



**Essence  
Software Engineering Essentialized**

# **Essence** ***Software Engineering Essentialized*** **Part 3C – Running with Use Case Lite**

Giuseppe Calavaro, Ph.D.  
IBM Big Data Practice Leader  
External Professor at University of Rome “Tor Vergata”

[www.semat.org](http://www.semat.org)

# Agenda of this Teaching Module

- Use Case Introduction
- Use Case Lite Practice using Essence example
  - Use Case Lite Alphas
  - Use Case Lite Products
- Kick-Starting using Use Case Lite Practice on our project
- Use Cases vs Use Case Slices
- Visualizing the impact of UC for the team
- Progress and Health of UC Slices

# Module Objectives

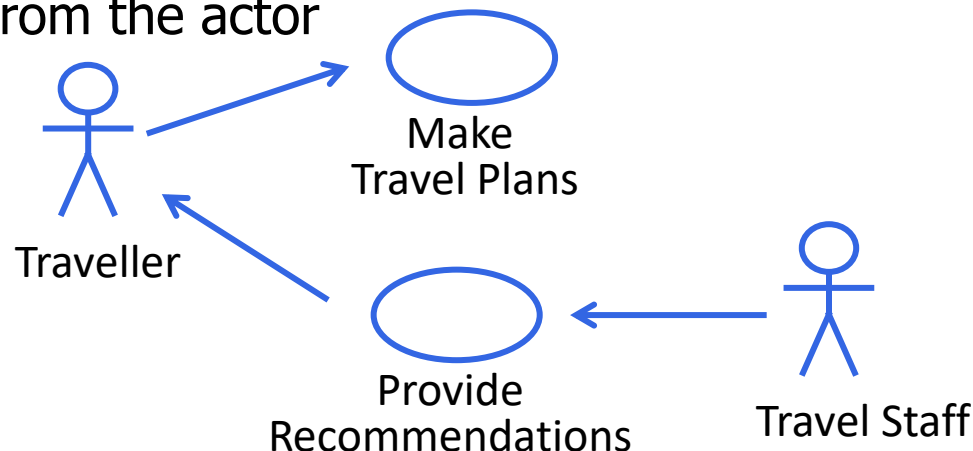
- To understand how a project team could tailor and use an essentialized practice by using a Use Case Lite practices customized for this example
  - We will see why and how Smith's team started applying use cases in their development at Travel Essence.
- The Use Case practice is a requirements analysis technique that has been widely used in modern software engineering since its introduction by Ivar Jacobson in 1987.
  - Use cases can help teams understand the bigger picture and how product backlog items are related.
- The next slides briefly introduce Use Cases leaving the reader to study other literatures for a complete and detailed presentations.

# Use Case Explained

- A use case is all the ways of using a system to achieve a particular goal for a particular user
- How do they compare with User Stories?
  - User stories represent stories (each one is a single scenario or example of story) of using the system
  - Use Stories help flesh out missing requirements by encouraging informal discussion between developers and users, yet:
    - Many User stories ... sometime are too many
    - It is not clear how these user stories make up something complete
    - Lack of structure
  - Epics can help consolidate many stories and provide structure
- Use cases give requirements a structure, or a systematic way to organize requirements
  - This structure makes it easier for teams to conduct analysis, user interface (UI) design, service design, implementation, tests and so on

# Use Case Models

- In the Unified Modeling Language (UML) the relationships between users and use cases are represented in what is referred to as a Use-Case Model
- Use cases include the actual functionality and behaviour of the system.
  - Each use case is described in a Use-Case Narrative.
  - A use case narrative provides a textual description of the sequence of interactions between the actor and the system.
  - It also describes what the system does as a response to each message from the actor



# Use Cases narrative

- The use case narrative is usually separated into two parts referred to as the basic flow and the alternate flows.
- The **basic flow** describes a normal or a “basic” use of the described use case often called the “happy day scenario”.
  - The basic flow is worded in a way you would test and verify the behaviour of the functionality.
  - It is a sequence of steps you would expect when using or testing the system.
- The **alternate flows** are variations of the basic flow to deal with more specific cases
  - These variations can be enhancements, special cases, etc.
  - There are multiple alternate flows

