Agile Requirements with User Stories

Trainer: **Anh To**
Business Analyst

Introduction

- Your name
- Your role
- Your background and experience in
  - Software Development
  - Software Requirements
- What do you want from this course

Course Objectives

At the end of the course, you will have acquired sufficient knowledge to:

- Read and Understand requirements in Agile projects
- Identify User Story/EPIC/Theme
- Split user story
- Write user story effectively
- Differentiate between the requirement in Agile projects compared with non-agile projects

Agenda

Course Audience and Prerequisite

- The course is for participants who would like to understand more about how to read, analyze and document requirements in Agile projects
- The following are prerequisites to this course:
  - Have knowledge about Agile process

Duration and Course Timetable

- Course Duration: 6 hours
- Course Timetable:
  - 2 sections
  - Break 15 minutes in each section

Further References

- Agile Course(s) on Skills port:
  - Planning an Agile Software Development Project

- Training Materials:
  - JIRA AGILE - An Introduction for Agile Project Management
  - Workshop 1-User Story and Product Backlog Grooming
  - Agile Requirements

Course Administration

- In order to complete the course you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Pass Final Test: 7/10
  - Provide your feedback in the End of Course Evaluation

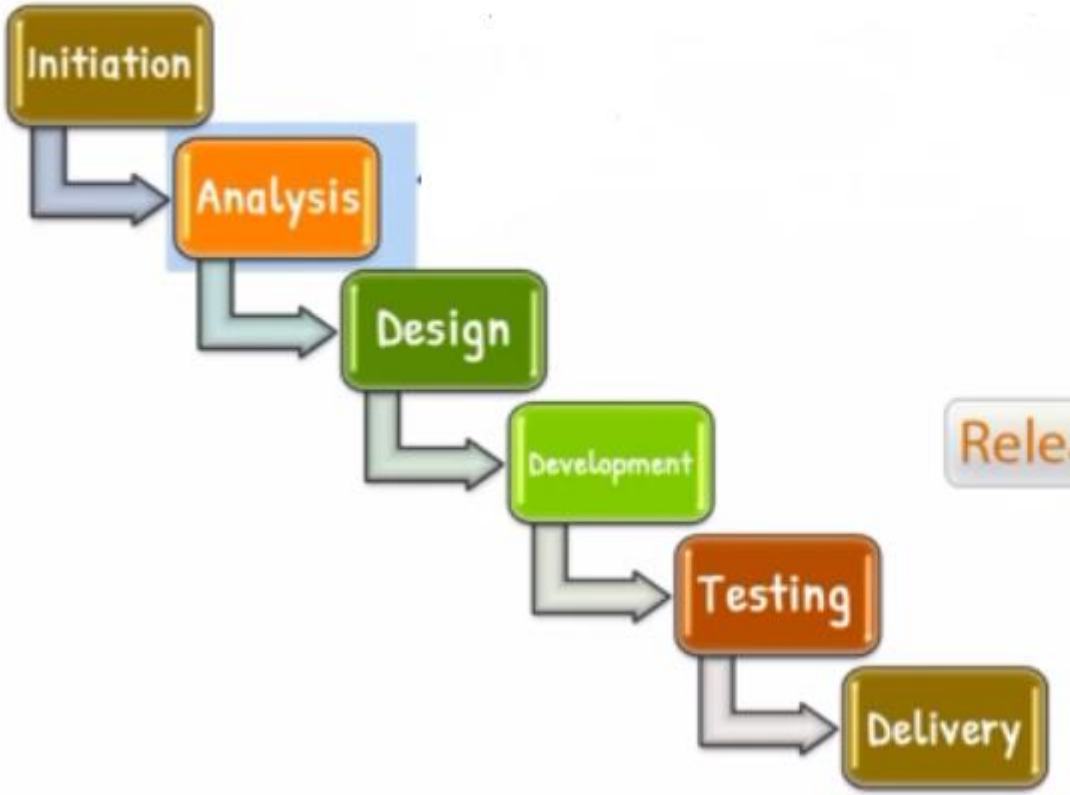
Requirement Overview

What is a “Requirement”?

- A requirement is:
  - A condition or capability needed by a stakeholder to solve a problem or achieve an objective (1)
  - A condition or capability that must be met or possessed by a solution or solution component to satisfy a contract, standard, specification, or other formally imposed documents (2)
  - A documented representation of a condition or capability as in (1) or (2)

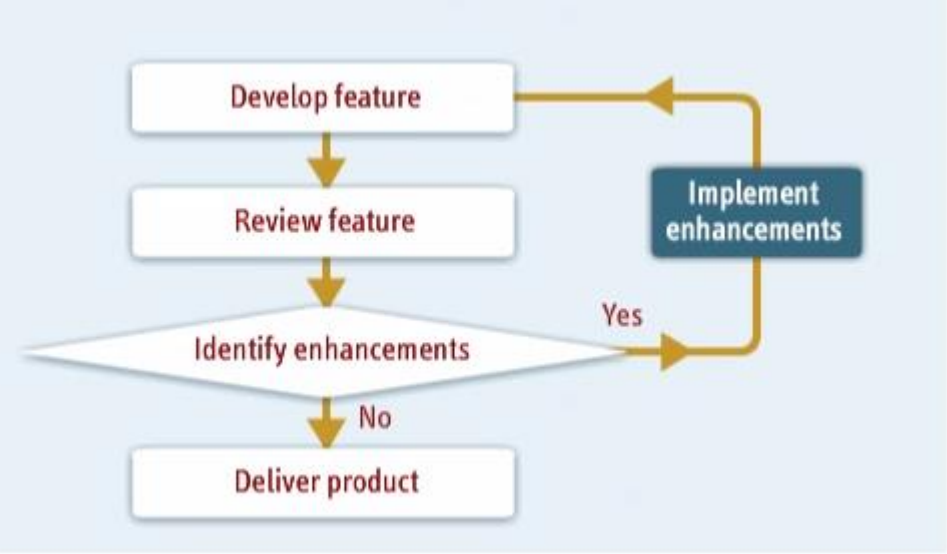
Requirements in Traditional and Agile Software Development Life Cycle (SDLC)

Traditional Process



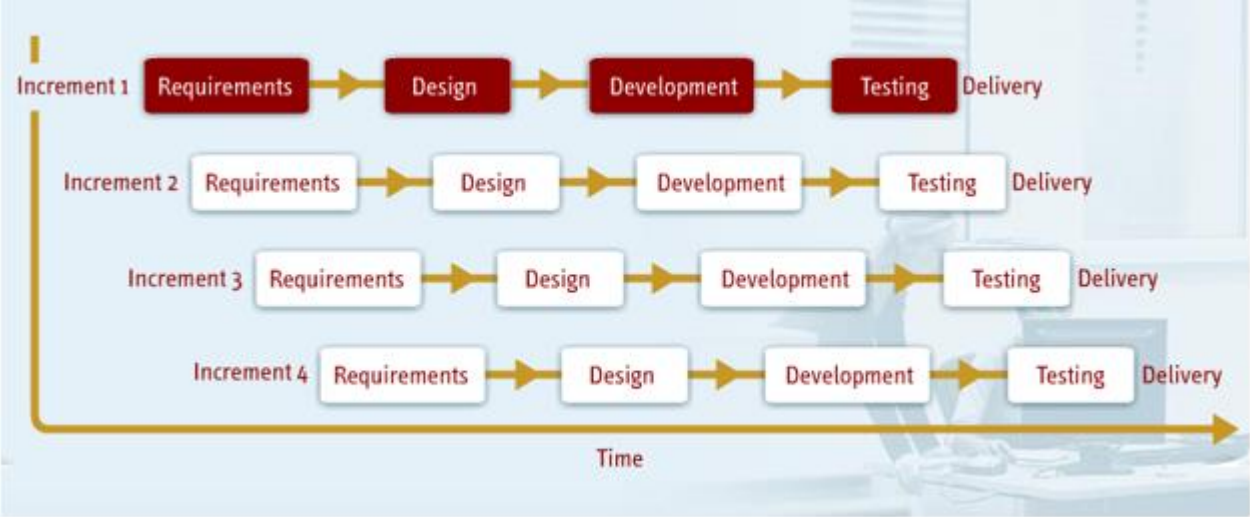
Agile Process

Requirements in Traditional and Agile Software Development Life Cycle (SDLC) (cont.)

