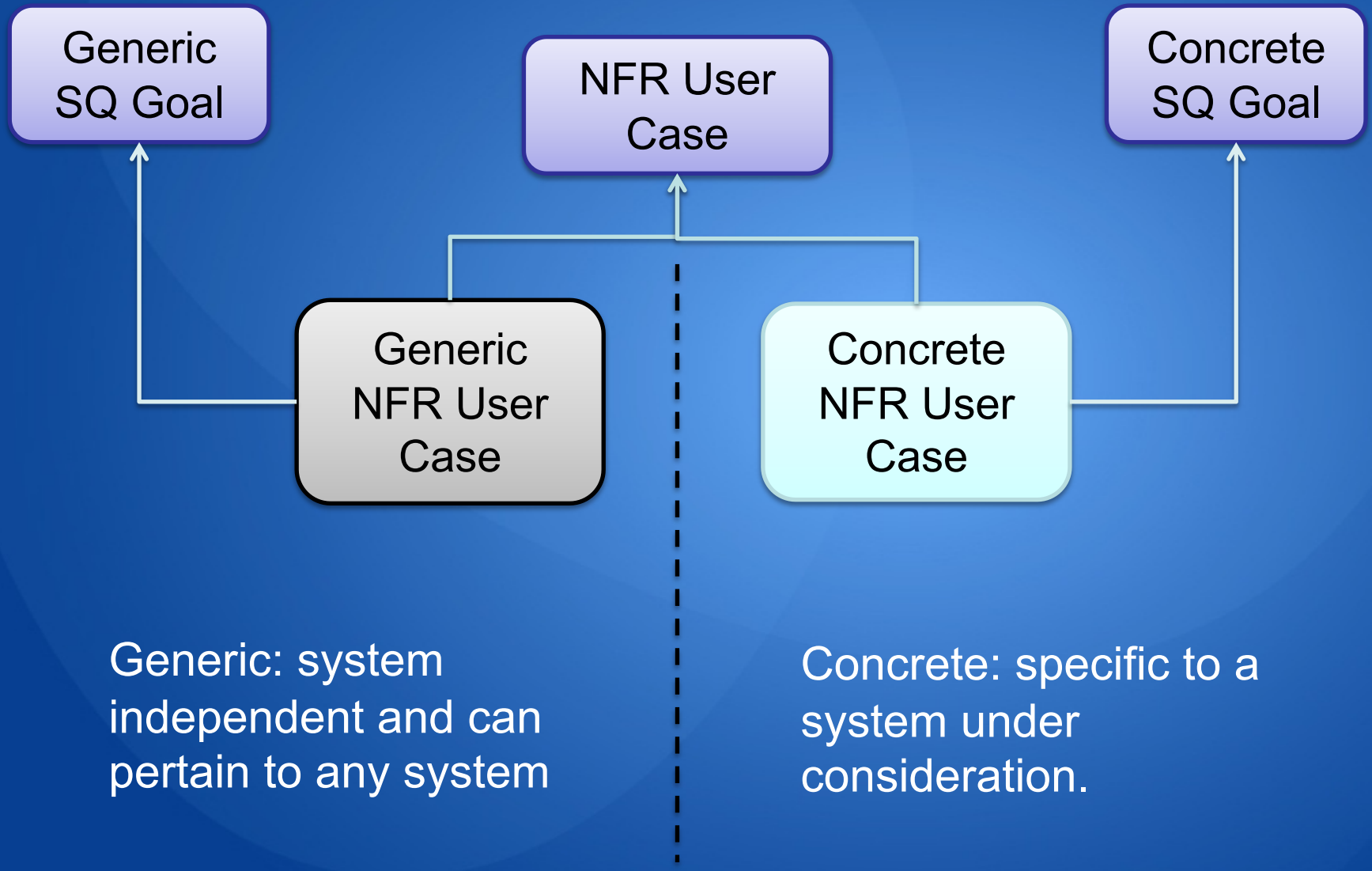


An SQs measure may not meet target. This will trigger the actions of a use case to brings the quality, its measure, back in line.

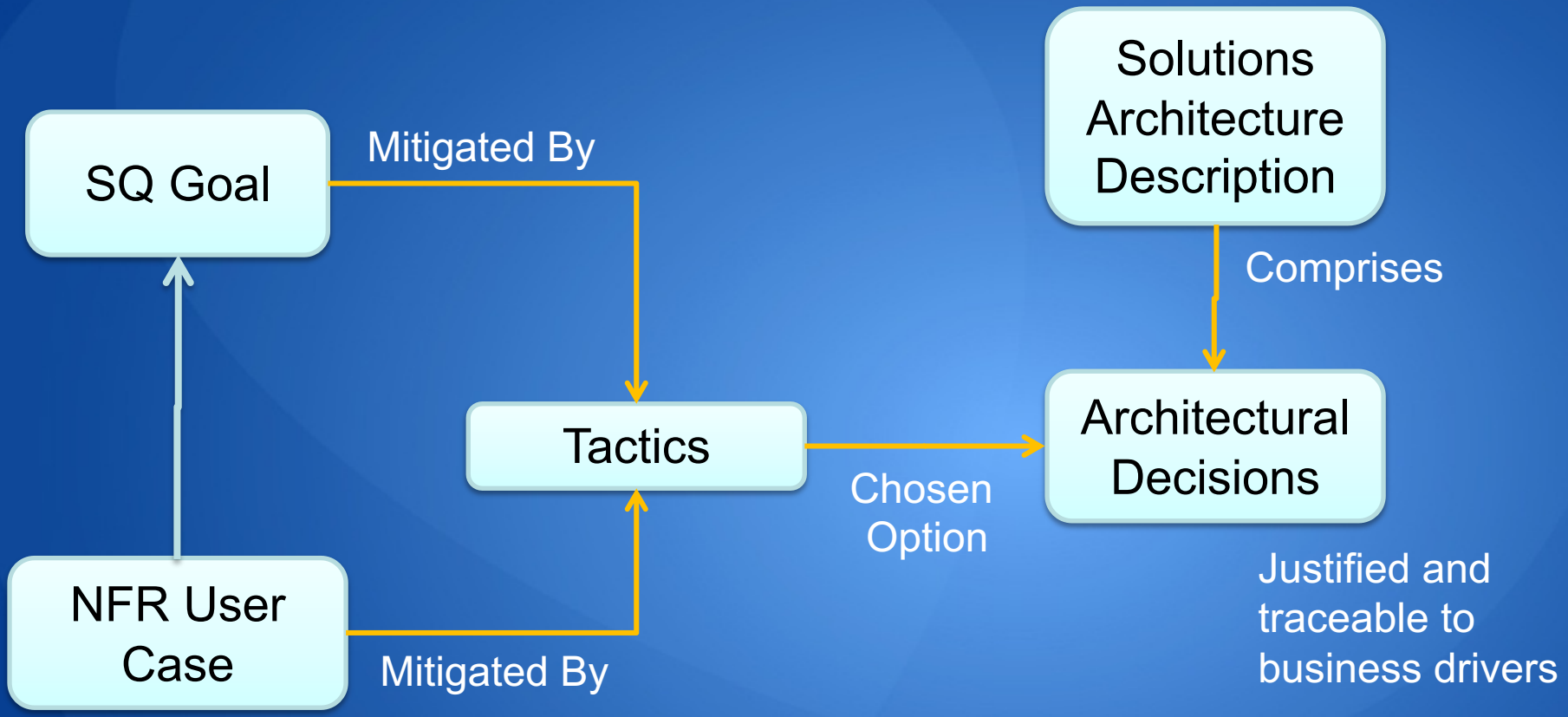
Has

Invokes

General or Concrete NFR Use Cases



SQs drive Architecture



SQs provide the goals Architectural Decisions solve. Viewing Architecture as a set of decisions makes evolution an inherent part of the architecture description.

Balance and Trade Off

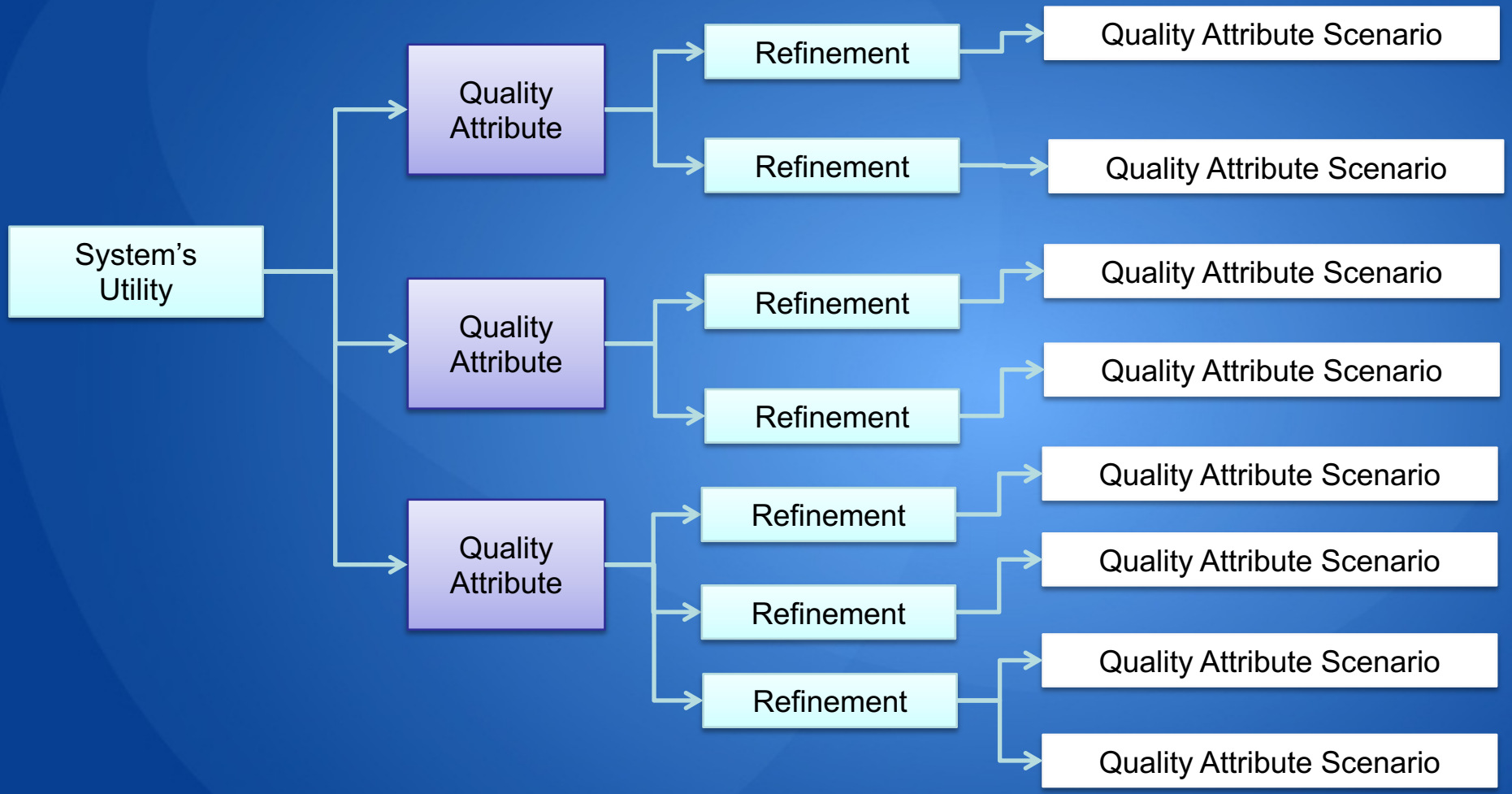
- Recalling INVEST; User Stories should be independent. However this is not true for NFRs as they are pervasive, cross-cutting and interact.
- They usually cannot be delivered in an iteration as they impact the system as a whole across iterations. Mostly impact the future of the system from day one onward.
- The fact that qualities interact means that ***trade off and balance*** is required, and a systematic 'method' is required to analyse these trade offs record and manage the process.
- BaaS, Clements and Kazman claim that system quality stems from architectural quality and describe an approach:
  - Qualities can be organised in a 'utility' tree structure, the nodes describe Quality Attributes;
  - The attributes can be decomposed or refined;
  - At the leaves of the tree, scenarios are employed as the atomic elements to describe the detail that are concrete enough for prioritisation and analysis.