# Use Case Narrative Examples

## UC Make Travel Plans

- **Basic Flow:**
  1. Traveller provides travel details (travel dates and destination)
  2. System searches and displays hotels for the travel dates and destination.
  3. Traveller selects a hotel and room type.
  4. System computes and display.
  5. System makes a tentative reservation for the traveler on the selected hotel.
- **Alternate Flows:**
  - A1. Travel plan with multiple destinations
  - A2. Travel plan having a single destination but non-consecutive dates
  - A3. Travel plan with non-consecutive dates and multiple destinations

## UC Provide Travel Recommendations

- **Basic Flow:**
  1. Traveller verifies travel details (travel dates and destination)
  2. Traveller requests recommendations
  3. System provides list of recommendations
  4. Traveller browse recommendations
  5. Traveller selects and view recommendation.
- **Alternate Flows:**
  - A1. Recommendations of different entities
    - a. Hotel, b. Place of Interest
  - A2. Recommendations
    - Recommendations based on (a) popularity rating, (b) on pricing,
  - A3. Recommendation request trigger
    - (a) User initiated, (b) System triggered
  - A4. Sorting of recommendations
    - (a) Sorting based on prices

# Use Case Considerations

1. The use cases help you see the big picture through the use-case model
2. The use case approach provides structure through the separation of basic and alternate flows
  - This structure also makes the requirements easier to understand, especially on endeavours that are large and complex.
3. A use case often contains too much functionality to be developed in one iteration, such as a single sprint when using Scrum.
  - That is why a use case is split up into a number of intelligently selected smaller parts that are referred to as use-case slices.
  - These use-case slices taken together represent the whole use case and when all the use-case slices are described, the whole use case is described



# Use Case Slices

- A **use-case slice** is a slice through a use case that is meaningful to describe, design, implement and test in one go.
  - It doesn't need to by itself give value to a user, but together with all other slices of the same use case, the value is achieved.
  - For example, the basic flow of a use case is a good candidate to become an early use-case slice.
  - Additional slices can then be added to complete the whole use case later.
- The slicing mechanism enables you to create slices as big or small as you need **to drive your development**.
  - The use-case slices include more than just the requirements.
  - They also slice through all the other aspects of the system (e.g. user experience (user interface), architecture, design, code, test) and its documentation.

# Disclaimer on this Practice

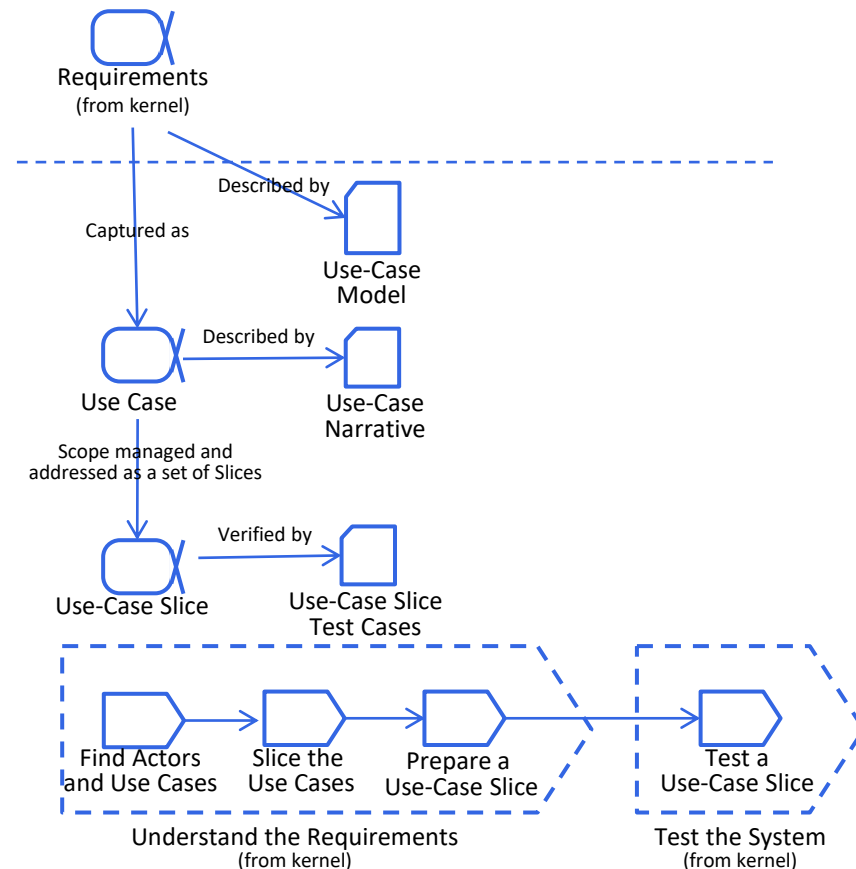
- The Use-Case Lite practice that we discuss provides a scalable, agile practice that uses use cases to capture the functionality of a software system and test them to ensure the system fulfils them
  - Use cases provide an approach for putting product backlog items into context from the user's point of view
- It is not our intention in this book to explain why one practice is better than other practices
  - All of the practices we present in this book are example practices.
- We want the reader to understand:
  - How they can be represented in an essentialized form
  - What value essentialization brings to comparing practices