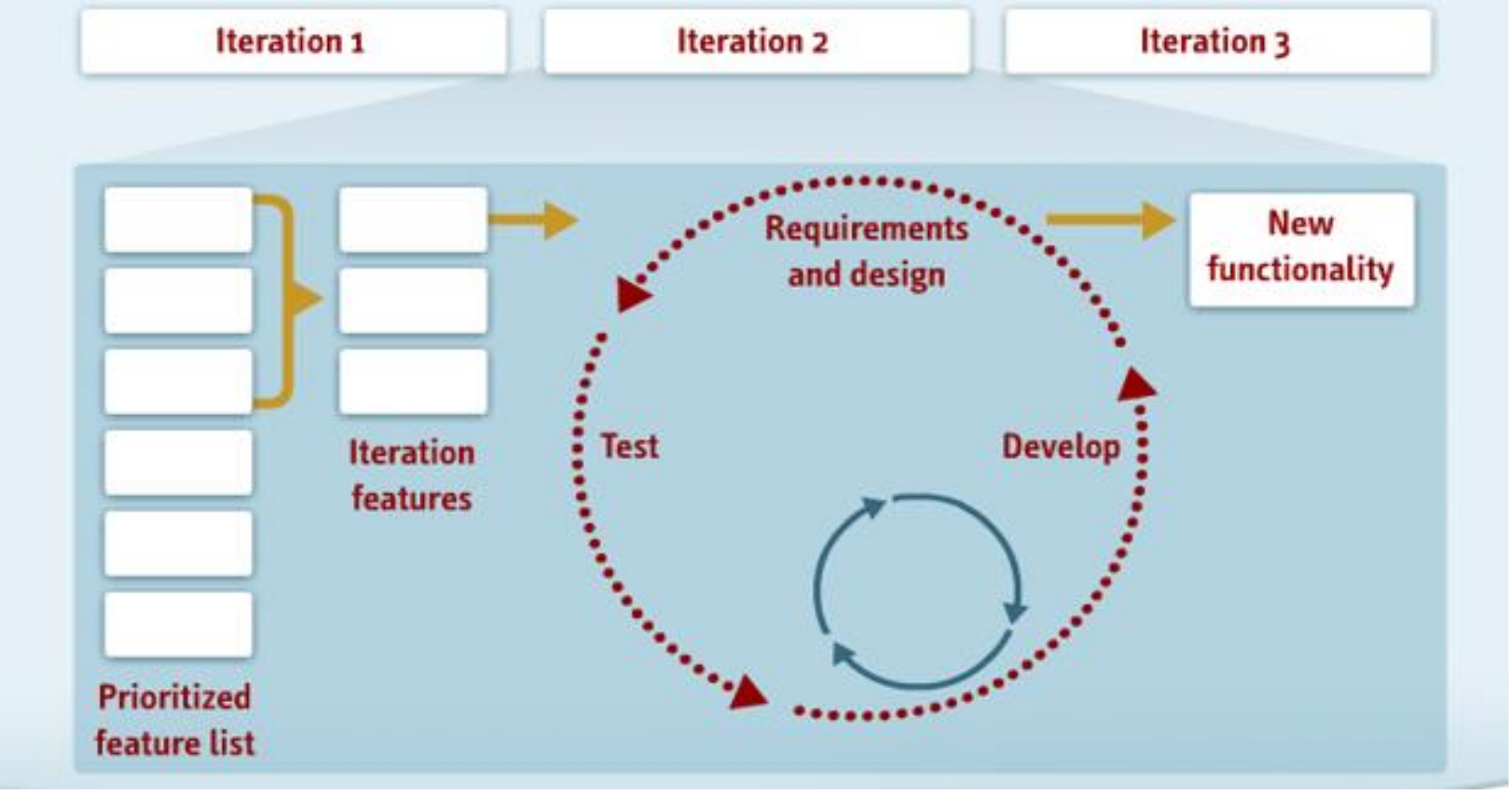


Change Request Workflow

Incremental Process



Requirements in Traditional and Agile Software Development Life Cycle (SDLC) (cont.)



Agile Model

Functional Requirements

- Define WHAT a system suppose to accomplish, a function of a system and its components
- Are supported by non-functional requirements (also known as quality requirements)
- Are expressed in the form "system must do <requirement>“
- As defined in requirements engineering, functional requirements specify particular results of a system
- Functional requirements and agile processes:
  - User Story and scenario

Non-Functional Requirements

- Define HOW a system suppose to do, specify “how well” the “What” must behave
- Quality expectation
- Represent system-level constraints that typically cut across functional requirement during the design or implementation (such as performance requirements, security, or reliability)
- Are expressed in the form "system shall be <requirement>",
- Affect the design and testing of most or all stories in the Product Backlog in Agile or the whole system in traditional projects

- Non-functional requirements and agile processes
  - Improving quality during construction
  - Improving quality during execution

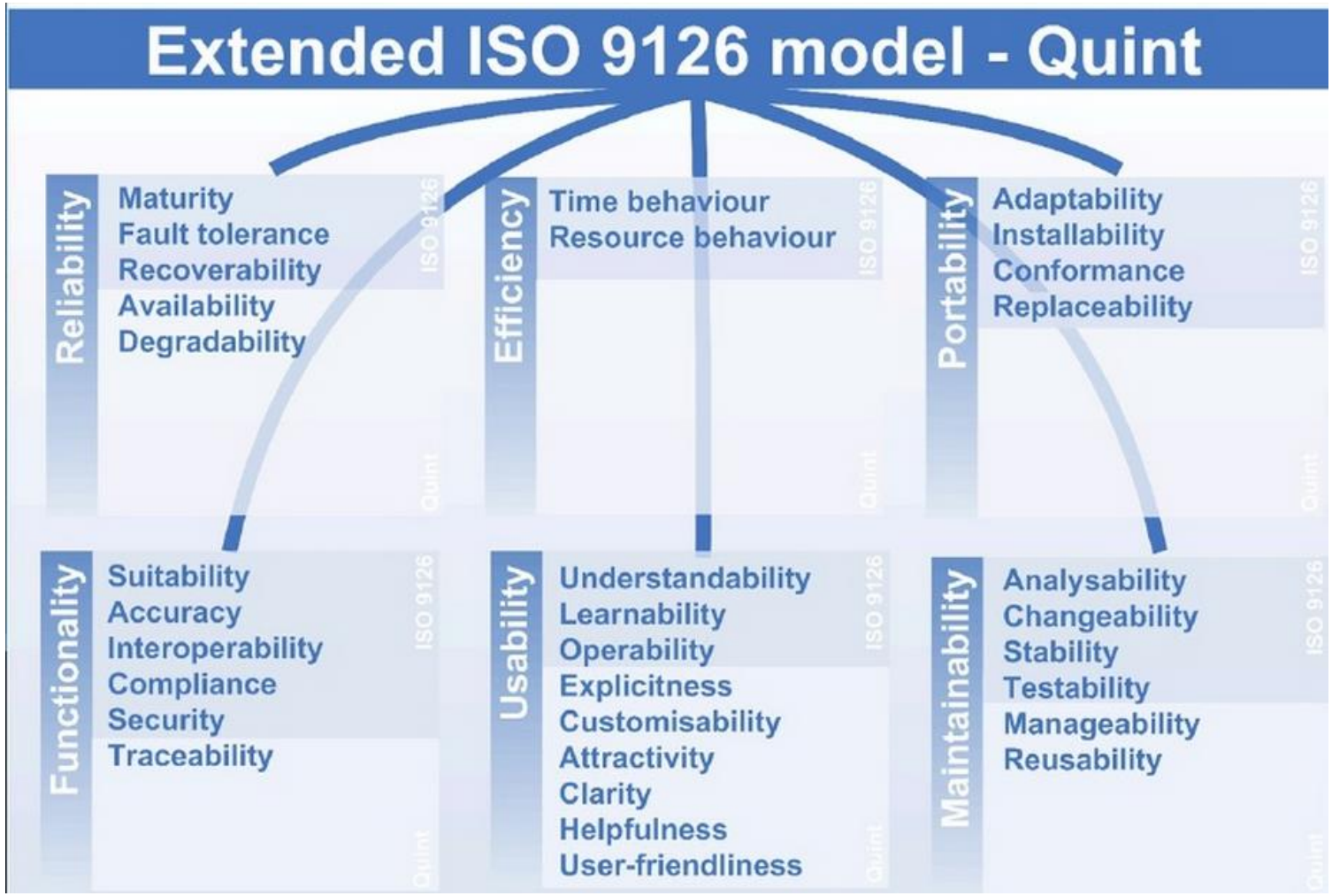
Non-Functional Requirements (cont.)