Definitions from IEEE

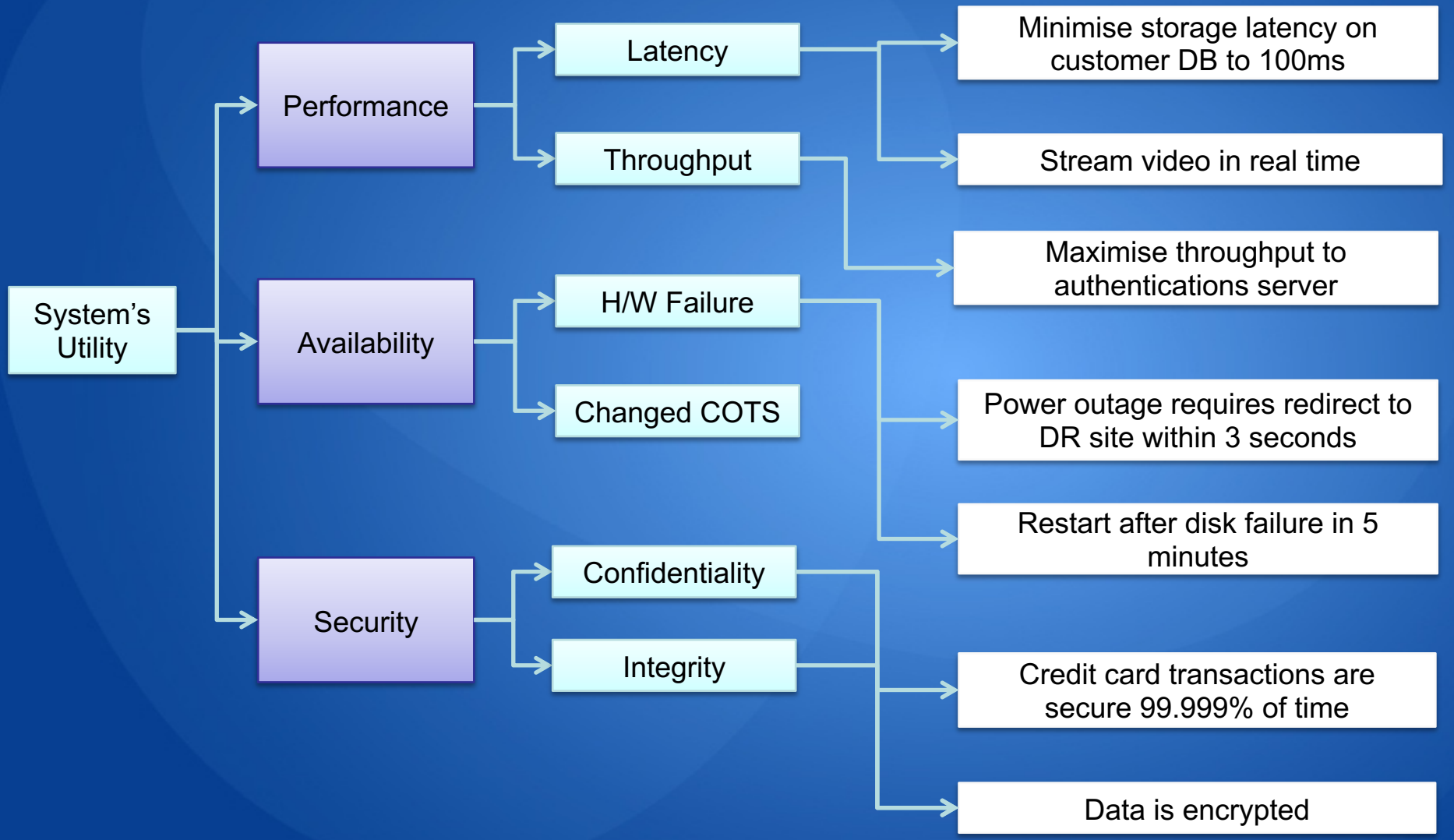
IEEE standard 1061 “Standard for a Software Quality Metrics Methodology” for quality attributes and related terms:

- ***Quality Attribute -*** *A characteristic of software, or a generic term applying to quality factors, quality subfactors, or metric values.*
- ***Quality Factor -*** *A management-oriented attribute of software that contributes to its quality.*
- ***Quality Subfactor*** *- A decomposition of a quality factor or quality subfactor to its technical components.*
- ***Metric Value -*** *A metric output or an element that is from the range of a metric.*
- ***Software quality metric -*** *A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.*

Quality Tree



Quality Tree Example



NFR Use Cases / Scenarios

- **Used to**
  - Represent *stakeholders*' interests
  - Understand quality attribute requirements and how they are actioned.
- **Cover a range of**
  - Anticipated functional uses of (operational use cases),
  - Anticipated changes to (growth use cases), or
  - Unanticipated stresses (exploratory use cases) to the system.
- A good operational use case makes clear what the TRIGGER is that causes it and what responses or ACTIONS (the scenario steps) are of interest.

Types of Non-Functional Use Cases

- **Operational**
  Example: Remote user requests a database report via the Web during peak period and receives it within 5 seconds.

- **Growth**
  Example: *Add a new data server to reduce latency in scenario 1 to 2.5 seconds within 1 person-week.*

- **Exploratory**
  Example: Half of the servers go down during normal operation without affecting overall system availability.

Typical Use Case Scenario Descriptions

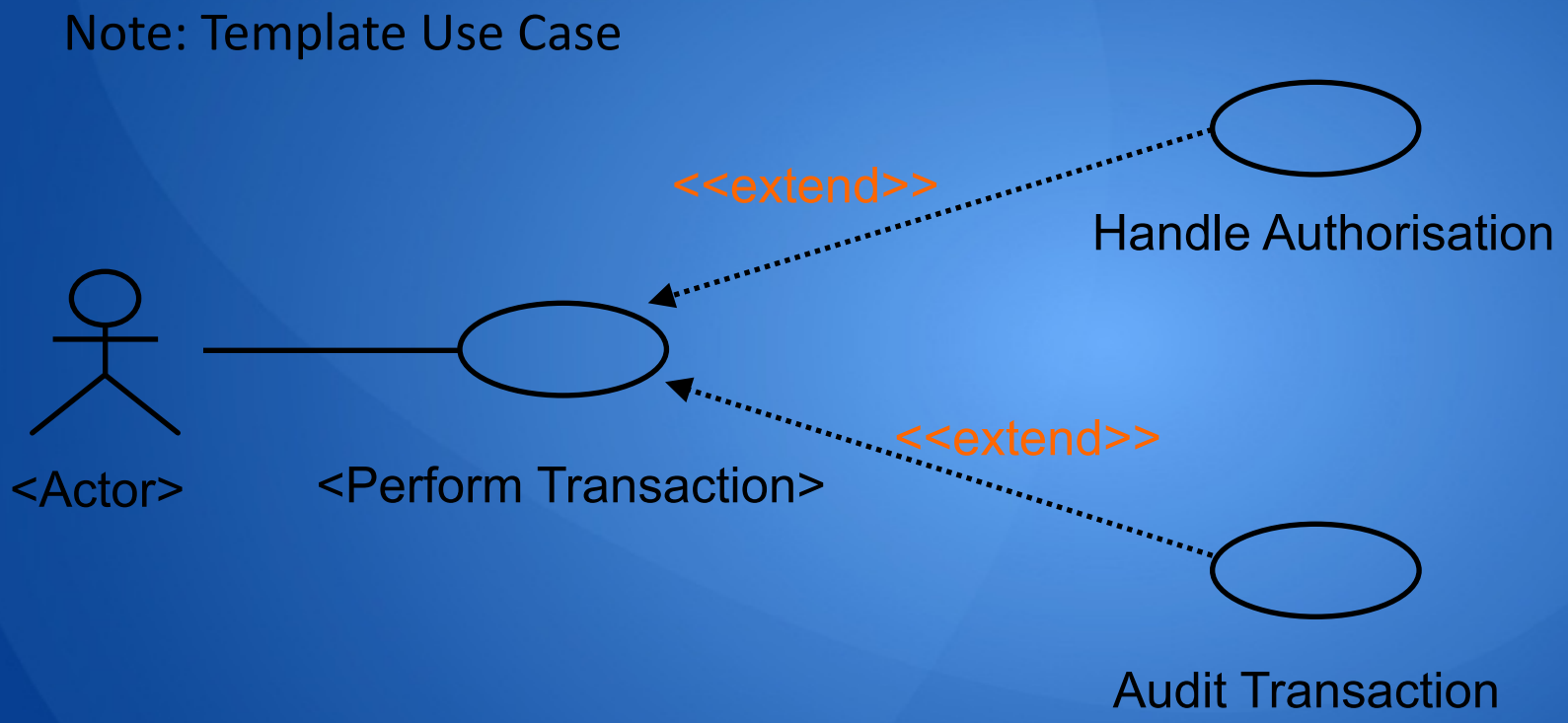
The following are objectively testable, and provide success criteria:

- A remote user requests a large (2 Meg) database report during peak loading of the system between 9:00 am and 11:00 pm weekday. The system provides the report with 10 seconds with no impact on regular web users. Outside this period the report should take less than 5 seconds.
- A system critical to the current order transaction is unavailable and the request cannot be completed. The system sends an alert to the Operations Centre. The user is notified to try again later. The unavailable system is required to be back online with 10 minutes.
- The authority of a user to change and edit their configuration can be removed within 30 minutes of a security breach.

Infrastructure Use Cases

- Added by Jacobsen to capture the need for Infrastructure mechanisms, e.g.
  - authorisation
  - transaction management
- Modeled as ***extensions*** to application use cases. Are Templates or Patterns, for realising each individual use case.

Infrastructure Use Cases Example



Typical Growth Descriptions

These scenarios represent anticipated changes to the system, examples:

- The load on the system increases by 20% and the operators add a new server to the cluster within 4 hours. The new server is deployed within the existing production VM and also at the VM at the DR site. The current applications are deployed onto the new server within the time frame.
- A new application is deployed to the existing application cluster within 1 day. Existing VMs are used to house the application. The new application instance destination is added to the load balancers and firewalls. If the current VMs cannot provide the capacity a new VM is requested and supplied with the application within 2 days.
- Increase the size of the data base tables to incorporate double the number of users and keep the current SLAs for requests within 1 second.
- Migrate the system to a new brand of linux operating system, or a new release of the operating system within one man week.

Typical Exploratory Descriptions

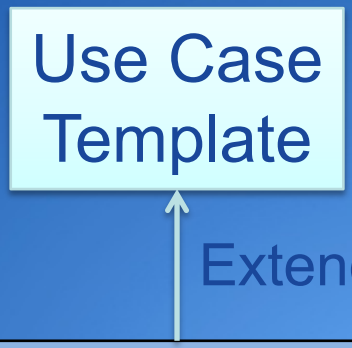
These scenarios push the envelope of the system, example:

- Change the current textual reporting of address information to a Google map view that allows users to drag and drop a pin locating their address, within 3 man months of effort.
- Change the system Platform from Sun Unix to a PC running Windows with two project release periods with the current development team.
- Improve Mobility and make the current Web application available on a mobile phone within 4 man weeks.
- Change the system so that it is reliable when half of the servers fail and the same level of availability is provided.

Use Case Details – Template (repeated)

Attribute	Description
ID	An Identifier, UC-<project number>-[B|O]-<unique identifier>
Name	A short, descriptive name usually verb-noun
Description	A short summary description of the use case, 1-2 sentences
Trigger	A business or operational event, condition or stimulus that starts the use case.
Actors	The source of the trigger. Only the ones initiating the use case (active)
Goal	A description of the goal of the Use case; what the actor is trying to achieve
Main Flow	The normal sequence of actions; the “happy path”
Alternate Flows	Exception and variation paths, alternate actions
Pre-conditions	What must be true prior to starting the use case
Post-conditions	What becomes true after use case ends
Special Requirements	Link to generic NFRs or NFR Use Cases
Time Frame	Required now or a future state and evolutionary goal
Priority	Business priority (high, medium or low)

NFR Use Case Extra Attributes



Attribute	Description
Concern	The refinement of the quality attribute, Broken down into levels, Top - Level 1 - Level 2 etc, e.g. Performance - Response Time.
Environment	Conditions when the trigger occurs, e.g. high load in production
Artefact	What things or part of the system is impacted, e.g. code user interface.
Measures	Measureable and testable aspect of the Flows. Can be viewed as acceptance criteria for the system.

Generic Availability Scenario

Attribute	Possible Values
Actor	Internal or external to system
Trigger	Crash, fault, omission, timing, no response, incorrect response
Concerns	Hardware Failure, Software Failure, Network Failure
Environment	Normal operation; degraded (failsafe) mode
Artifact	Applications, Servers, Network, VMs, ESBs, communication channels, storage etc
Main Flow	System should detect the trigger and do one of: Log the failure, notify users/operators, disable source of failure, continue (normal/degraded) or be unavailable for pre-specified period
Measure	Time interval available, availability%, repair time, unavailability time interval, degraded mode time interval

Graphical Form, Availability Example

Generic Extensibility Scenario

Extensions can be through the addition of new functionality or through modification of existing functionality.

Attribute	Possible Values
Actor	End-user, developer, system-administrator, architect, operator
Trigger	Add/delete/modify functionality or quality attribute
Concerns	New channel, device, product feature, changed COTS
Environment	At runtime, compile time, build time, design-time
Artifact	System user interface, application, platform, environment
Flow	Locate places in architecture for change, make change, test changes and deploy.
Measure	Cost in effort, money, time. Extent impacts other system functions or qualities.