# Use Case Lite Essentialized Practice

- How to describe the Use Case Lite Practice using Essence?
- The first questions we always ask when essentializing a practice are:
  - What are the things you need to work with?
  - What are the activities you do?

# Use Case Lite Practice expressed in Essence Language

- **Requirements** are decomposed into **Use Cases**, which in turn are broken down into **Use-Case Slices**
  - All three are important things we need to work with and progress
  - Alphas represents important things we need to progress
- A Use-Case Model is a tangible description of the Requirements and therefore it is a work product.
- Each use case, within our Use-Case Lite practice, has two related work products:
- 1) Use-Case Narrative
  - Singleton
- 2) Use-Case Slice Test Case.
  - Exists in many instances



# Activities and Use Case Slices

There are four activities in our Use-Case Lite practice, namely:

1. **Find Actors and Use Cases** to gain an overall understanding of what the system is about.
2. **Slice the Use Cases** to break them up into a number of intelligently selected smaller slices that each fit within a single sprint. Such a slice is a Use-Case Slice.
3. **Prepare a Use-Case Slice** by enhancing the narrative and test cases to clearly define what it means to successfully implement the slice.
4. **Test a Use-Case Slice** to verify it is done and ready for inclusion in a release

- Use-case **slices are identified** by working through their use case to identify paths, scenarios or as we say the stories that build up the use case.
- Typically a **story is any path that you may want to follow** going through the use case its basic flow or its alternative flows.
- The **story idea is similar to the user story idea in the User Story practice** and is very important to find good use-case slices.
- A **use-case slice typically includes one or more stories.**

# Elements of the Use Case Lite Practice

Element	Type	Description
Use Case	Alpha	All the ways of using a system to achieve a particular goal for a particular user
Use-Case Narrative	Work Product	Tells the story of how the system and its actors work together to achieve a particular goal.
Use-Case Slice	Alpha	One or more stories selected from a use case to form a work item that is of clear value to the customer
Use Case Model	Work Product	A model that captures and visualizes all of the useful ways to use a system
Use-Case Slice Test Case	Work Product	Defines inputs and expected results to help evaluate whether a system works correctly. There can be one or more Test Cases to verify each Use-Case Slice.

Element	Type	Description
Find Actors and Use Cases	Activity	Agree on the goals and value of the system by identifying ways of using and testing it.
Slice the Use-Cases	Activity	Break use case up into a number of intelligently selected smaller parts for development.
Prepare a Use-Case Slice	Activity	Enhance the narrative and test cases to clearly define what it means to successfully implement the slice
Test a Use-Case Slice	Activity	Verify the slice is done and ready for inclusion in a release

# Use Case Lite Alphas

- The Alphas cards shows the short descriptions and the stated for this practice Alphas
- While the Requirements Alpha is part of the kernel, Use Case and Use-Case Slices are defined in this Practice



## Requirements

What the software system must do to address the opportunity and satisfy the stakeholders.

Conceived

Bounded

Coherent

Acceptable

Addressed

Fulfilled



Generated by UI Practice Workbench™

5.2.0



## Use Case

All the ways of using a system to achieve a particular goal for a particular user.

Goal Established

Story Structure Understood

Simplest Story Fulfilled

Sufficient Stories Fulfilled

All Stories Fulfilled

Relates to:  Requirements



Generated by UI Practice Workbench™

5.2.0



## Use-Case Slice

One or more stories selected from a use case to form a work item that is of clear value to the customer.