- If your nonfunctional requirement is not objectively measurable, you needs to revise, rewrite, or expand it
- Measurable objectives: 10,000 transactions per hour, 1 second response time, six packs of beer
- Subject quality: easy to maintain, high quality, good beer => not objective measurable => need to clarify

Non-Functional Requirements (cont.)

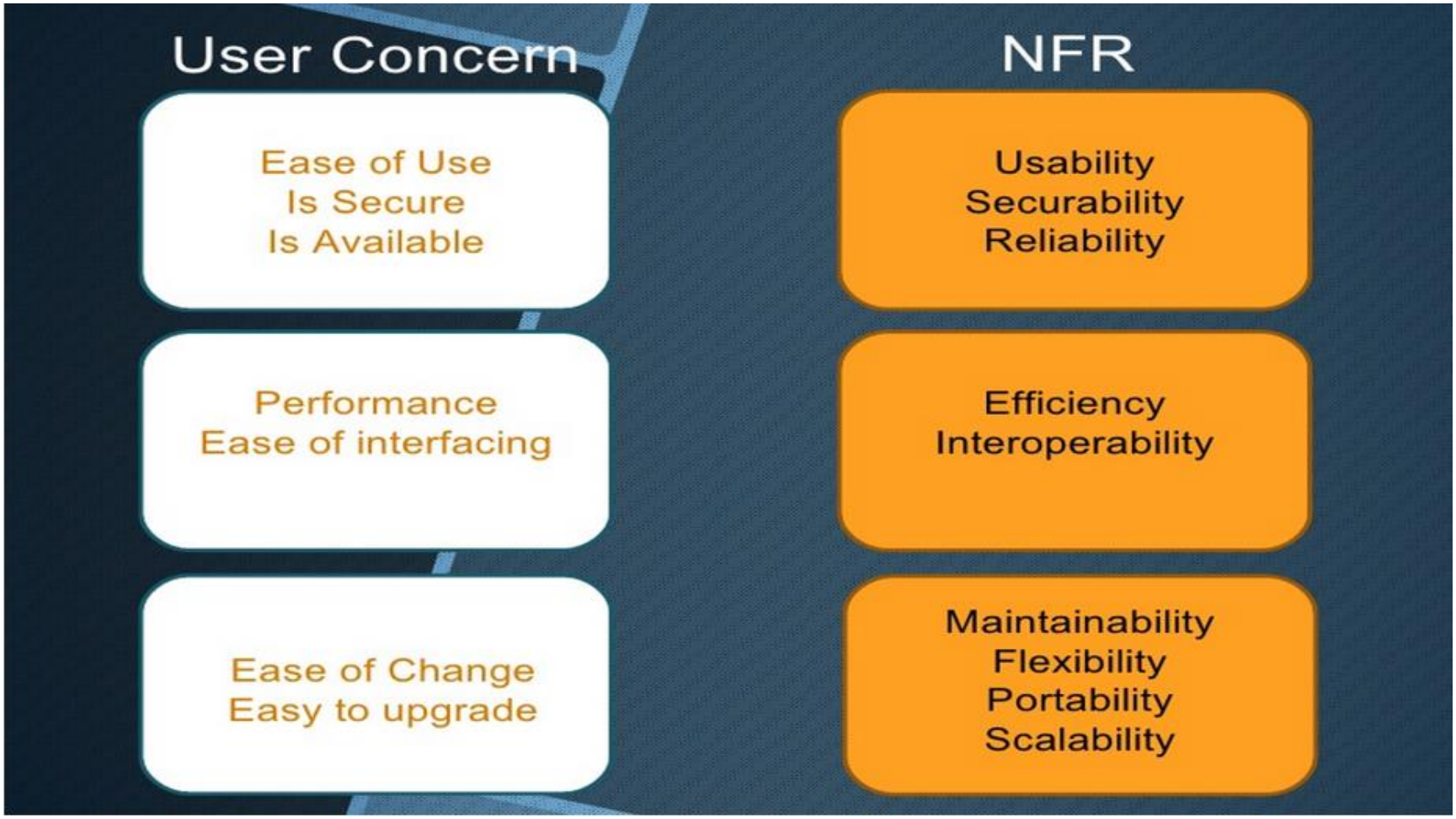
- Non-functional Features:
  - Frequency: how often?
  - Urgency: how quickly application response to user needs
  - Volume: how much data maintain?
  - Accuracy: how precise and timely for data?
  - Usability: what features easy to use by the role?
  - Learnability: how quickly the new user can learn to use application?
  - Flexibility/scalability: how volatile is usage?
  - Reliability: how critical that the app does not fail?

Non-Functional Requirements (cont.)



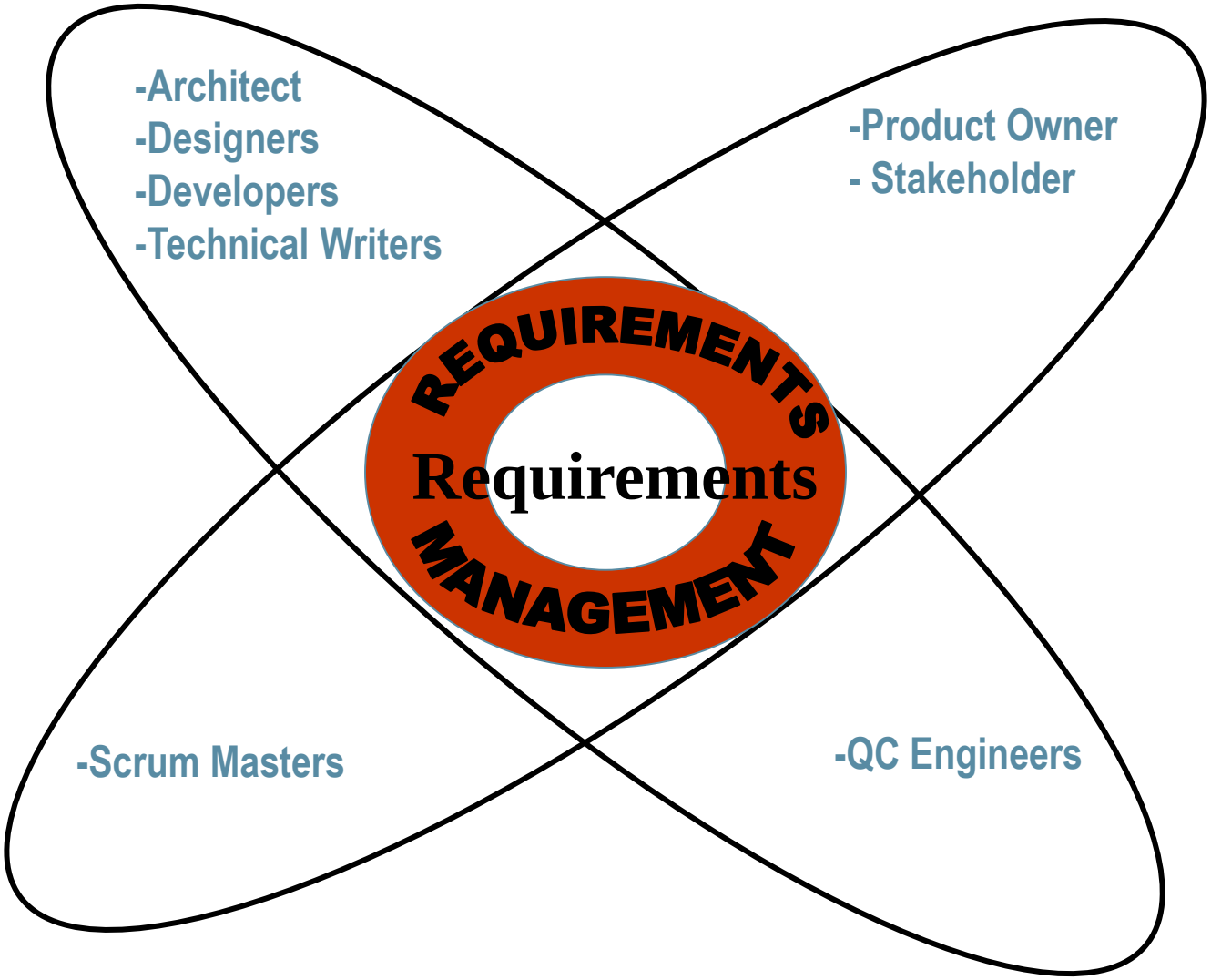
Extended List of Non-functional Requirements