Generic Performance Scenario

Attribute	Possible Values
Actor	Internal and External
Trigger	Periodic events, sporadic events, stochastic events
Concerns	Same as Measures
Environment	Normal mode; overload mode
Artifact	System, or possibly a component
Flows	Process stimuli; change level of service
Measure	Latency, deadline, throughput, jitter, miss rate, data loss

Generic Testability Scenario

Attribute	Possible Values
Actor	Unit developer, increment integrator, system verifier, client acceptance tester, system user
Trigger	Analysis, architecture, design, class, subsystem integration, system delivered
Concerns	Same as Measures
Environment	At design time, at development time, at compile time, at deployment time
Artifact	Piece of design, piece of code, complete system
Flows	Provide access to state data values, provides computed values, observes results, compares
Measure	% executable statements executed; probability of failure if fault exists; time to perform tests; length of longest dependency chain in a test; length of time to prepare test environment

Generic Usability Scenario

Attribute	Possible Values
Actor	End-user
Trigger	Wants to: learn system, use system, recover from errors, adapt system, feel comfortable
Concerns	Understand-ability, Learn-ability, Operability, Attractiveness, Recover-ability
Environment	At runtime, or configure time, install-time
Artifact	System user interface and system (e.g. services)
Flows	To learn system: • Help system is context sensitive • Interface familiar, consistent To use system efficiently: • Aggregation of data and/or commands, Reuse of command or data already entered • Efficient Navigation and Comprehensive searching • Distinct views with consistent operations; • Multiple simultaneous activities To recover from errors: • Undo, cancel, recover from system failures • forgotten passwords To adapt system: customize the system to user liking To feel comfortable
Measure	Task time, number of errors, number of tasks/problems accomplished, user satisfaction, gain of user knowledge, ratio of successful operations to total, amount of time/data lost

Generic Security Scenario

Attribute	Possible Values
Actor	User/system who is legitimate/imposter/unknown with full/limited access. Internal or External. Authorised / Not Authorised, Vast Resources
Trigger	Attempt to display/modify data; access services. Attempt CRUD.
Concerns	Confidentiality, Integrity, Availability, Assurance, Auditing, Non-repudiation
Environment	Normal operation; degraded (failsafe) mode
Artifact	System services and/or data
Flow	• Authenticate user; hide identity of user; grant/block access to data and/or services; allow access to data and/or services; grants or withdraws permission to access data and/or services; • Records access/modifications or attempts to access/modify data/services by identity; stores data in an unreadable format; • Recognize unexplainable high demand for services, and informs a user or another system • Restricts availability of services
Measure	Time /effort/resources to circumvent security measures with probability of success, probability of detecting attack; probability of identifying individual responsible for attack or access/modification of data and/or services; percentage of services still available under denial-of-services attack; restore data/services; extent to which data/services damaged and/or legitimate access denied

Documenting SLR's

- Describe the Quality Attribute Tree
- Fill out the Quality Attribute Scenarios Template.
- Requirements Document
  - Textually or in a table
  - Must be stated in a testable form
  - Must be prioritized
- If specific to a use case, in the use case details, may reference scenarios
- Some requirements, such as evolutionary, are described by their own use case or scenarios.

Prioritise Scenarios

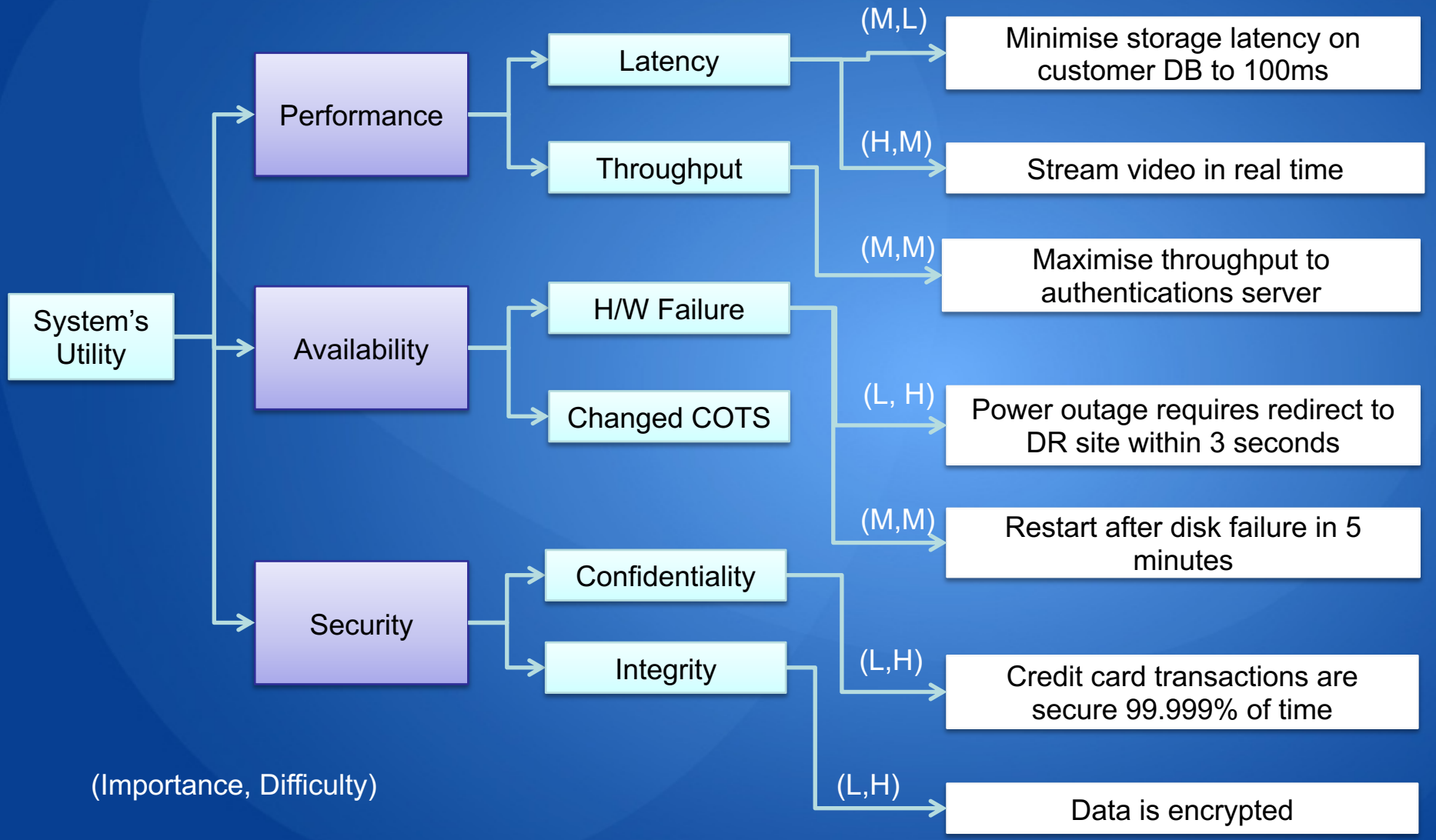
The scenarios are now detailed enough to be prioritised and two dimensions can be used to do this:

- The importance of the scenario to the success of the system
- The degree of difficulty in achieving the scenario

- Can be numeric 1…10, or categorical (L, M, H)

These priorities then feed into architectural analysis. Covered in separate module.

Quality Tree Example Priorities

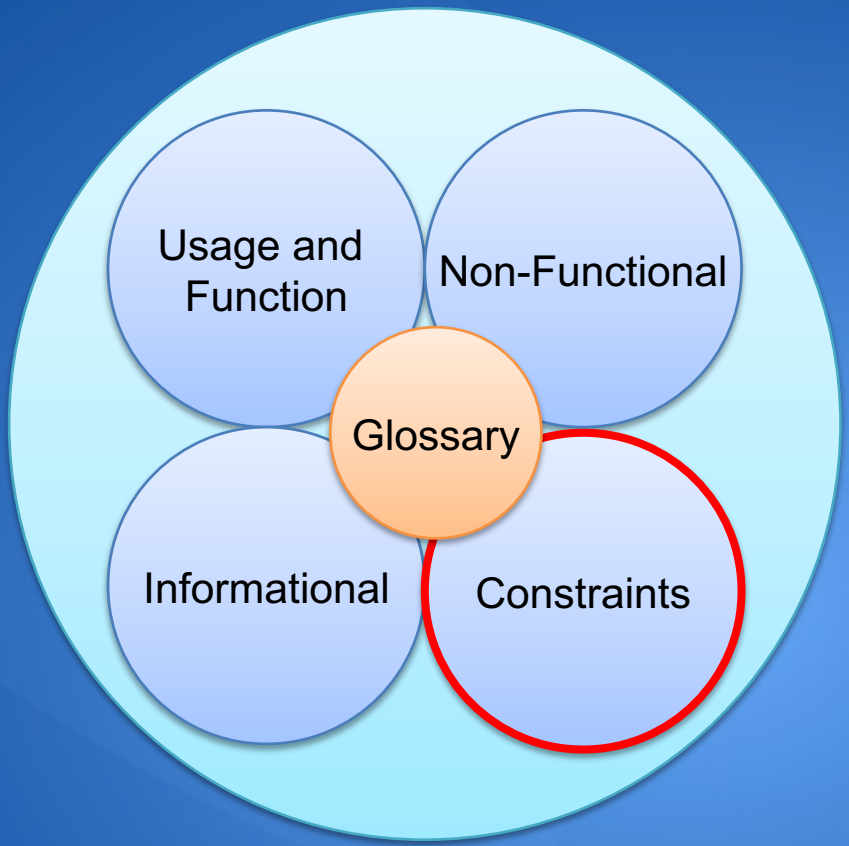


(Importance, Difficulty)

FURPS – Acronym *

- **Functionality** - Capability (Size & Generality of Feature Set), Reusability (Compatibility, Interoperability, Portability), Security (Safety & Exploitability)
- **Usability** (UX) - Human Factors, Aesthetics, Consistency, Documentation, Responsiveness
- **Reliability** - Availability (Failure Frequency (Robustness/Durability/Resilience), Failure Extent & Time-Length (Recoverability/Survivability)), Predictability (Stability), Accuracy (Frequency/Severity of Error)
- **Performance** - Speed, Efficiency, Resource Consumption (power, ram, cache, etc.), Throughput, Capacity, Scalability
- **Supportability** (Serviceability, Maintainability, Sustainability, Repair Speed) - Testability, Flexibility (Modifiability, Configurability, Adaptability, Extensibility, Modularity), Installability, Localizability

* Developed by HP.



Constraints

Constraints

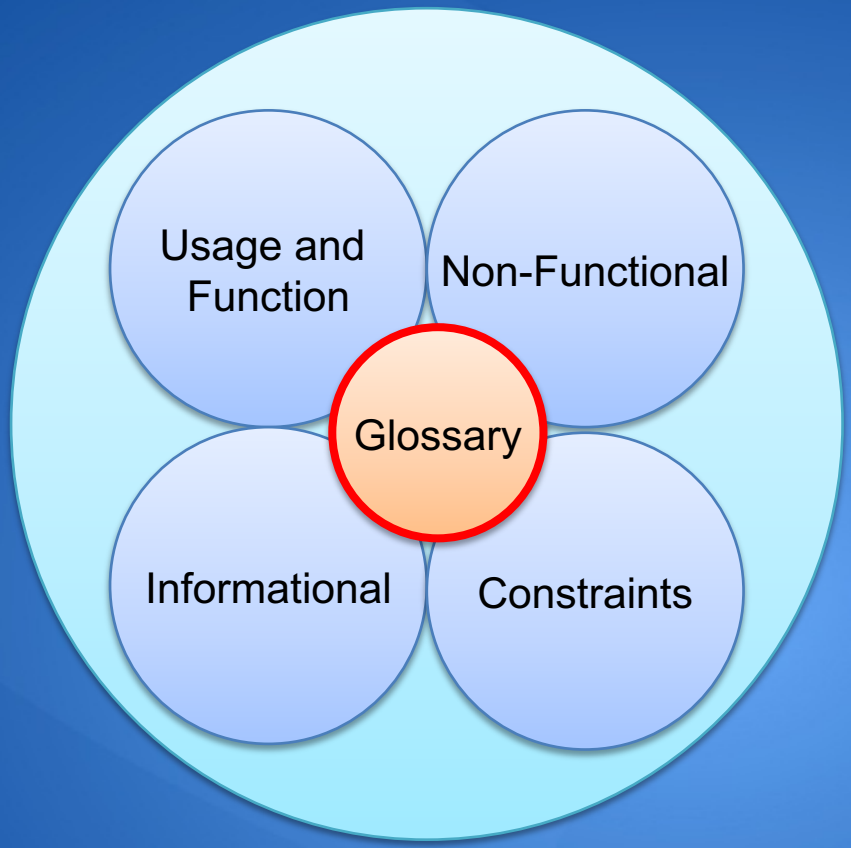
- A constraint is a limitation, dependency or assumption for the "solution“ to the functional and service level requirements
- Constraints are beyond your control that, if not properly managed, could limit the project success.
- Focus on how the system is built or run.
- The system is to be architected within the required constraints.