Scoped

Prepared

Analyzed

Implemented

Verified

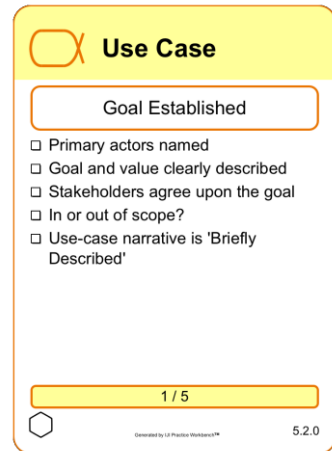
Relates to:  Use Case



Generated by UI Practice Workbench™

5.2.0

# Use Case Alpha state cards



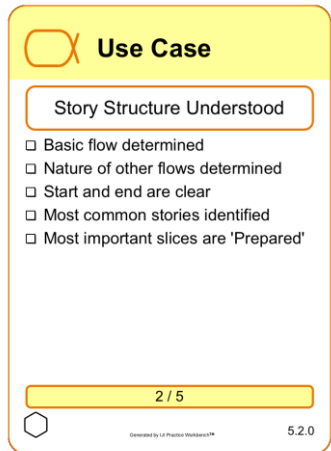
**Use Case**

Goal Established

- ☐ Primary actors named
- ☐ Goal and value clearly described
- ☐ Stakeholders agree upon the goal
- ☐ In or out of scope?
- ☐ Use-case narrative is 'Briefly Described'

1 / 5

5.2.0




**Use Case**

Story Structure Understood

- ☐ Basic flow determined
- ☐ Nature of other flows determined
- ☐ Start and end are clear
- ☐ Most common stories identified
- ☐ Most important slices are 'Prepared'

2 / 5

5.2.0



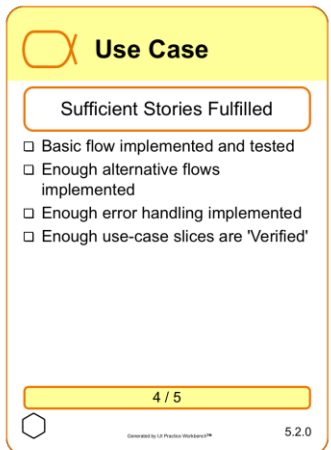
**Use Case**

Simplest Story Fulfilled

- ☐ Applicable use-case slice is 'Verified'

3 / 5

5.2.0



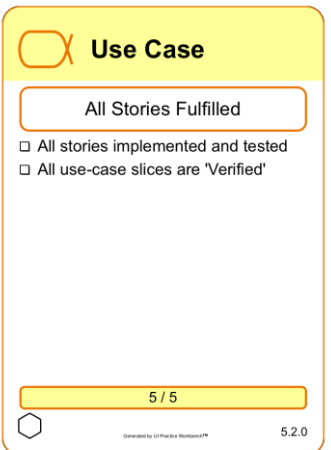
**Use Case**

Sufficient Stories Fulfilled

- ☐ Basic flow implemented and tested
- ☐ Enough alternative flows implemented
- ☐ Enough error handling implemented
- ☐ Enough use-case slices are 'Verified'

4 / 5

5.2.0



**Use Case**

All Stories Fulfilled

- ☐ All stories implemented and tested
- ☐ All use-case slices are 'Verified'


5 / 5

5.2.0

- **Goal Established** – The scope of a use case is defined (what the actor wants to achieve)
- **Story Structure Understood** – One of the key benefits of use cases is that it provides a structure.
- **Simplest Story Fulfilled** – Once this code skeleton is formed and stabilized, it becomes easy to implement the rest of the stories.
- **Sufficient Stories Fulfilled** – Once sufficient stories are fulfilled, the use case can be evaluated if it achieves the use case goal well.
- **All Stories Fulfilled** – Finally, the entire use case is completed



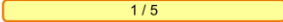
# Use-Case Slice Alpha State Cards


 **Use-Case Slice**


**Scoped**

☐ Stories have been identified  
☐ Requirements covered are clear  
☐ Relative priority is known  
☐ Implementation work has been estimated

1 / 5



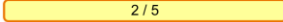
 Generated by UI Practice Workbench™ 5.2.0


 **Use-Case Slice**


**Prepared**

☐ In-scope requirements are agreed  
☐ Test cases are sufficient for verification  
☐ Tests are well-defined

2 / 5



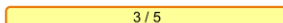
 Generated by UI Practice Workbench™ 5.2.0


 **Use-Case Slice**


**Analyzed**

☐ Enough information to successfully implement  
☐ Implementation impact is agreed  
☐ Impact is acceptable  
☐ Use-case realizations are 'Implementation Elements Identified'