Non-Functional Requirements (cont.)



Mapping Business Concern to Non-Functional Requirements

Who needs to understand requirements?



All project participants need to understand requirements

User Stories

User Stories/ EPIC/ Theme – What are they?

- A template often uses the following type of format:
  - As a <role>, I want <feature> so that <reason>

- **EPIC** – A very large user story that will not fit into a single iteration, does not pass the test for inclusion in an iteration, and will need to be subdivided to be considered

- **Theme** - A collection of features, epics, & stories that describe a broad business purpose

EPIC and Theme - Sample

As a..	I want/would like..	So that..
(Who)	(What)	(Why)
Epic		
sales person	to set my password	I can log into the system
Theme		
a customer	to view my last 100 transactions in under 2 seconds	I can quickly spot inconsistencies
a developer	to pre-aggregate and cache the sales numbers as of the previous day	they do not need to be recalculated each time the report is run

User Stories/ EPIC/ Theme – What are they? (cont.)

- **User Story** is**:**
  - a convenient format for expressing the desired business value
  - crafted in a way that makes them understandable to both business people and technical people
  - used to provide a great placeholder for a conversation
  - written at various levels of granularity and are easy to refine

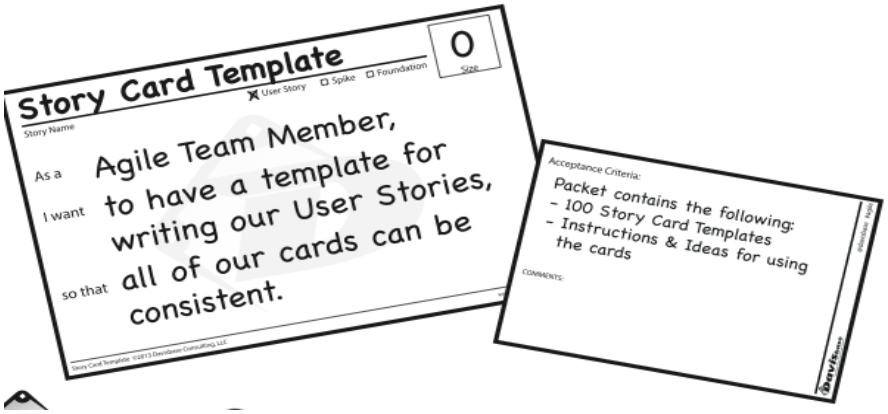
User Story - Sample

As a..	I want/would like..	So that..
(Who)	(What)	(Why)
Functional User Story		
User	to upload photos	I can share photos with others.
User	a trade ticket report that can be viewed on mobile devices	offline users can review the ticket
User	receive boarding pass confirmation email	I can save time at the airport
Non-functional User Story		
the Chief Technology Officer (CTO)	the system to use our existing orders database rather than create a new one	we don't have one more database to maintain.
User	the site to be available 99.999 percent of the time I try to access it	I don't get frustrated and find another site to use.

User Stories

- **3Cs**: Card, Conversation, Confirmation
- **Card**:
  - **Who** – specify User Role
  - **What** – what User Role wants to achieve (the goal)
  - **Why** – why User Role wants to achieve the goal (the benefit)
  - **Acceptance Criteria** - a list of questions, scenarios that enable the User Role to sign off the story as “done”

User Story Template Cards



User Stories (cont.)

- **3Cs**: Card, Conversation, Confirmation
- **Conversation**:
  - Ongoing dialog among Product Owner, Stakeholders, and Development Team during Sprint
  - Enable richer form of exchanging information and collaborating to ensure that the correct requirements are expressed and understood by everyone
  - Supplemented by documents

User Stories

- **3Cs**: Card, Conversation, Confirmation
- **Confirmation**:
  - Confirmation information in the form of conditions of satisfaction or acceptance criteria
  - Used by the development team to between understand what to build and test
  - Used by Product Owner to confirm that a user story has been implemented

User Stories – Template and Sample

- User Story Template
- User Story Sample

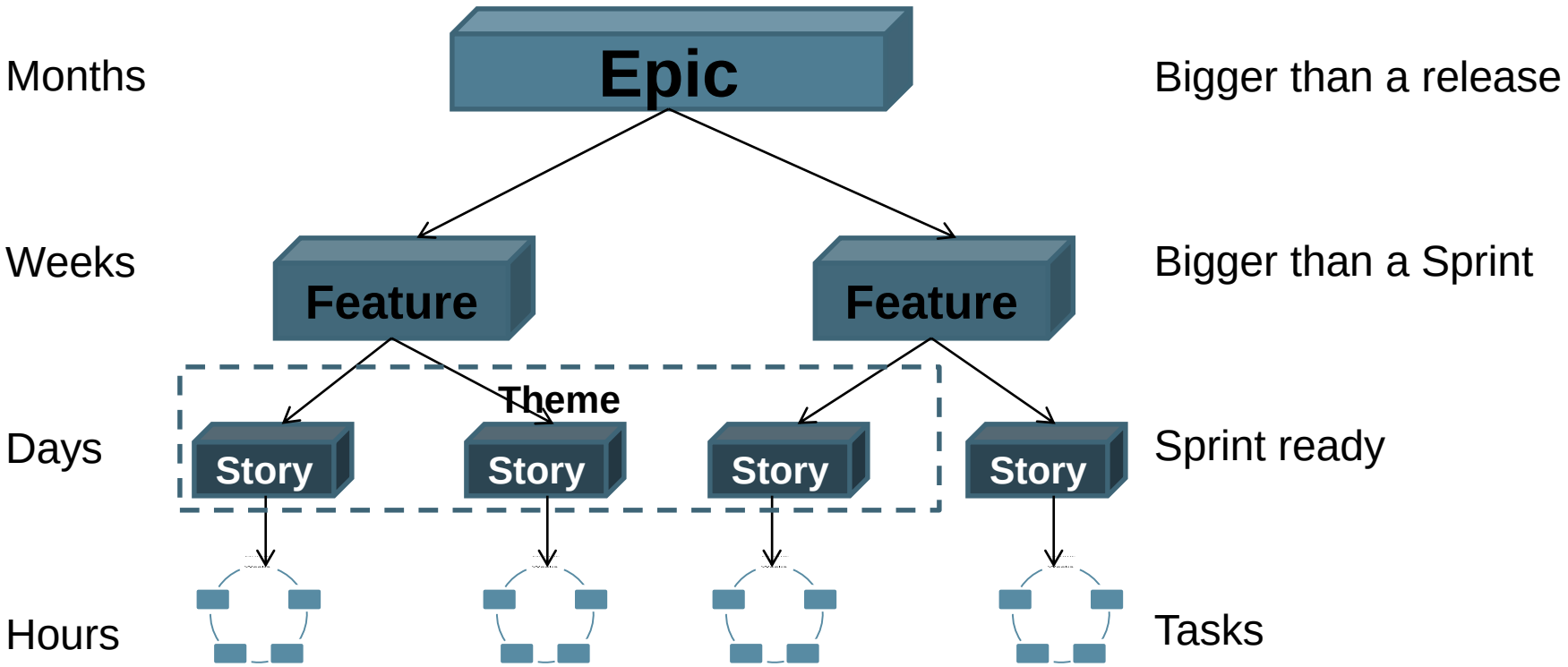
Gathering Stories

- Involve users as part of the team that is determining what to build and is constantly reviewing what is being built