Types of Constraints

- Development Process and Team
  - Describe the manner in which a project may be formed and operated in order to build the proposed system
- Environment and Technology
  - Describe the environment in which this system will be built and operated
- Delivery and Deployment
  - Describe criteria the system must meet in order to provide a complete solution

Constraints - Examples

- The project development process must be in-line with the Enterprise Methodology
- It is required that operational actors be skilled in JBOSS App Server in order to operate the system
- The system must reuse the following Business Services and Frameworks…
- The system will depend on this legacy system for report generation
- The system functionality is delivered to the user via two communication channels: a browser running on a desktop and a browser displayed in a cell phone
- The following service contract must be signed before the system can operate…


Usage and Function
Non-Functional
Glossary
Informational
Constraints

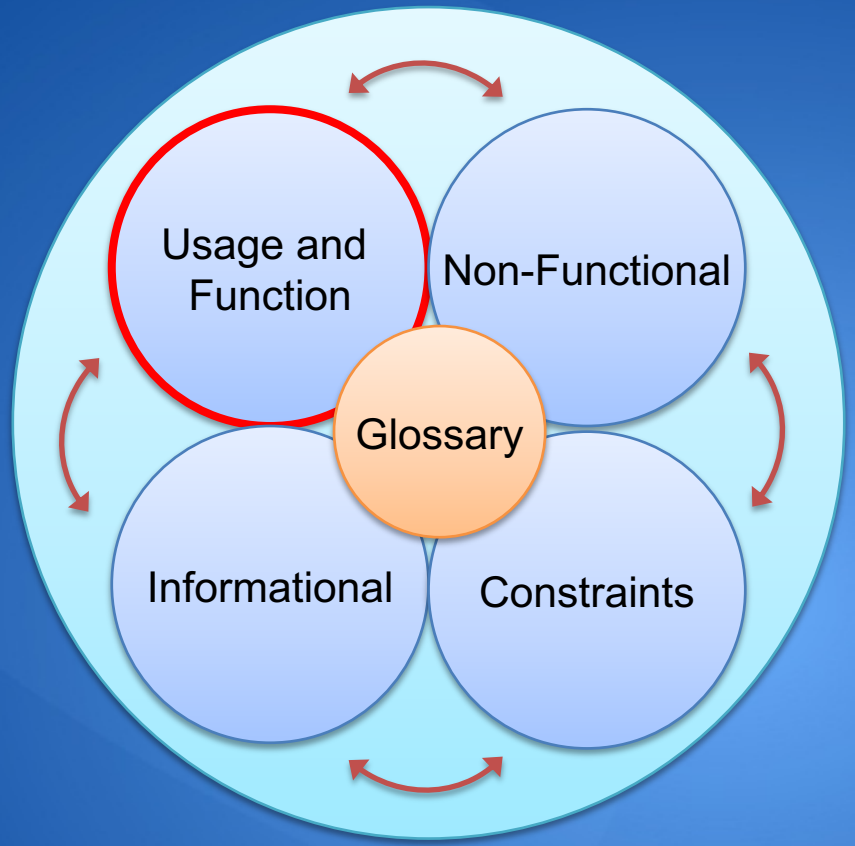
Glossary

Glossary

- An important part of the requirements process is creating a shared an understanding of the domain language, and resolve ambiguity.
- Defines acronyms and TERMS referenced in Vision and Requirements documents;
- Includes all entities in the DM;
- Many glossary items should be accounted for at least once in some use case;
- If multiple terms are used to refer to the same thing, include all terms in the glossary but define only the most common term and reference it from the other terms. Need to agree of a preferred term.

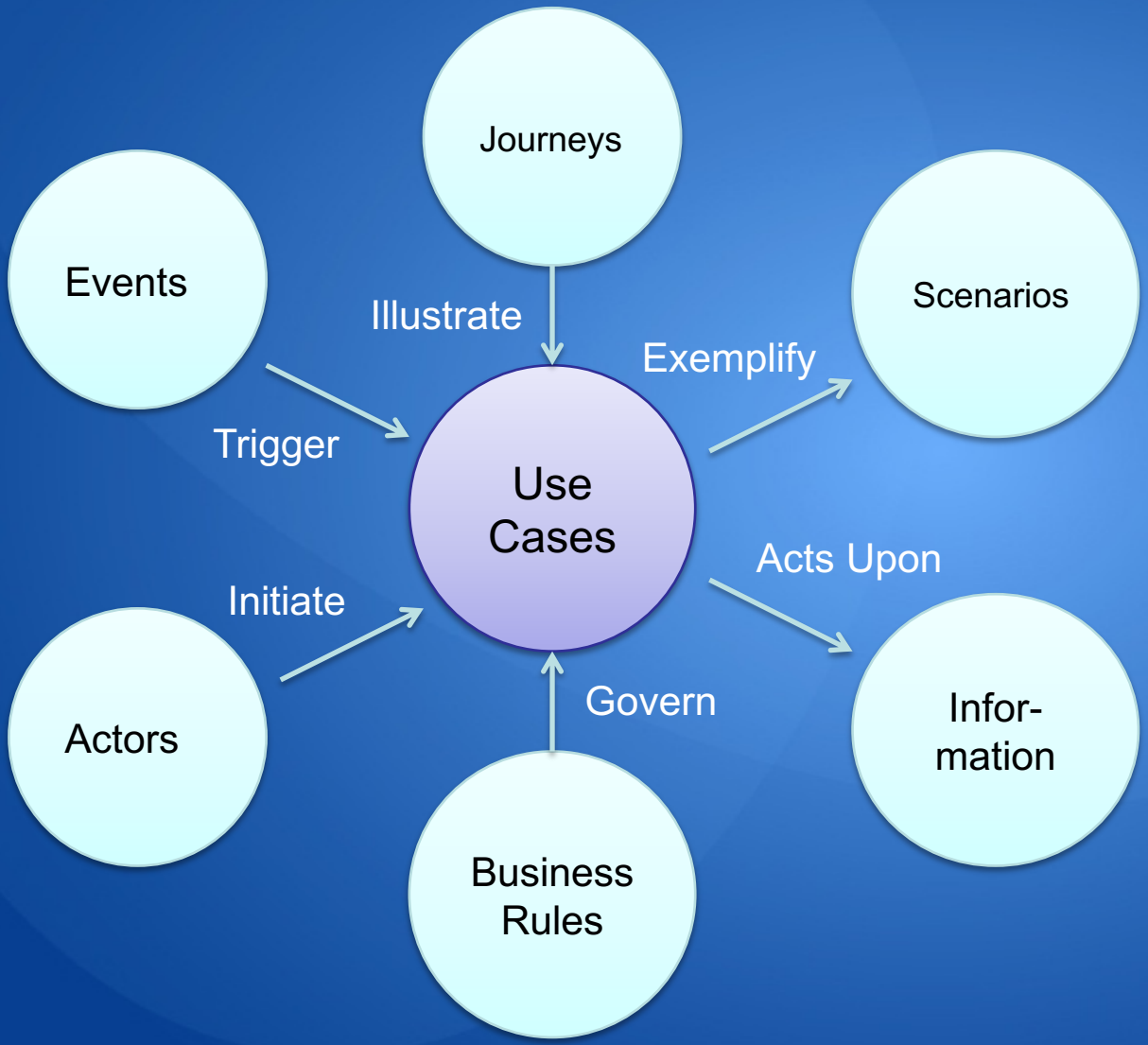
Role of a Glossary

- Necessary for clarity and domain education;
- Agreement on language within the team and source of definition for those outside;
- Without a glossary, everyone has their own interpretation;
- In some cases the lack of a glossary may be catastrophic to a project.



Reinforcement

Models that feed off Use Cases



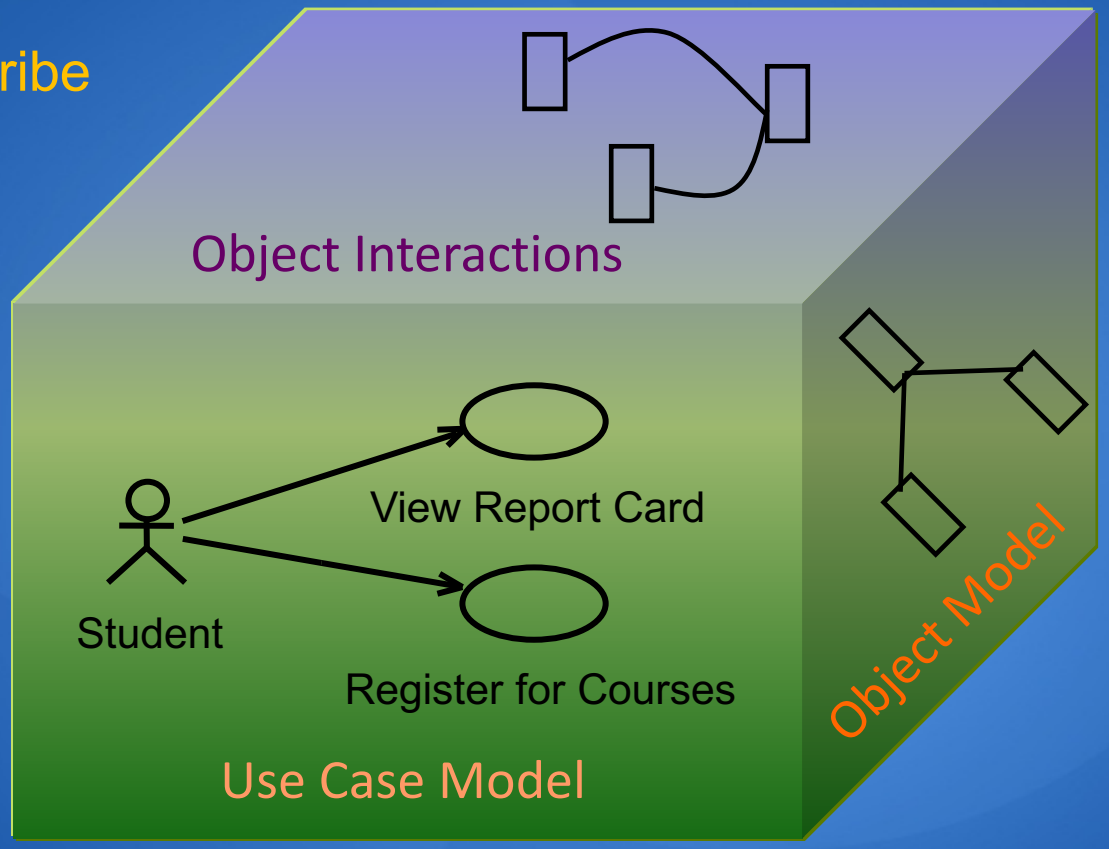
An Integrated View

Use case and object models compliment each other (Armour and Millar), Object model:

- Provides a representation of common business abstractions and their relationships that are a responsibility of the system.
- Aids in consistent description of key concepts;
- Balances the functional aspects of the use case model and reduces functional decomposition of use cases.