3 / 5



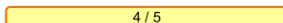
 Generated by UI Practice Workbench™ 5.2.0


 **Use-Case Slice**


**Implemented**

☐ Code fulfils the slice  
☐ Implementation elements are verified

4 / 5



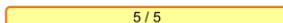
 Generated by UI Practice Workbench™ 5.2.0


 **Use-Case Slice**

**Verified**

☐ Tests have executed successfully  
☐ Software system passes sufficient test cases  
☐ Test cases added to regression suite

5 / 5



 Generated by UI Practice Workbench™ 5.2.0

- **Scoped** – At this state, the use case slice has been identified and its scope clarified.
- **Prepared** – the information the development team needs to implement the use case slice are available, including priorities relative other slices, estimates of cost to implement, dependencies to other use-case slices, etc.
- **Analyzed** – At this state, the development team has a common agreement on how the use-case slice will be implemented. This includes agreeing on things like user interfaces, persistence, and so on.
- **Implemented** – At this state, the use-case slice is implemented. This involves writing actual code.
- **Verified** – At this state, product owners verifies the use-case slice does what it is expected.

# Use Case Model Product

A use-case model describes not just one but several use cases and how together they provide value to its users (i.e. actors).

Levels of detail:

- **Value Established** – the value of the use cases and hence the use-case model is established. Readers of the use-case model have a good understanding of what the use cases are about, what they do and how actors benefit from them.
- **System Boundary Established** – the scope and boundaries of the Requirements are described. The team and stakeholders have a clear understanding of what is within or out of scope
- **Structured** – The use-case model is well-structured. There is little or no overlap between use cases. The dependencies and relationships between use case are described clearly



## Use-Case Model

A model that captures and visualizes all of the useful ways to use a system.

Value Established

System Boundary Established

Structured

Describes:  Requirements



Generated by IJL Practice Workbench™

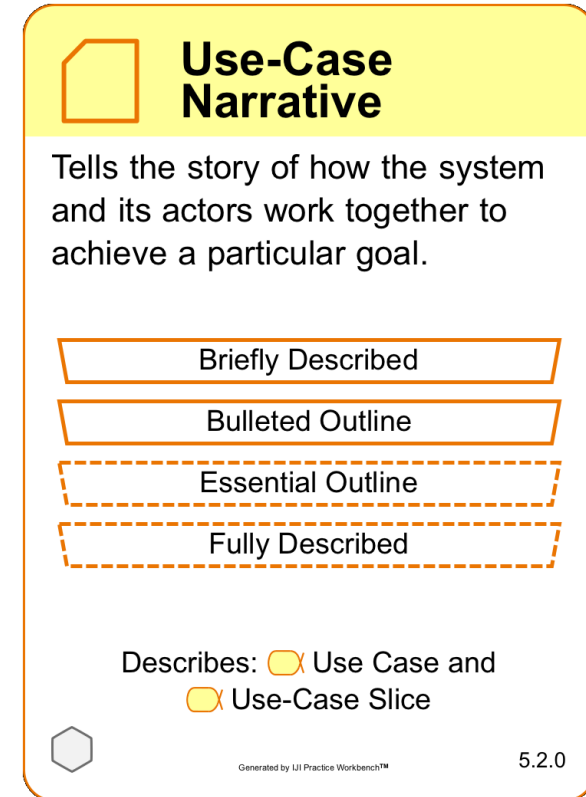
5.2.0

# Use Case Narrative Product

- A use-case narrative describes the story (i.e. sequence of steps) of how the system and the actors work together to achieve a particular goal

Levels of detail:

- **Briefly Described** – At this level of detail, the use-case narrative only has a brief description of the use-case goal and some words about what it is about
- **Bulleted Outline** – the story of how the system and actors work together is available
- **Essential Outline** – the story is full blown. In the context of requirements of the software system, the various alternative usages and exceptions to be handled are clearly described.
- **Fully Described** – This is a very detailed description of the use-case. All conversations are clearly spelt out



# Use Case Model Product

The Use-Case Slice Test Case work product defines the inputs and expected outputs to help evaluate whether a use-case slice is implemented correctly

Levels of detail:

- **Scenario Chosen** – At this level of detail, the different scenarios required to test the use-case slice are described.
  - This includes the normal way of using the use-case slice and other variations (alternative usages and exception cases).
- **Variables Identified** – the different variables are listed.
  - For example, in the variables for testing the Make Travel Plans use case include Traveler identification, destinations, and recommendations popularity ratings.
- **Variables Set** – At this level of detail, the actual variables are defined.
  - For example, the Traveler might be Sam, whose identification is 12345678. The destination is Singapore. The popularity rating of the Singapore Zoological Gardens, Shangri-La Hotel, are set.
- **Scripted or Automated** – the test cases are clearly described such that a person can run the test case by following a step by step procedure without misinterpretation or a software tool can execute it repeatedly with pass/fail results clearly defined.