- **Techniques:**
  - **User-Story-Writing Workshop:**
    - Brainstorm desired business value and create user story placeholders for what the product or service is supposed to do
  - **Story Mapping**
    - Decompose high-level user activity into a workflow that can be further decomposed into a set of detailed tasks

User Story

- **Size of User Story**



Split Stories

- When:
  - It is too large to fit into a single iteration or Sprint
  - If a more accurate estimate is necessary
  - If smaller stories have different priorities

- How:
  - Split large stories by the type of data that user could enter
  - Split large stories based on the operations that are performed within the story
  - Split large stories into separate CRUD operations
  - Remove cross-cutting concerns such as securities, logging, error handling, and so on and create two versions of the story: one with and one without support for the cross-cutting concern

Split Stories (cont.)

- How:
  - Split large stories by separating the functional and nonfunctional aspects into separate stories
  - Each of new spitted story should be well within the size the team could complete in a two-week Sprint
- Don't:
  - Split a story into development tasks
- Combine Stories:
  - Combine related stories as that will make it easier to prioritize them, e.g. combine multiple bug reports and treat them as one item

Sample

As a..	I want/would like..	So that..
(Who)	(What)	(Why)
Epic		
sale person	to set my password	I can log into the system
Break to User stories		
an administrator	to send an email to a new salesperson containing a tokenized access link	they can temporarily access the system in order to set their password
sale person	edit my profile	I can set my password
an administrator	to ensure that all sale people's password meet corporate strength requirements	I can harden access to the system

Assessing the Readiness of Stories for An Iteration

Independent

Negotiable

Valuable

Estimable

Small (appropriately sized)

Testable

User Stories

- **Independent:**
  - User stories should be deliverable independently of each other
  - Independent stories enable the team and customer to inject small stories into the backlog that can be delivered in timescales aligned to Sprint
  - User stories that exhibit a high degree of interdependence complicate estimating, prioritizing, and planning
  - Write stories in a way that minimizes dependencies

- **Negotiable**:
  - The details of user stories should be negotiable
  - A story will be refined over time and is negotiable up until the point that the story is planned within a print

User Stories

- **Valuable:**
  - User stories need to be valuable to a customer. This include technical stories
  - Treat technical stories like any other business-valuable story

- **Estimable**:
  - Stories should be estimable by the team that will design, build, and test them
  - Estimates provide an indication of the size and effort and cost of the stories
  - It is essential that the team is involved in the refinement of stories, in cooperation with the customer and stakeholders, to have solid understanding of the story and be able to create realistic and achievable estimates

User Stories

- **Small:**
  - Stories should be sized appropriately – each a few days in size to fit in Sprint

- **Testable**:
  - Being testable means having good acceptance criteria
  - Stories must include testable criteria

INVEST - Sample

Sample:
As a user, I want the system to be fast, so I don't have to waste my time to wait for the page loading

Not INVEST: TESTABLE

Improve:
As a user, I want the web pages should generally load within 2 or 3 seconds, so I can do what I want faster.

INVEST - Sample

Sample:
As a product owner, I want to write game rules, so the player will follow the rule to play game

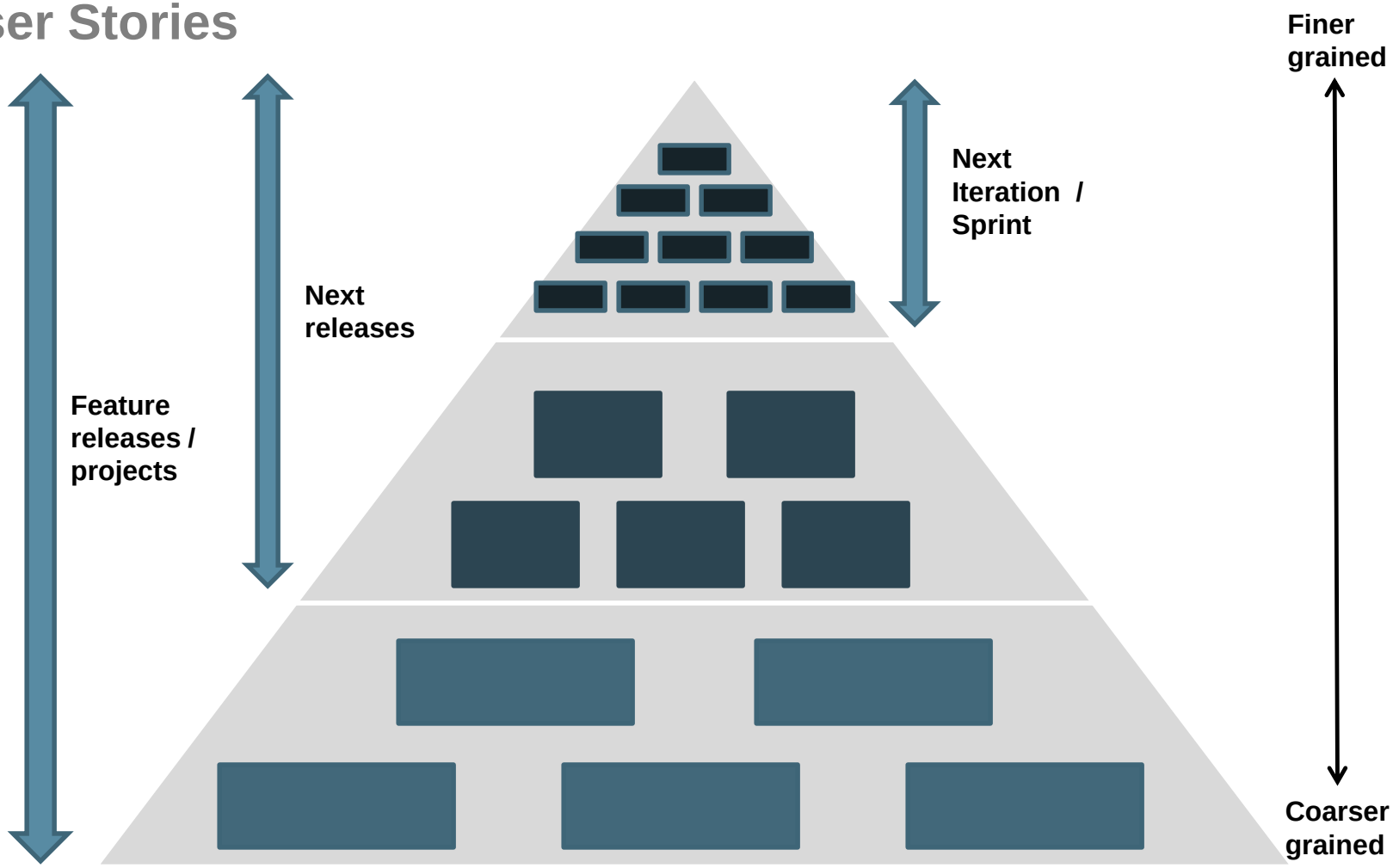
Not INVEST: Independent, Small

Improve:
As a newbie game player, I want to know who goes first so we can start the game

User Stories

- **Backlog refinement:**
  - Stories are continually refined within backlog throughout the whole lifetime of the product
  - Stories should be refined JIT basis for next sprint
- **'Spike' stories:**
  - Is a story that drives technical or functional research effort or investigating work
  - Story-driven activity to investigate something specific
- **Planning pyramid:**
  - A feature breakdown structure of parent-child stories may be required when delivering large, complex projects