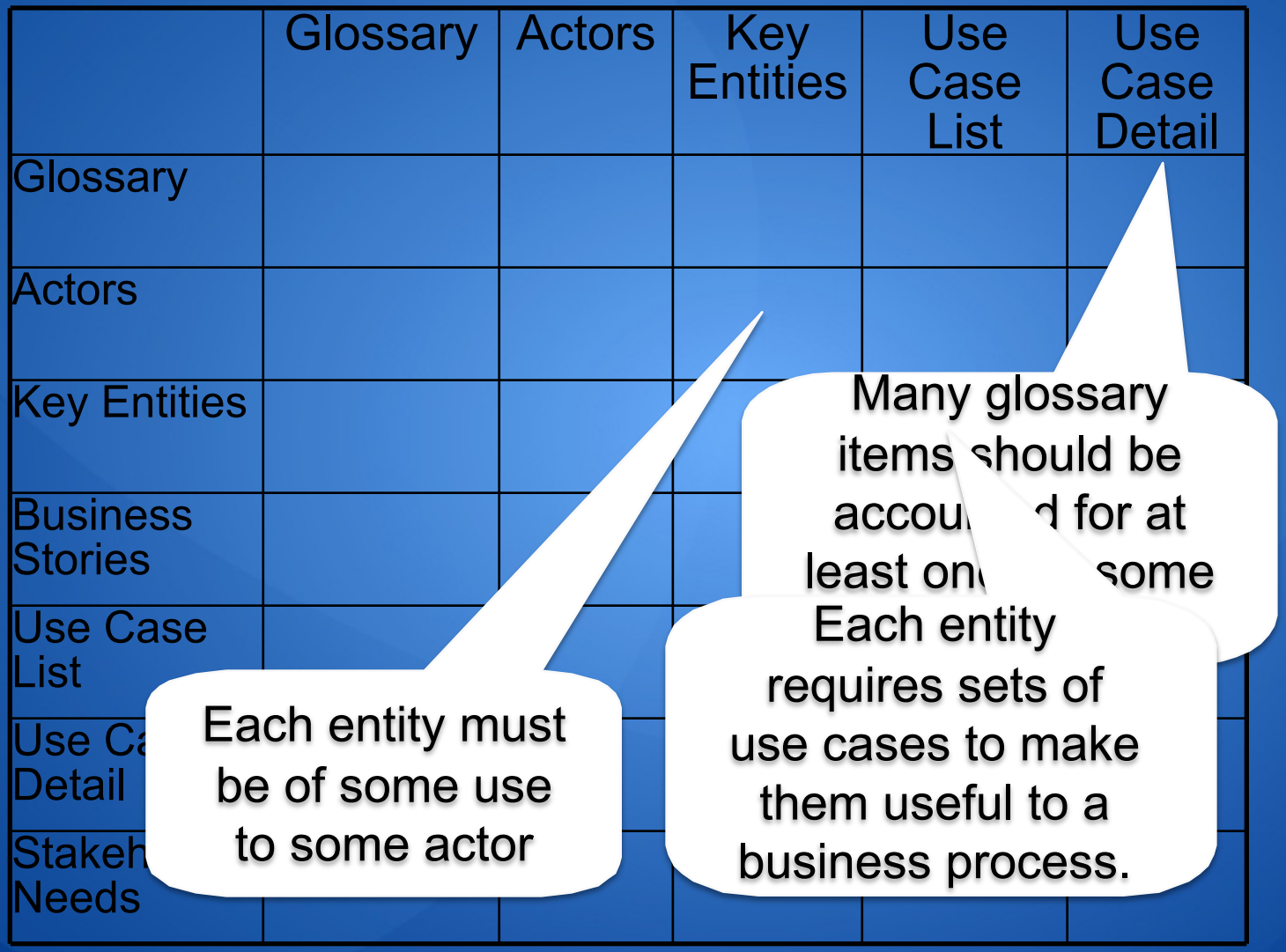
“Requirements” Cube

Armour and Millar describe a requirements cube:



Following topics will cover Use Cases and Object Models

Incremental Reinforcement



The Truth About Requirements

Ideally	Reality
Users know what they want	IKIWISI *
Completed before design	Inefficient, analysis paralysis
Design follows requirements	Also requirements follow design
Requirements are fixed	Business, technology, users, and understanding changes

Iterations and parallel workflows address this discrepancy

* I'll know it when I see it

Good Requirements

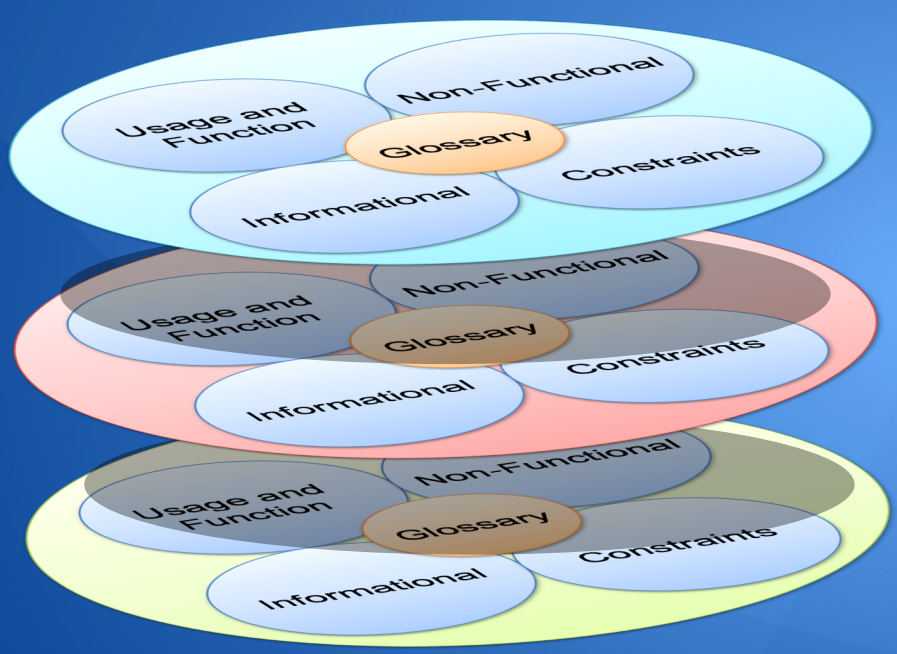
- Clear
- Unambiguous
- Complete
- Correct
- Understandable
- Consistent (internally and externally)
- Concise
- Feasible
- Testable
- Not over-simplified
- Not over-complicated
- No reference to the solution

[Boehm, Barry W., "Verifying and Validating Software Requirements and Design Specifications," IEEE Software, Vol. 1, No. 1, January 1984]

Process and Workflow

Use Cases are Hierarchical

Use Cases can be at any level in an organisation or system



Business Level
Why the project is being undertaken ?

Solution Level
What users can do with the solution ?

System Level
What needs to be built ?

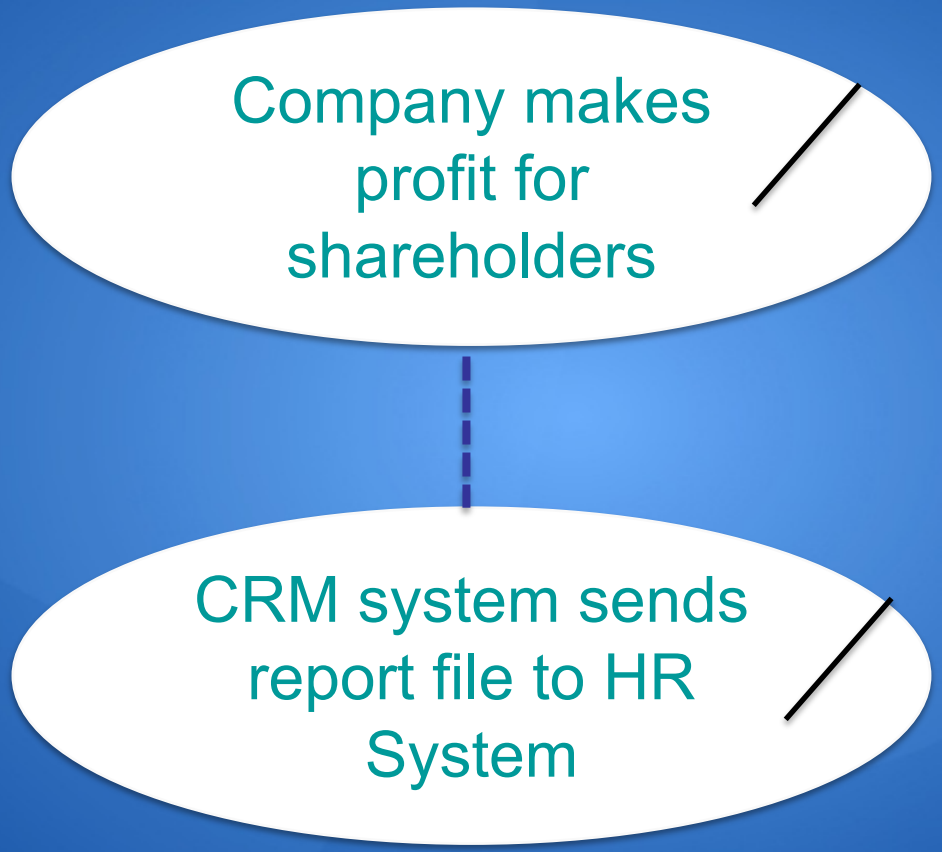
Use Cases Describe Process

- Use Cases detail activity steps in a process, workflow or task - to achieve a goal;
- Use Cases can be detailed at any level in Enterprise;
- Are hierarchical;
- Can be viewed as Activity Diagrams, with Swim Lanes depicting Actors;
- Can be viewed as BPM or BPEL flows;
- Describe a **Contract** between Actors.

Use Cases are Hierarchical

Enterprise Business Level

System Level



Business Level Use Cases

- Taking a ‘Systems Thinking’ perspective and viewing the business (surrounding the system or product) as a system, we can describe actors and use cases at the next level up.
- Therefore we can have:
  - Business Actors
  - Business Use Cases
  - These terms are special in UML and have special notations (diagonal line to the right).
- This next level up is seen as useful in understanding the context of the system.
- In theory this approach could go all the way up, providing traceability to the top of the Enterprise.

‘Business’ Terminology

- **Business Vision:** Vision for the organization in which a system is to be deployed
- **Business Use Cases:** represents a specific workflow in the business and defines what should happen in the business when it is performed; it describes a sequence of actions that produces a valuable result to a particular business actor. A business process either generates value for the business or mitigates costs to the business.
- **Business Use Case Model:** A model of the business goals and intended functions from an external perspective. It is used as an essential input to identify roles and deliverables in the organization.
- **Business Actors:** the role someone, or something plays while interacting with the business.
- **Business Events:** When an association between a business actor and a business use case is named, a corresponding event can be used to signal (trigger) when the use case is initiated, a important happening for the business.
- **Business Goals:** each business use case should support at least one business goal. Goals ensure that business processes execute the business strategy by steering actions at all levels of the organization towards the ultimate business goal, the mission.

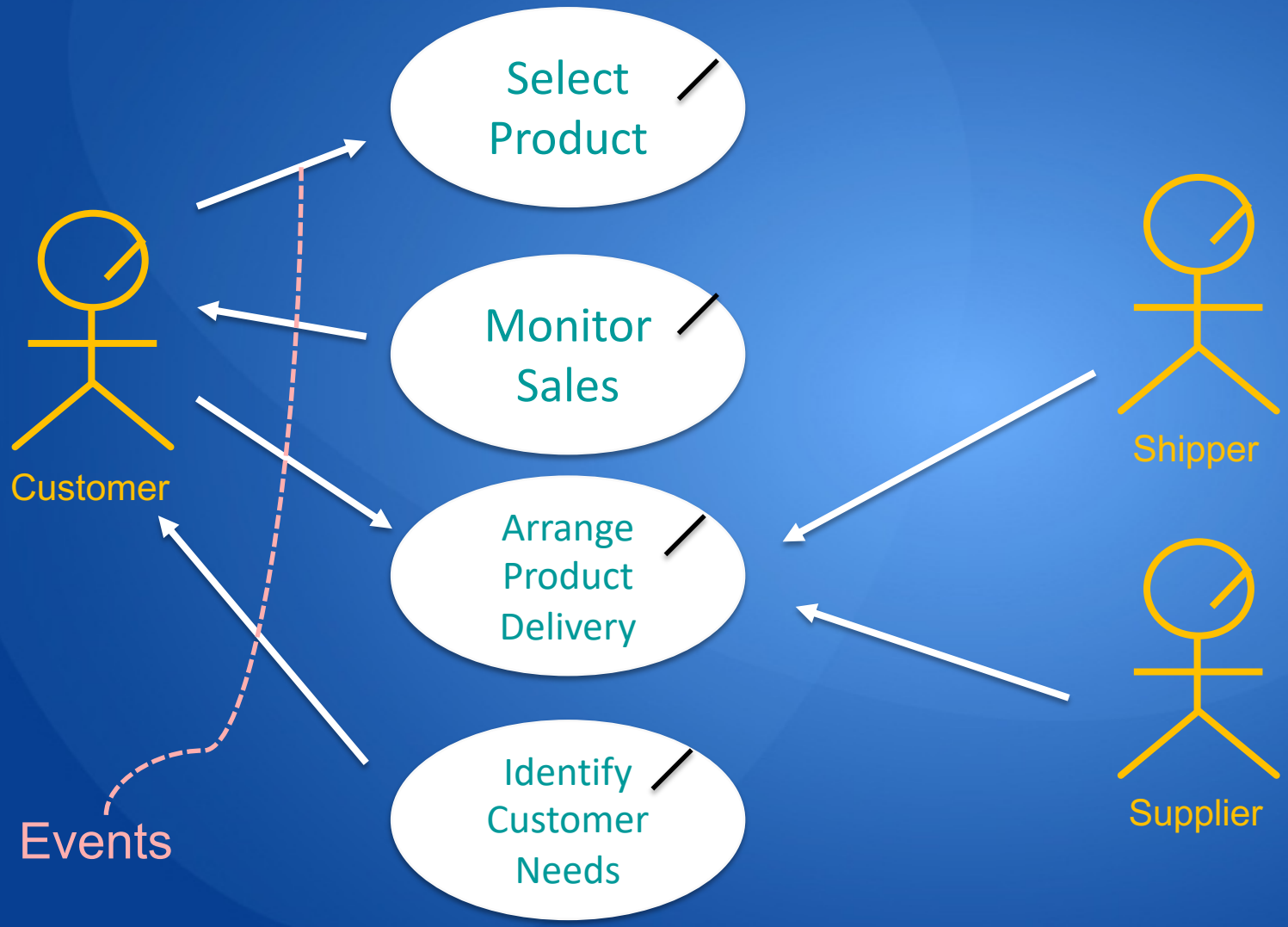
Business Use Case Model

May include:

- Introduction: A textual description that serves as a brief introduction to the model.
- Business Use-Case Packages: The packages in the model, representing a hierarchy.
- Business Goals: The business goals in the model, owned by the packages.
- Business Use Cases: The business use cases in the model, owned by the packages.
- Business Actors: The business actors in the model, owned by the packages.
- Relationships: The relationships in the model, owned by the packages.
- Diagrams: The diagrams in the model, owned by the packages.