## Use-Case Slice Test Case

Defines test inputs and expected results to help evaluate whether a Use-Case Slice works correctly.

Test Scenarios Chosen

Variables Identified

Test Variables Set

Scripted or Automated

Describes:  Use-Case Slice

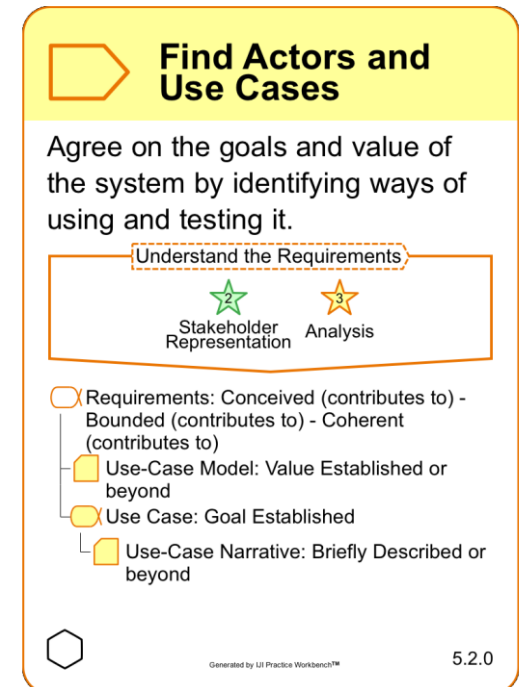


Generated by UI Practice Workbench™

5.2.0

# Find Actors and Use Cases Activity

- The Find Actors and Use Cases activity is about agreeing on the goals and value of the Software System by identifying the different ways of using it.
  - The card also indicates that:
    - the use case model needs at a minimum to achieve the **Value Established** level of detail
    - the use case narrative at a minimum must be **Briefly Described**.
    - The use case alpha needs to achieve the Goal **Established** state



# Use Case Model and Narrative Example



## Use Case: **Provide Travel Recommendations**

### Basic Flow:

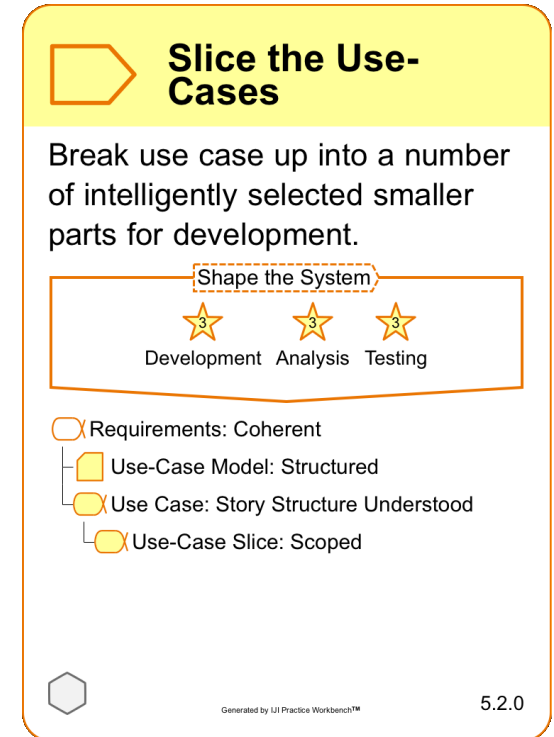
1. Traveller provides travel details (travel dates and destination)
2. Traveller requests recommendations
3. System provides list of recommendations
4. Traveller browse recommendations
5. Traveller selects and view recommendation.