User Stories



A planning Pyramid contains both coarse-grained and fine-grained stories; the coarse-grained stories are being refined to be fine-grained stories as delivery progress

Planning Pyramid

User Stories

Prioritization – a MoSCoW acronym

- Arrange stories in a sequence within sprint time-box

- **M**ust have**:**
  - These are stories that must be delivered within sprint time-box

- **S**hould have**:**
  - A story that is very important within a time-box, that will cause significant problems to customer if not delivered

User Stories

- **C**ould have:
  - A story that is very important within a time-box, that may cause some problems to customer if not delivered
- **W**on’t have:
  - Agreed between customer and team that a particular story won’t be delivered “this time”. It might be added to a later time-box or removed completely from PB

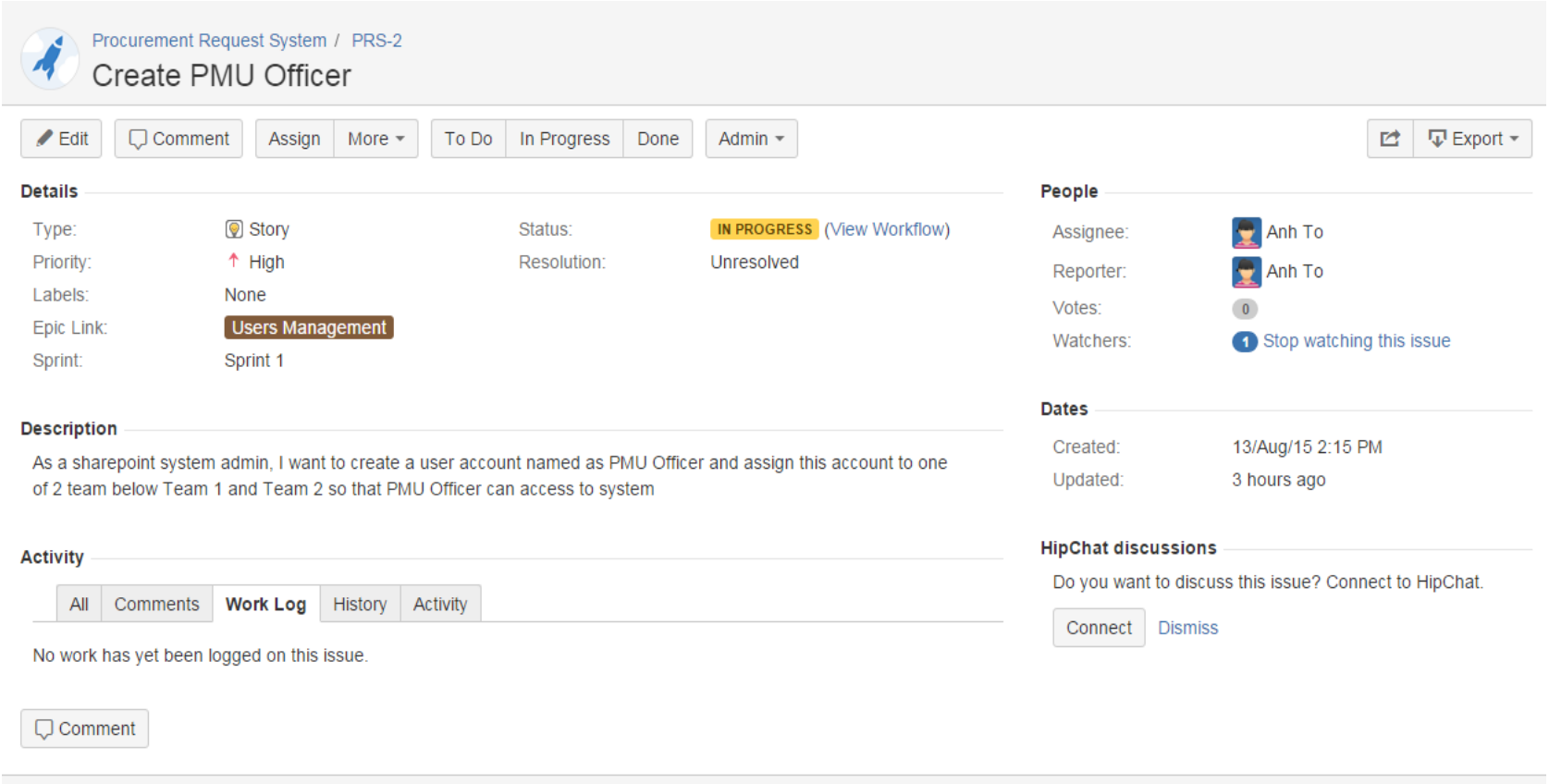
Product Backlog

- A placeholder of requirements and desires from all stakeholders
- A prioritized and emerging list of functional, nonfunctional, architectural, infrastructural, risks elements that required to fulfill the Product Vision
- More granular items kept towards the top, general epics at the bottom
- Product Backlog contents will change over time
- PO is ultimately responsible for the content and state of the Product Backlog, though anyone is able and encouraged to contribute to the Product Backlog
- Each PBI should be small enough to fit into a Sprint and must be clear by specifying the acceptance criteria

Tools in Agile Projects

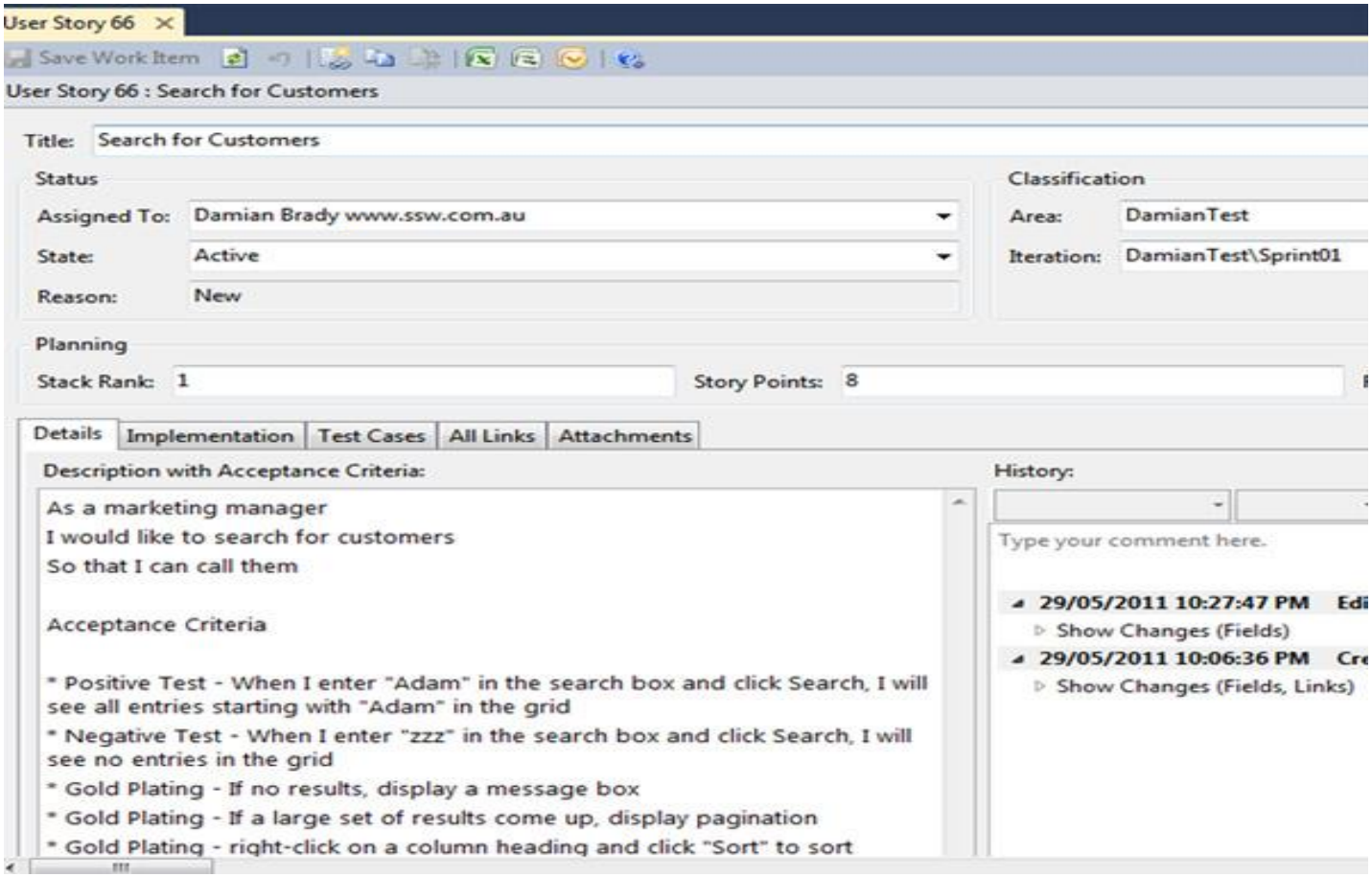
- JIRA Agile
- Team Foundation Server (TFS)