- May have 2 models:
  - 'as is' organisation
  - 'to-be' organisation, target as a result of re-engineering

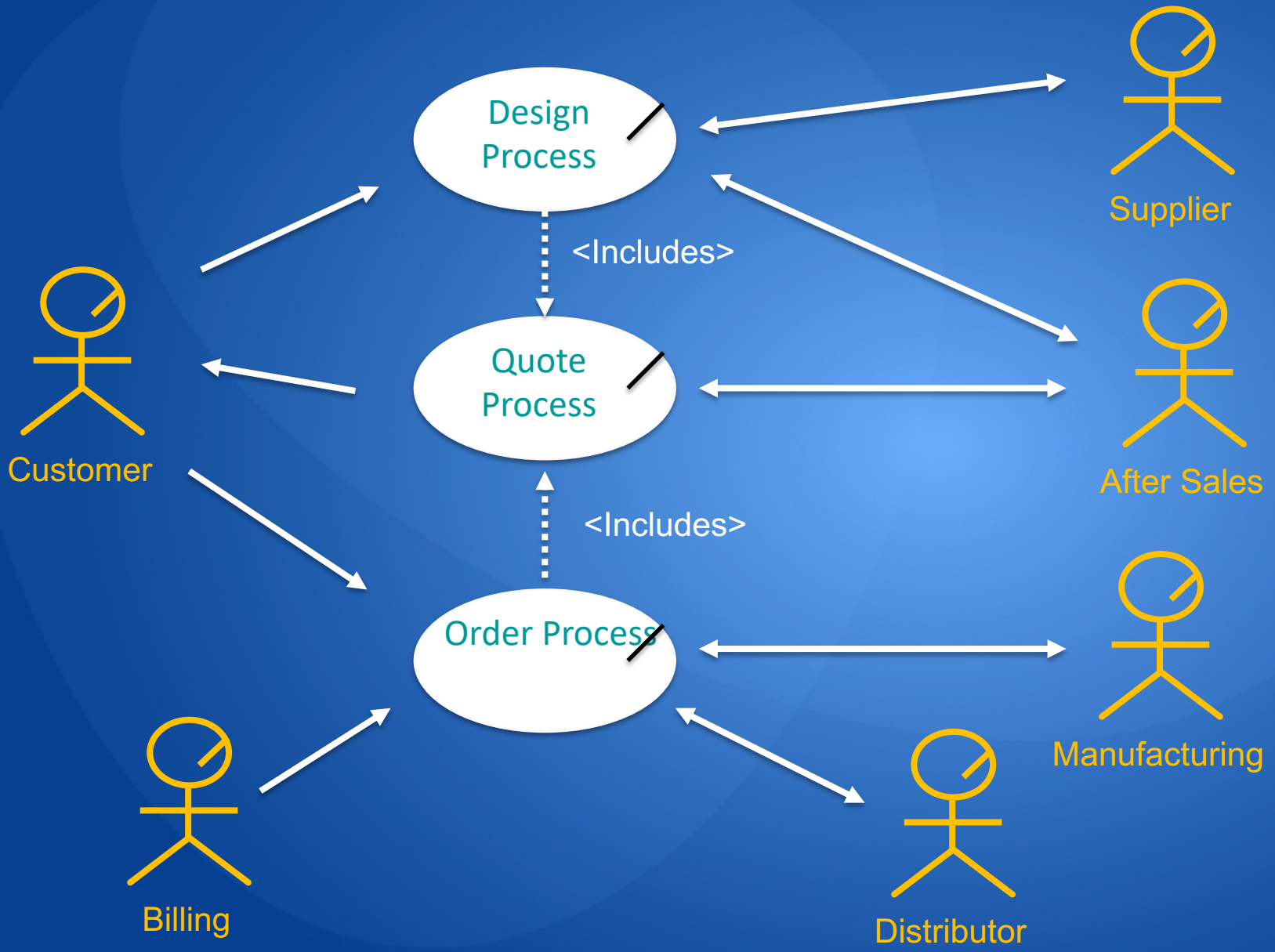
Business Use Case Model



Business Realisation

- A business use case presents an external view of the business, which defines **what** is essential to perform for the business to deliver the desired results to the actor.
- A collection of business use cases gives an overview of the business that is very useful for describing what different parts of the business are doing, and what results are expected.
- A business use-case realization, on the other hand, gives an internal view of the business use case, which defines **how** the work should be organized and performed in order to achieve the same desired results.
- A realization encompasses the business workers and business entities that are involved in the execution of a business use case and the relationships between them that are required to achieve the use case business goal.
- Business Workers show the set of responsibilities a person may carry.
- Business Entities represent deliverables, resources, and events that are used or produced.

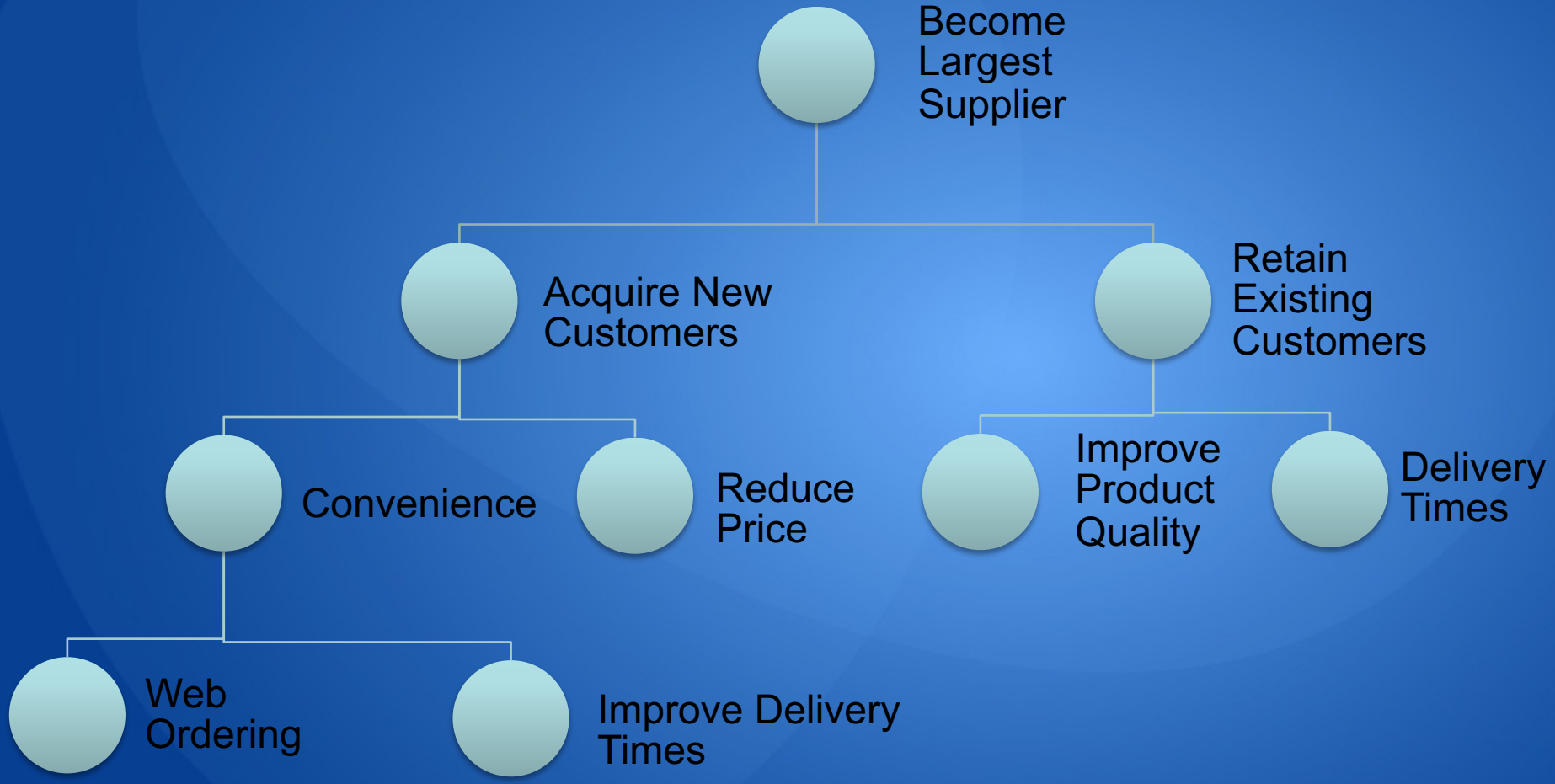
Business Use Case Model



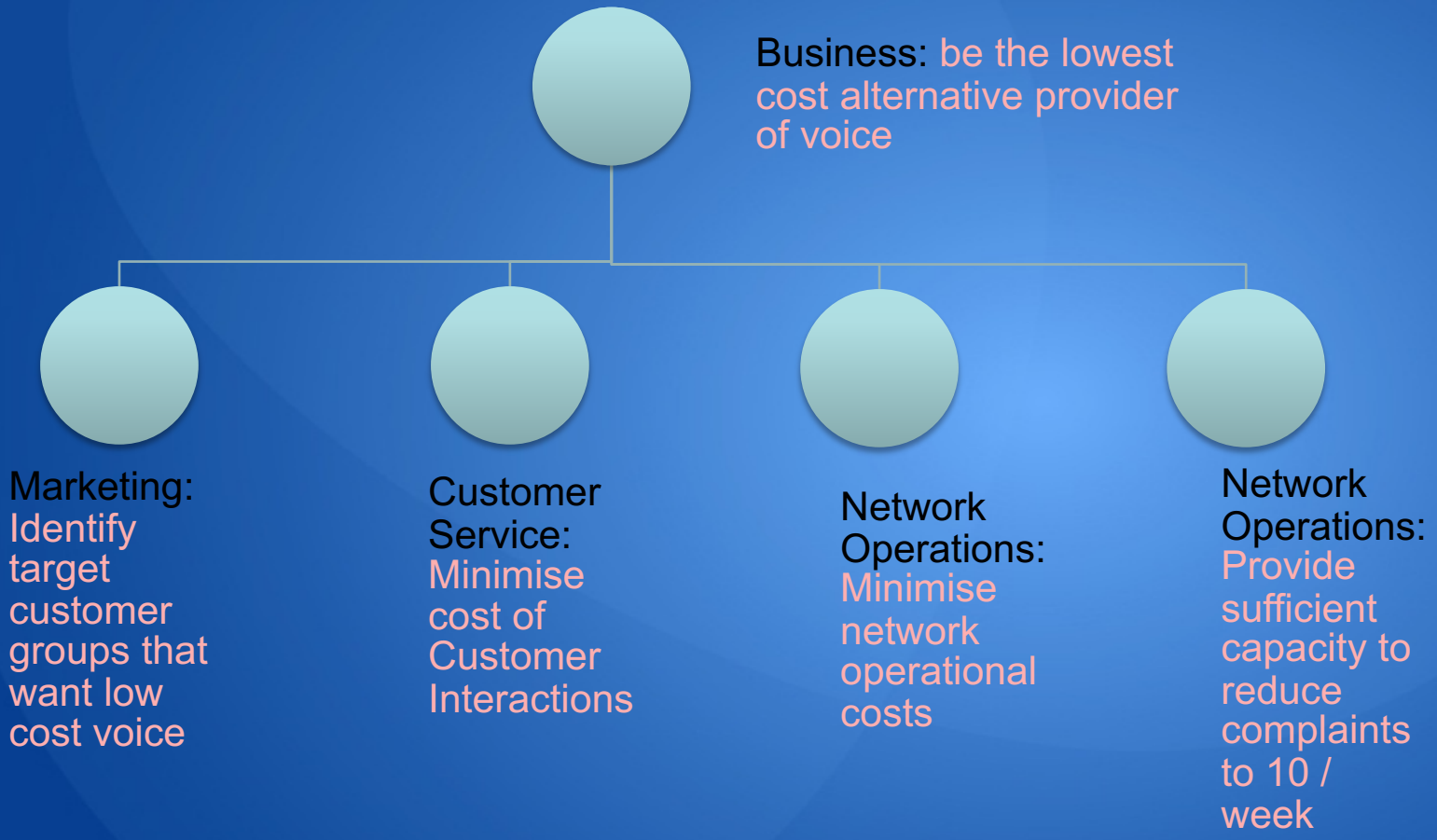
Extent of a Business Use Case

- Sometimes hard to decide if a service is one, or several business use cases.
- Example: airport check-in process. A passenger hands his ticket and baggage to the check-in agent, who finds a seat, prints a boarding pass and starts baggage-handling.
  - If the passenger has normal baggage, the check-in agent prints baggage tag and customer claim check, and terminates the business use case by applying the tag to the baggage, and giving the customer claim check, together with the boarding pass, to the passenger.
  - If the baggage has a special shape or special contents so that it cannot be transported normally, the passenger must take it to a special baggage counter.
  - If the baggage is heavy, the passenger must continue on to the airport ticket office to pay for it, because check-in agents do not handle money.
- Do you need one business use case at the check-in counter, another at the special baggage counter and a third at the ticket office?
  - No, it is only the complete procedure-from the moment the passenger approaches the check-in counter until he has paid the extra charge-that has value (and that makes sense to the passenger).