### Alternate Flows:

1. Recommendations of different entities
  - a. Hotel, b. Place of Interest
2. Recommendation computation
  - a. Recommendations based on popularity rating
  - b. Recommendations based on pricing
  - c. **#New** Recommendations based on advertisements
  - d. **#New** Recommendations based on favorites
  - e. **#Updated** Weighting function for the above parameters (popularity, pricing)
3. Recommendation request trigger
  - a. User initiated, b. System triggered
4. Sorting of recommendations
  - a. Sorting based on prices, b. **#New** Sorting based on vicinity
5. **#New** Recommendation actions
  - a. **#New** Add selected recommendations to favorites

# Slice the Use-Cases Activity

- Slice the Use Cases means to break it up into smaller parts to facilitate development
  - Several alternative paths can be in a single slice.
- After we implement our chosen slices for each sprint we will need to verify that each one is done and is ready for inclusion in our next Release





# Example of Use Case Slices

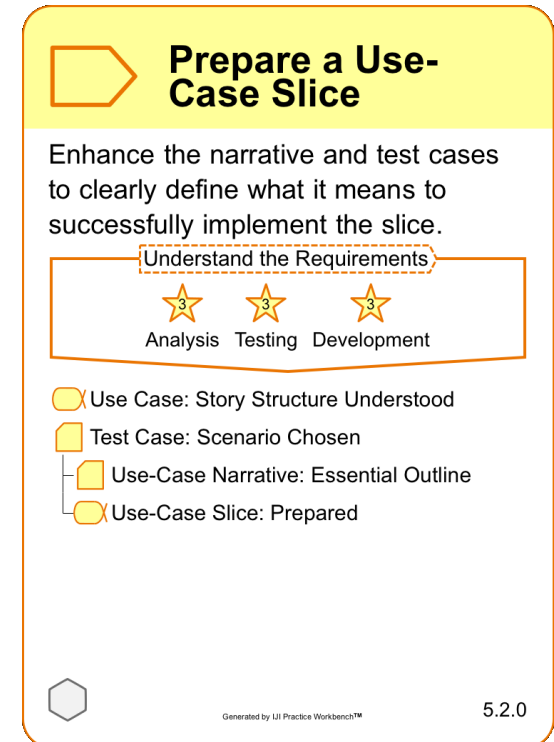
Given the use case “Provide Travel Recommendations” the possible initial slices could be:

Use-Case Slice Name	Use-Case Slice Description
Recommendation By Advertisements	<b>#New</b> Recommendations based on advertisements <b>#Updated</b> Weighting function for the above parameters
Sorting by Vicinity	<b>#New</b> Sorting based on vicinity
Handle Favorites	<b>#New</b> Add selected recommendations to favorites <b>#New</b> Recommendations based on favorites <b>#Updated</b> Weighting function for the above parameters



# Prepare a Use-Case Slice Activity

- The Prepare a Use Case Slice activity enhances the use-case narrative and the use-case slice test cases to clearly define what it means to successfully implement the use-case slice.
- This is an activity performed for each slice
  - For brevity, we will show in next slides only one use-case slice: Handle Favorites.



# Preparing “Handle Favorites” slice

- Favorites are just a list, which the application stores.
- If a user determines that a recommendation is useful for him/her, he might want to store this recommendation in his favorites list.
- This favorite list also acts as an input to the recommendation engine.

Use-Case Slice Instance	Use-Case Test Cases
# <b>New</b> Add selected recommendations to favorites	1. New Favorite 2. Favorite already exists 3. Maximum number of favorites
# <b>New</b> Recommendations based on favorites	1. No Favorites 2. One Favorite within vicinity of traveller destination 3. One Favorite outside vicinity of traveller destination
# <b>Updated</b> Weighting function for the above parameters	1. Weightage of favorites set to zero 2. Weightage of favorites set to 0.5