Jira Agile



Screenshot of Jira Agile Tool with User story: Create PMU Officer

Team Foundation Server (TFS)



Screenshot of TFS Tool with User story: Search for Customers

Writing Effective User Stories

Write Effective User Stories

Rule 1:

- Keep your User story Simple:
  - Simple sentence, no compound sentences, no conjunctions (if, and, or, but, uncles, except, without
  - Keep It Simple (KIS)
  - States only 1 thing
  - Can use “and” to combine common characteristics to connect common type of data

Write Effective User Stories

Rule 1 – sample:

- For example: As an applicant, I can navigate to the coverage screen, enter personal and vehicle data, **and** submit the application online to request automobile insurance coverage

- Clearer sample:
  - As an applicant, I can navigate to the coverage screen to select the insurance coverage I need.
  - As an applicant, I can enter personal and vehicle data to compare premiums
  - As an applicant, I can submit the application online to request automobile insurance coverage

Write Effective User Stories

Rule 2:

- Expresses the WHAT, not the HOW to accomplish it:
  - No preconceived
  - Business Result NOT Technology
  - Destination NOT Journey

Write Effective User Stories

Rule 2 - sample:

- For example: As an applicant, I can select my state from a **drop-down box** of abbreviations to avoid entering an invalid state

⇒ Problem: dropdown box is specific technology – solution

- Better example: As an applicant, I can submit a valid state abbreviation to ensure an accurate quote for insurance coverage

⇒ Problem: other option beside dropdown box?: automatic process by using zipcode and web app then returns the states automatically

Write Effective User Stories

Rule 3:

- Writing RELEVANT User Stories
  - Must be in project boundary based on project charter (high level requirement), project scope statement (processes, functions, organizational units, roles/jobs, etc.)
  - Define something about solutions that business need or want
  - Has short tail, no creating cascading effective of changes that exceeds a project authority

Write Effective User Stories

Rule 4:

- Avoid Ambiguity in your User Story
  - Easily Understandable
  - Unambiguous
  - Clear to all target audience

- Causes of ambiguity:
  - It seems so Simple, everyone can get it right the way

Write Effective User Stories

Rule 4 (cont.)

- Need to do:
  - Be the reader not the author
  - Review in different environment (time, place) from the one you wrote statement – desk checking
  - Ask colleague, peer, or manager to rewrite by using the difference word except for articles (a, an, the), prepositions, pronouns, conjunctions
  - Review with different gender, job. Different gender/jobs title thinks differently
  - Think outside the box

Write Effective User Stories

Rule 4 - sample:

- For example: As a **telephone operator,** I **can complete** at least **12 reservations** per hour during **peak volume** to **reduce the wait times for customers**

⇒ Colleague rewrite: As a reservationist, I am able to process a minimum of a dozen request for travel accommodations within 60 minutes during the busiest time of the year to minimize dropped calls

- Revised Statement: As a reservationist, I can complete at least 12 non-holiday reservations per hour during daily peak times to reduce dropped calls caused by long wait times

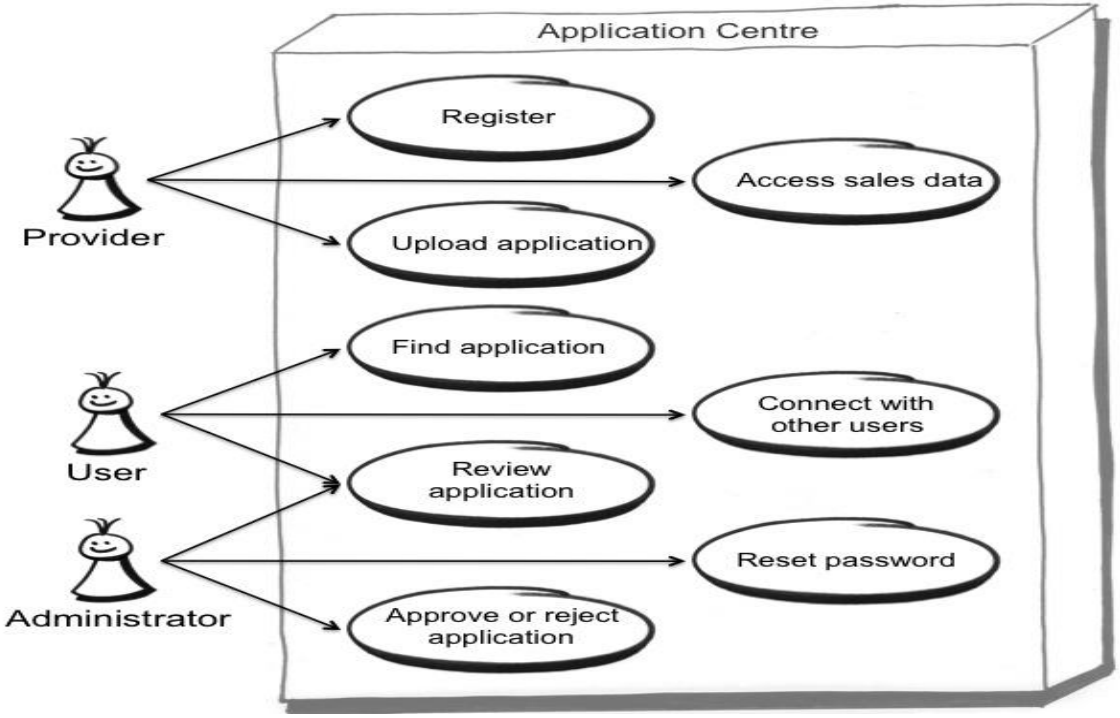
User Stories Modelling

- User stories are not well suited to express the relationships between different features and to describe workflows

- Main Sections:
  - Context Diagram
  - Activity Diagram

Context Diagram with Epics

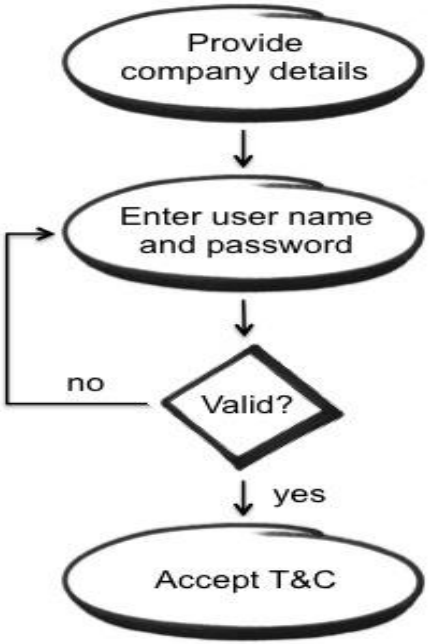
- A context diagram that depicts user roles and epics, large and coarse-grained stories, is great to provide an overview of the product's functionality



Context Diagram of Application Centre

Activity Diagram with Stories

- Great at capturing sequences and workflows by connecting individual user stories. They also support the creation of complex test scenarios that go beyond a single story