Business Goal Hierarchy

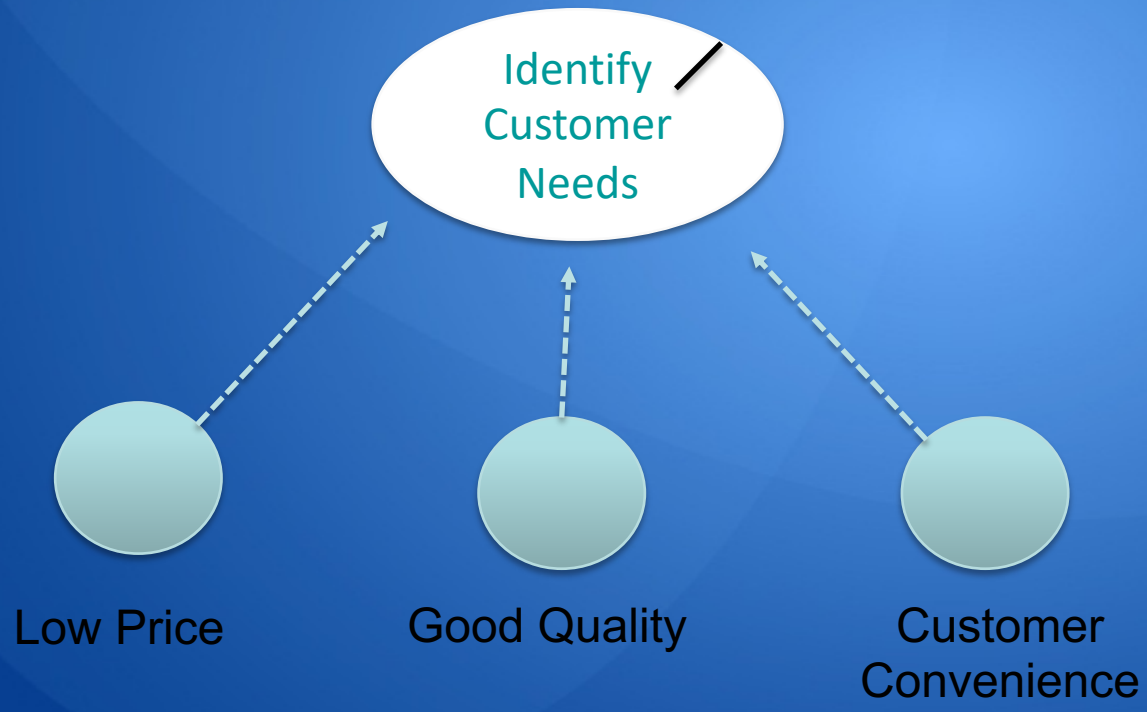


Telco Goals by Business Unit



Business Use Case Supports Goals

- Each business use case should support at least one business goal.
- Business processes are the vehicle with which the business does things.
- Business strategy is achieved via business goals, which ensure that business processes are steering actions at all levels of the organization



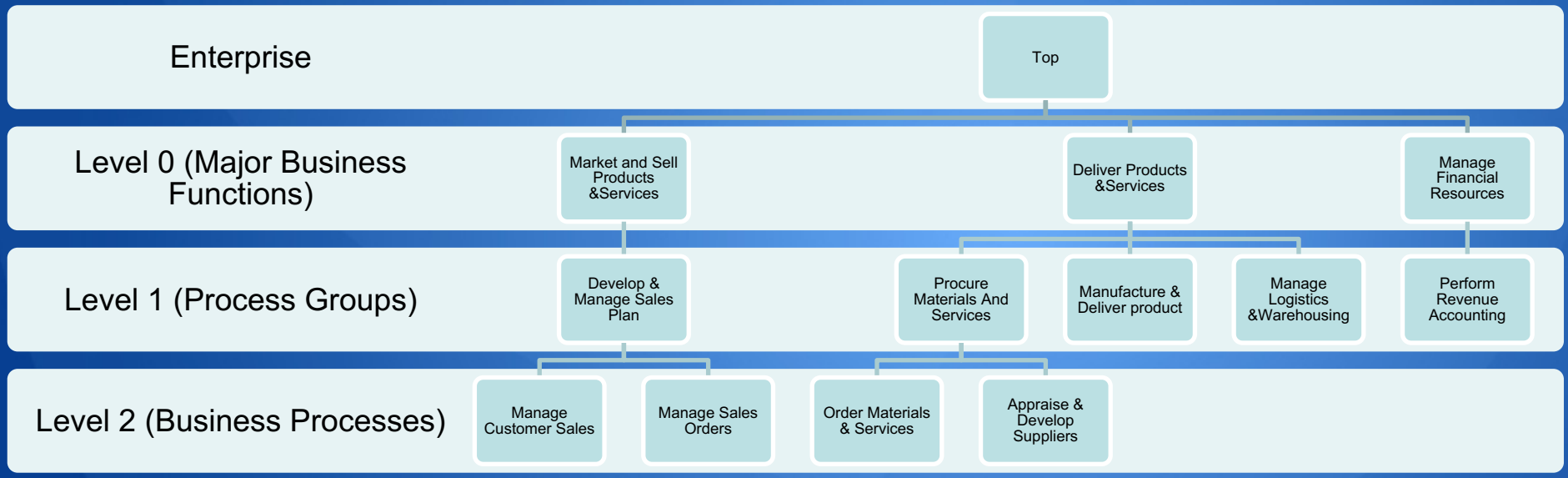
Capabilities and Services

- Goals are achieved with Capabilities and Services.
- A Capability is the ability to act and produce an outcome that achieves a result. It can specify a general capability of a participant as well as the specific ability to provide a service.
- A Capability can specify dependencies or internal process to detail how that capability is provided, including dependencies on other Capabilities.

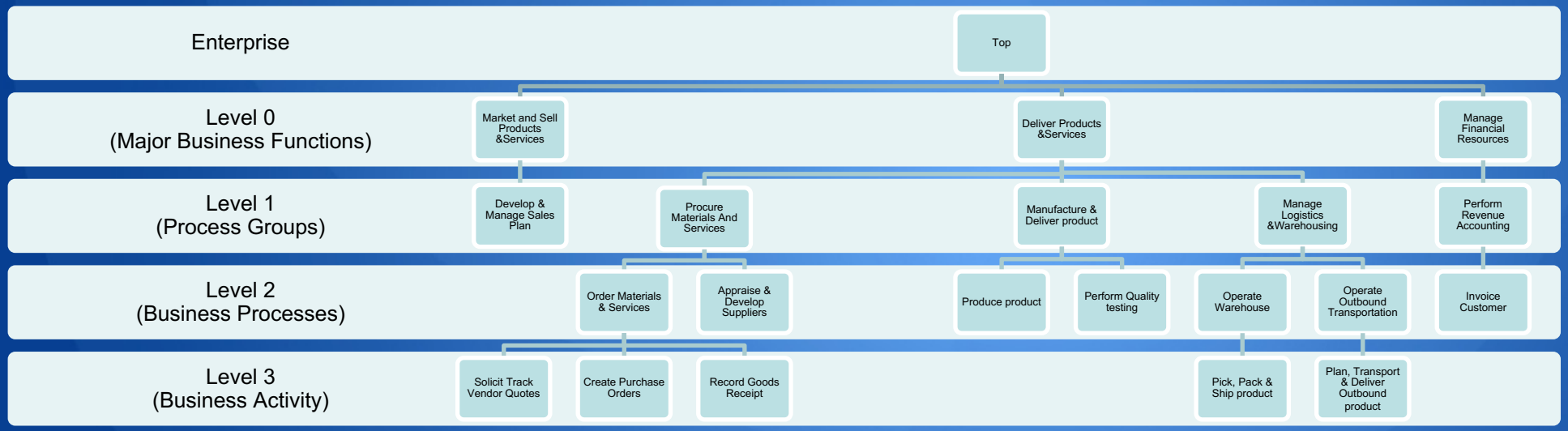
Five Process levels

- Enterprise
- Level 0 - Major Business Functions
- Level 1 - Process Groups
- Level 2 - Business Processes
- Level 3 - Business Activity
- Level 4 - Business Task
- Level 5 - Business Step