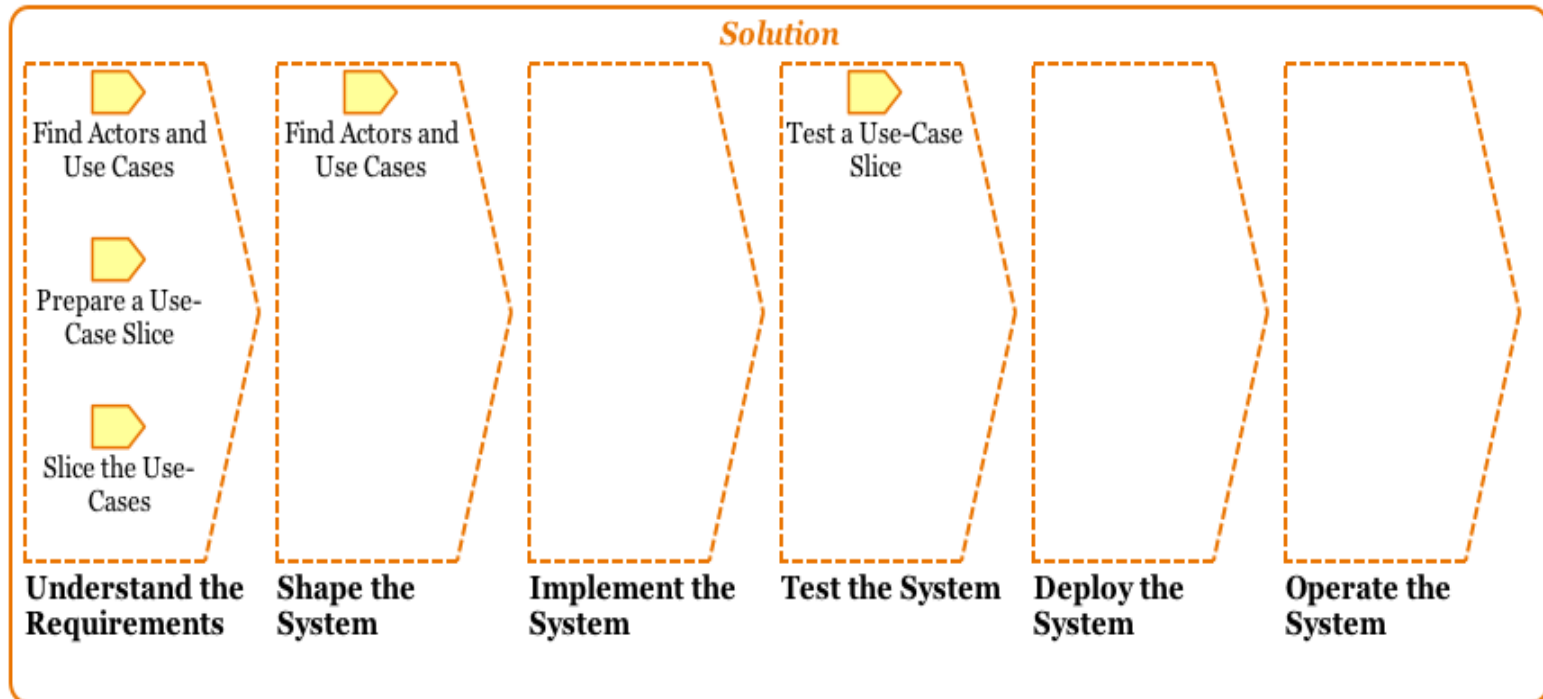
# Test Use-Case Slice Activity

- The goal of the Test a Use-Case Slice activity is to verify that the slice is done and ready for inclusion in a release
- The Use-Case Slice test cases chosen in previous slides are input to this activity.
- During testing, these test cases are refined further with additional details to make sure that they are repeatable



# Impact of use case for the team

- Use cases help the team find they could see the big picture of the system better through their use case diagrams and use case model
- This is made visible by looking at the Use Case Lite's coverage of the solution activity spaces



# Monitoring the project progress

- Teams work with multiple use-case slices from multiple use cases at any point in time.
- They complete use-case slices within each sprint
  - i.e. drive them to the Use-Case Slice Verified state.
- While individual use-case slices are completed in each sprint, often it requires multiple sprints to complete a full use case.
- How to monitor the progress and health of Use Cases and Use Cases Slices?
  - **The alpha state cards for Use Case and Use-Case Slices provide a tool for this purpose**

# Example Project and the other practices

**TravelEssence** team has chosen multiple practices:

Therefore, they have a number of alphas to juggle

From the **Scrum Lite practice**

- **Sprint** – focusing on the goals for the Sprint
- **Product Backlog Item** – a change to be made to the product in a future release

From the **Use-Case Lite practice**

- **Use-Case Slices** – the UC Slices to be Verified by the end of the Sprint
- **Use Cases** – they need not be completed for each sprint, but they are useful for determining which use-case slices should be implemented first.
  - Thus, different use cases will be at different states at the end of each sprint.

From the **Essence kernel**

- **Work** – the team needs to maintain the Under Control state as development progresses.
- **Requirements** – the Requirements alpha progresses towards Addressed or Fulfilled depending on the goals of the sprint