Activity Diagram of Application Registration

User Story Modelling Tips

- Model collaboratively
- Focus
- Keep it simple
- Use whiteboards and flip charts rather than electronic tools

Other Requirement Artifacts

Problems of User Stories

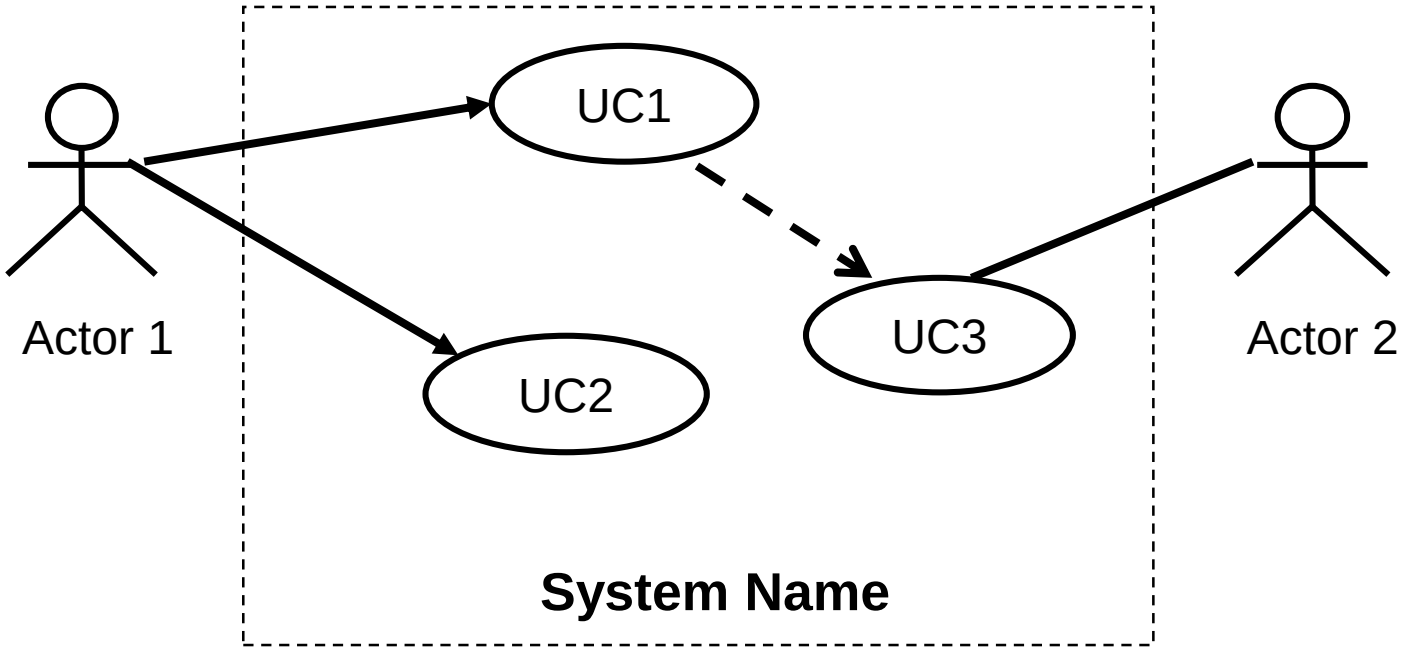
- No account of the needs and behaviors of real users
- Were that not indictment enough, suffer from demonstrable flaws in structure and are often written by the wrong people at the wrong time
- Words mean different things to different people. Even meticulously written user stories that follow a standard form leave plenty of room for interpretation
- Also fall into the all too common trap of defining a solution (what to do) instead of presenting problems

Use Case Model

- What is Use Case Model?
  - A high level view of the system from User's perspective

- Main Sections:
  - Introduction
  - Actors
  - Use Cases' brief descriptions

Use Case Model (cont.)



Use Case Model Diagram

Use Case

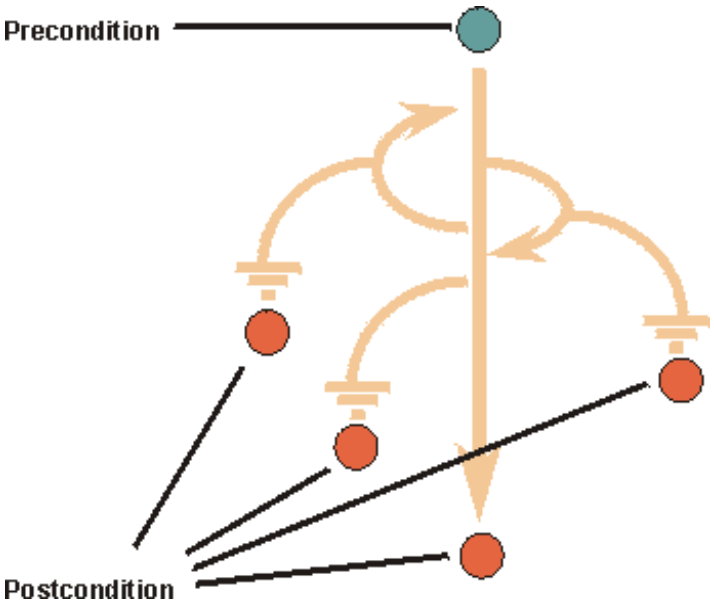
- Main Sections:
  - Pre-Conditions
  - Post-Conditions
  - Flow of Events
    - Basic Flow (only one)
    - Alternative Flows (one or many)
    - Exception Flows (one or many)
  - Business Rules
  - Special Requirements
  - Supplementary Information

Use Case - Pre and Post Conditions

- Pre-Conditions:
  - The state of the system and its surroundings that is required before the use case can be started
  - NOT the event that starts the use case

- Post-Conditions:
  - A list of possible states the system can be in immediately after a use case has ended

- Pre and Post conditions must be observable to actors



Use Case – Flow of Events

- One basic flow:
  - "Happy Path"
- Many Alternative Flows:
  - "**Regular Variants**"
  - "**Odd Cases**"
- Many Exceptional Flows
  - "**Errors**"

Use Case – Other Sections

- Business Rules:
  - Definitions, rules or specifications of the business that explain System Responses in a use case transaction

- Special Requirements:
  - Requirements about this use case but not covered in previous sections
  - Usually non-functional requirements specific for this use case

- Supplementary Information
  - Additional materials that pertain to this use case
    - Field Tables
    - Message Logs

- Use Case Sample

Functional Specifications – What is it?

- A functional specification focuses on what various outside agents (people using the program, computer peripherals, or other computers, for example) might "observe" when interacting with the system. A typical functional specification might state the following:
  - *When the user clicks the OK button, the dialog is closed and the focus is returned to the main window in the state it was in before this dialog was displayed*

- It does not define the inner workings of the proposed system; it does not include the specification how the system function will be implemented

Functional Specifications – Sample

- Functional Specifications usually contains following information:
  - GUI screen
  - Description about behaviors of screen elements

Requirement Specifications – What is it?

- Requirement Specifications so- called Software Requirement Specifications (SRS)
- SRS captures complete software requirements for the system, or a portion of the system.
- SRS fully describes the external behavior of the application or subsystem identified.
- SRS also contains nonfunctional requirements, design constraints and other factors necessary to provide a complete and comprehensive description of the requirements for the software

Requirement Specifications – Sample

- SRS usually contains following information
  - Project Scope
  - Business Process
  - Business Rules
  - Functional Requirements
  - Non-Functional Requirements

Points to Remember

Summary

- What is Functional Requirements? Non-Functional Requirements?
- What is User Story?
- What does INVEST stand for? Meaning of each?
- How can we write the effective user story?
- When to split a User Story?
- What does it mean by splitting a User Story across Data and Operational boundaries?
- What are cross-cutting concerns?
- List non-functional requirements?

Q&A

Thank You

Revision History

Date	Version	Description	Updated by	Reviewed and Approved By
30-Sep-2015	1.0	Initial	Anh Truong Thy Vo Anh To	Khanh Lam Quang Tran

CSC

BUSINESS SOLUTIONS
TECHNOLOGY
OUTSOURCING