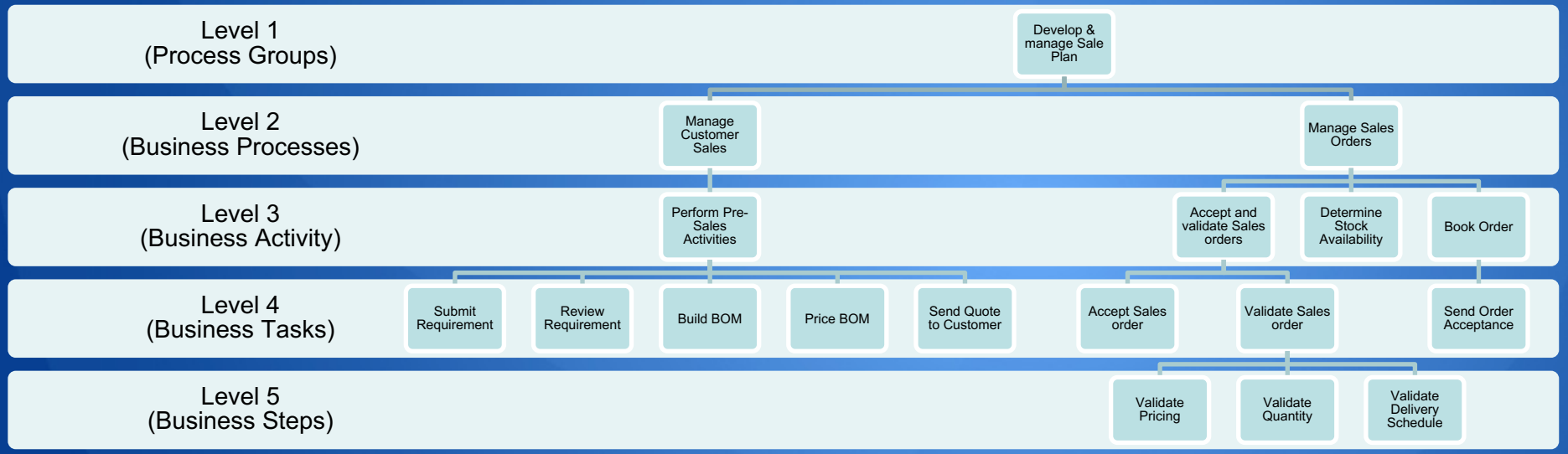
Process Levels



Process Levels



Process Levels



SOA Use Cases

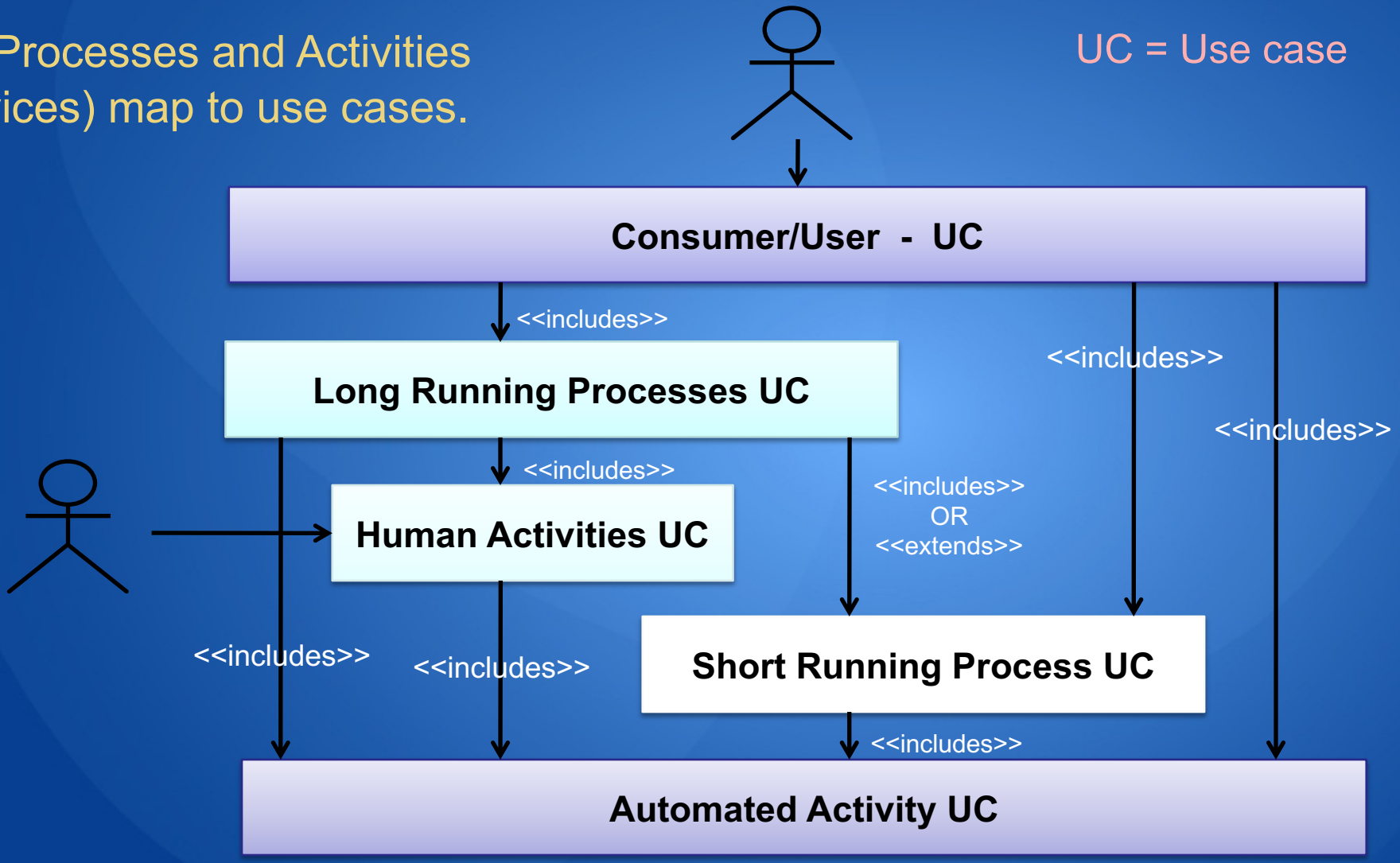
Use Cases and SOA Services

- Create high level **business use cases**;
- Create high level **application use cases** (implement human processes);
- Organise lower level use cases by higher level process ('include' relationship) - capture:
  - Orchestration: Short Running processes;
  - Choreography: Short Running processes;
  - Workflow: Long Running processes.
- **Integration Uses Cases** describe how systems communicate, automated activities and the contracts between them - capture:
  - Orchestrations;
  - Compositions;

Analysis of Processes – Use Cases

The Processes and Activities (services) map to use cases.

UC = Use case



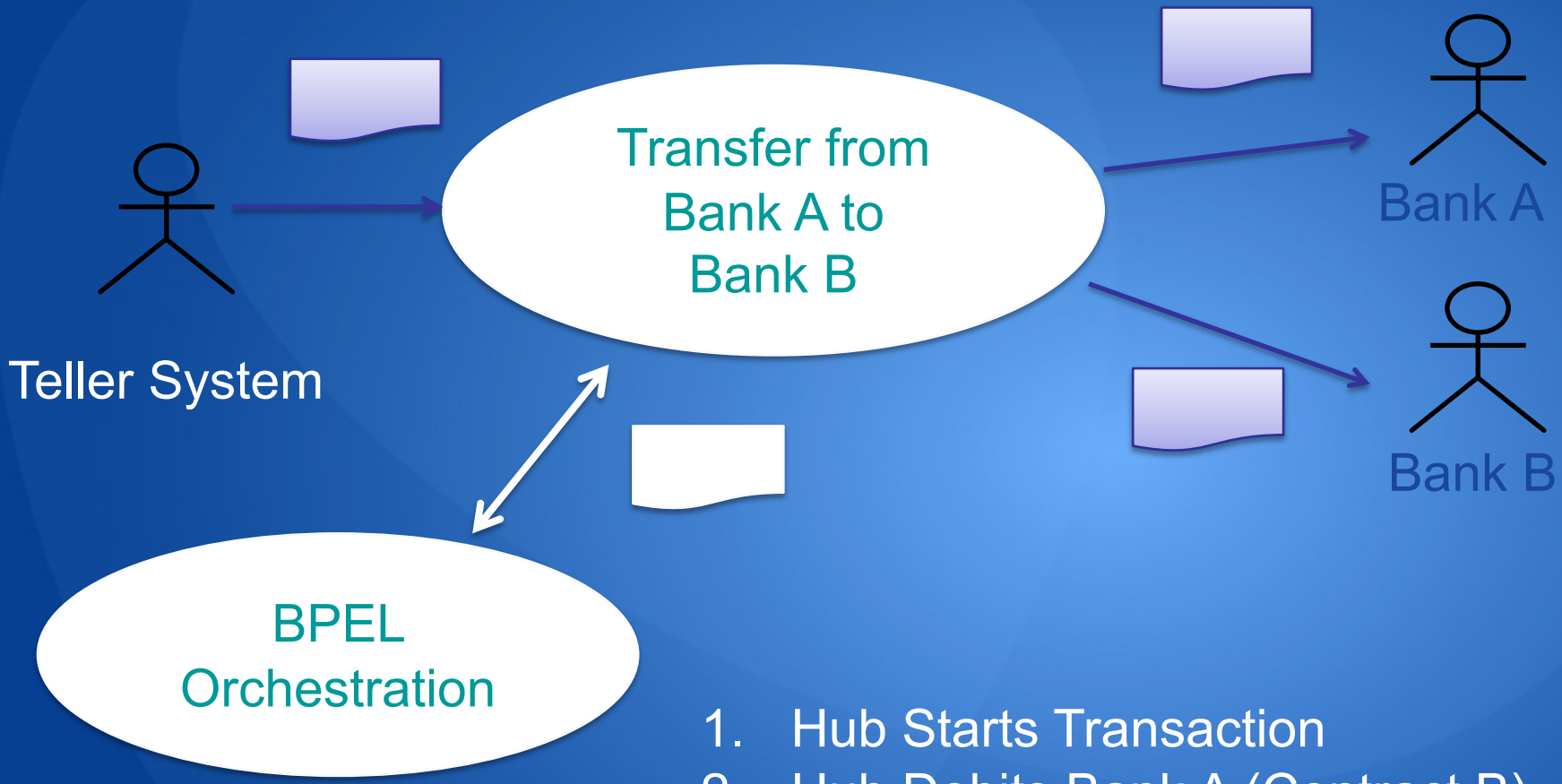
Terminology

- **Manual and Customer Experience Process:** the user interface interaction from an external user or external system (B2B). These processes are outside the system and carried out by the user.
- **Long Running Processes:** are workflow processes that are started by the consumer/user processes user, then later human interaction is controlled by the system.
- **Human Activities** are invoked by long-running processes
- **Short Running Processes**: are service compositions or orchestrations of automated activities that are non-conversational. They inform the user of completion in real-time.
- **Automated Activities:** are atomic steps within business processes. These are candidate service operations. These are steps in a use case.

Integration Use Case

1. The Teller system sends Transfer Amount, Bank A Name and Bank B Name to Hub (Contract A)
2. Hub checks Bank A has the money
3. Hub Debits Bank A (Contract B)
4. Hub Credits Bank B (Contract C)

Contracts & Include Orchestration



1. Hub Starts Transaction
2. Hub Debits Bank A (Contract B)
3. Hub Credits Bank B (Contract C)
4. Hub End Transaction

Service Specification

- Every Service should trace to a Use Case, that provides context.
- Service Specification Includes:
  - **Service Category Table** : the triggering business event and the consequent response by the business (actions), new or old function, gaps ..
  - **Service Definition Table** : more detail on service, its function, description ..
  - **Service Interface** : Input/Output, Methods, BPM Flow, Data Model of Interface ..

Business Events Trigger Actions

Event Number	Business Event	Event Description	Actions in Response to Event
E1	Business event 1	<Description of the event>	R1.1 Response 1 to business event 1 is action. . . R1.n Response n to business event 1 is action
E2	Business event 2	<Description of the event>	R2.1 Response 1 to business event 2 . . . R2.n Response n to business event 2
E3	Business event 3	<Description of the event>	R3.1 Response 1 to business event 3 . . .

The Business Events trigger Use Cases

Service Category Table

Response	Description	Service Category	Existing/New Systems
<Response>	<Description>	<Category>	<Systems>
R1.1 Response name	What are the actions or services performed?	Service category	<Category>
Rn.n Response name	What are the actions or services performed?	Service category	<Category>

Service Description

Service	Functions	Description	Existing/New
Service Name	Response number and name,for responses 1 through n.	Implementation description	<Systems>
Service n	Response number and name, for responses 1 through n.	Implementation description	Example: database, ERP, mainframe, new

High Level Service Interface

Service:	Service Name
Inputs	List all inputs, Requests
Outputs	List all outputs, Responses
Operations	List all operations
Operation descriptions	Description of each operation using • informal description or • pre/post condition template • example showing typical calling usage
Protocol	EDI, SOAP/HTTP, REST, TEXT etc…
Exceptions	The name and data content for each operation's exceptions
Service Level	Non-functional requirements to be met by operations
Implementation	Web service, packages adapter, custom adapter, application API, other

Separate Spec for Orchestrations

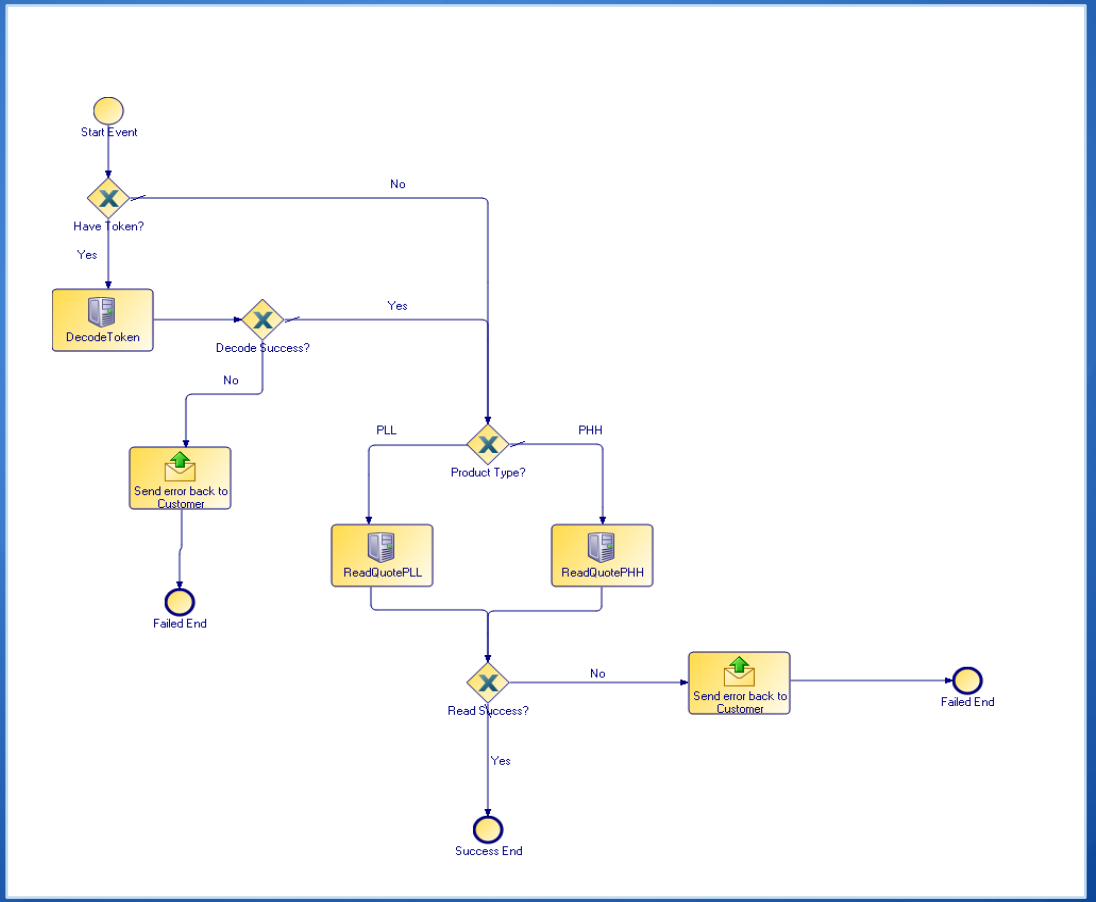
Orchestrations include a Process description:

- BPM
- BPEL
- Activity Diagram

Exceptions:

- Alternate/exception flows.
- Exceptions/faults /errors received from other services.

Compensations and SAGAS



Request/Response Data Model