LO4 – Prepare Project Analysis

At this stage UML (Unified Modeling Language) diagrams are used to record (some) of the results of the analysis.

A common mistake at this point is to assume that the point of these exercises and learning steps is to create diagrams. In fact, the most important thing to remember is that the goal of this phase IS NOT to develop UML diagrams, it's to figure out what the client wants the system to do (their requirements).

LO 4.1 - Use Case Diagrams

Actors can be used to drive conversation around requirements. You look at the features of the system each actor will use. To document this we use a Use Case Diagram.

Use Case Diagram

* Part of UML
* Purpose:
  + Graphical representation of the users/actors of the system
  + Graphical representation of the actions those users perform, or the functionality they rely on.
  + Representation of the hierarchy of those features and requirements.

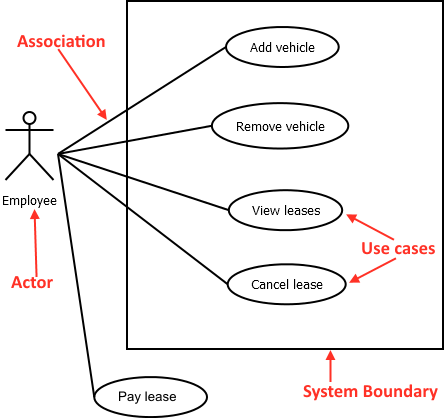
**Use Case Components**

There's a good description of use case diagrams at the Agile Modeling website: <http://www.agilemodeling.com/artifacts/useCaseDiagram.htm>.

Use case diagrams are very simple diagrams. They contain:

* actors: (represented by stick figures)represent entities performing some action using the system.
* use cases: (represented by ovals) represent the functions the actors perform using the system.
* associations: (represented by lines). Represent interactions between actors and use cases.
* system boundary box: this shows the boundaries of the system, enclosing use cases which are part of the system and excluding those that are not.  
  + What term have we used before to describe what's in and out of the system? Scope. Things that are inside the box are within the scope of the system; things that are outside are outside the scope of the system.
  + You can also think of the place where the association intersects the system boundary box as an interface to the system.

Example of a use case diagram for an employee parking system:



Note the different parts of this use case diagram. It contains one Actor (Employee), and 5 use cases and associations. One use case is noted as being outside of the boundary of the system - while it is an activity a user may want to perform, it will be performed by some other system.

Some things to note:

* actors aren't just people – they can also be organizations or external systems that interact with your system.
* actors aren't specific people, but rather roles. Think about what role someone plays in the system. [Personas](http://www.agilemodeling.com/artifacts/personas.htm) can be used to give life to a generic actor role when determining more concrete requirements.
* One person can be performing in multiple roles, and what they do when interacting with the system under that role is different.
* a use case is functionality that provides something of measurable value to an actor
* the UML use-case diagram is not the only way to describe use-cases. The important part of the diagram is the use-cases themselves, so spending time on the text of the use case is more important than a syntactically correct diagram.

**Actor types**

**Primary actors** – is an actor that is trying to perform the desired functionality.

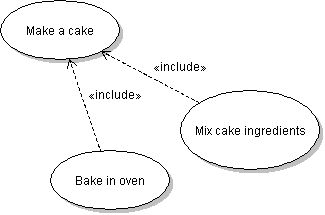
**Secondary actors** – is an actor that our system uses to perform that functionality.

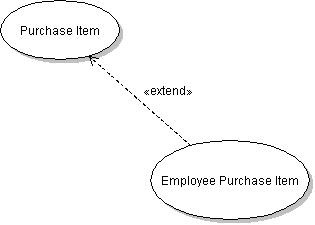
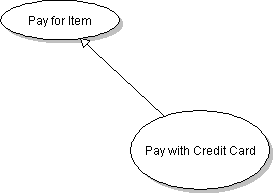
Often, primary actors are shown on the left side of the system, and secondary actors are shown on the right side. Not all systems have secondary actors.

**Relationships Between Use Cases**

Use cases can have relationships with other use cases. There are three possible use case relationships:

* **Include** – relationship where one use case includes the functionality of another use case as a part of itself.  
  + Indicated by a dashed line with an open ended arrow starting at the "part" which is included in the "whole", annotated with "includes"
  + Example: to complete the "make a cake" function, the "Bake in oven" and "Mix cake ingredients" functions must also be completed.



* **Extend** – relationship where one use case takes the functionality and characteristics of another use case and then adds on to it or changes it in some way.  
  + Often used to note that functionality is intended to be re-used in slightly different ways by different actors, or to add additional steps to an existing use case.
  + Indicated by a dashed line with an open-ended arrow pointing from the "child" (extending) use case to its parent (extended) use case.
  + Example: "Employee purchase item" includes all of the functionality of "Purchase item", but adds extra behaviour (such as applying a discount) to that base behaviour  
    
* **Generalization –** this association exists between use cases when one use case is a specialized version of another use case.
  + Similar to extends but with one difference – In generalization, any child use case can replace the parent use case without affecting business flow. In extends, the child use case is actually adding functionality on top of the parent’s use case functionality and therefore depends on the existence of the parent.
  + The more general use case will often not be implemented, but serves as a template describing the actions the child cases will have to perform.
  + Noted with a solid line with an hollow boxed arrow pointing from the child use case to the parent
  + For example: "Pay for item" describes the basic functionality, which could be implemented in many ways. One of these ways is "Pay with credit card", shown here, but others like "Pay with cash" could exist  
    

LO4.2 - Developing a use-case diagram

Ask the client (sometimes the client works with you when you are developing the diagram); typically, though, you make a list of questions and ask them the next chance you get. Remember, the whole purpose of this activity is to gain a better understanding of the system and the client's requirements, NOT simply to produce a diagram.

**Actor Identification**

* An actor is anything or anyone that will interact with the system
* Actors are always external to the system. Could be:
  + The varieties of different people using the system
  + Another software program that performs actions on the system.
  + A piece of equipment
* To identify actors look at existing project information (ie, project plan and project proposal) and ask:
  + Who operates the system?
  + Who shuts down the system?
  + Who maintains the system?
  + Are there external software systems that must be connected to?
  + Is there equipment that the system must interface with?
  + Who/What will supply, use, and remove data to/from the system?
* Another method for identifying actors is to look for nouns in any of the system descriptions/documentation.

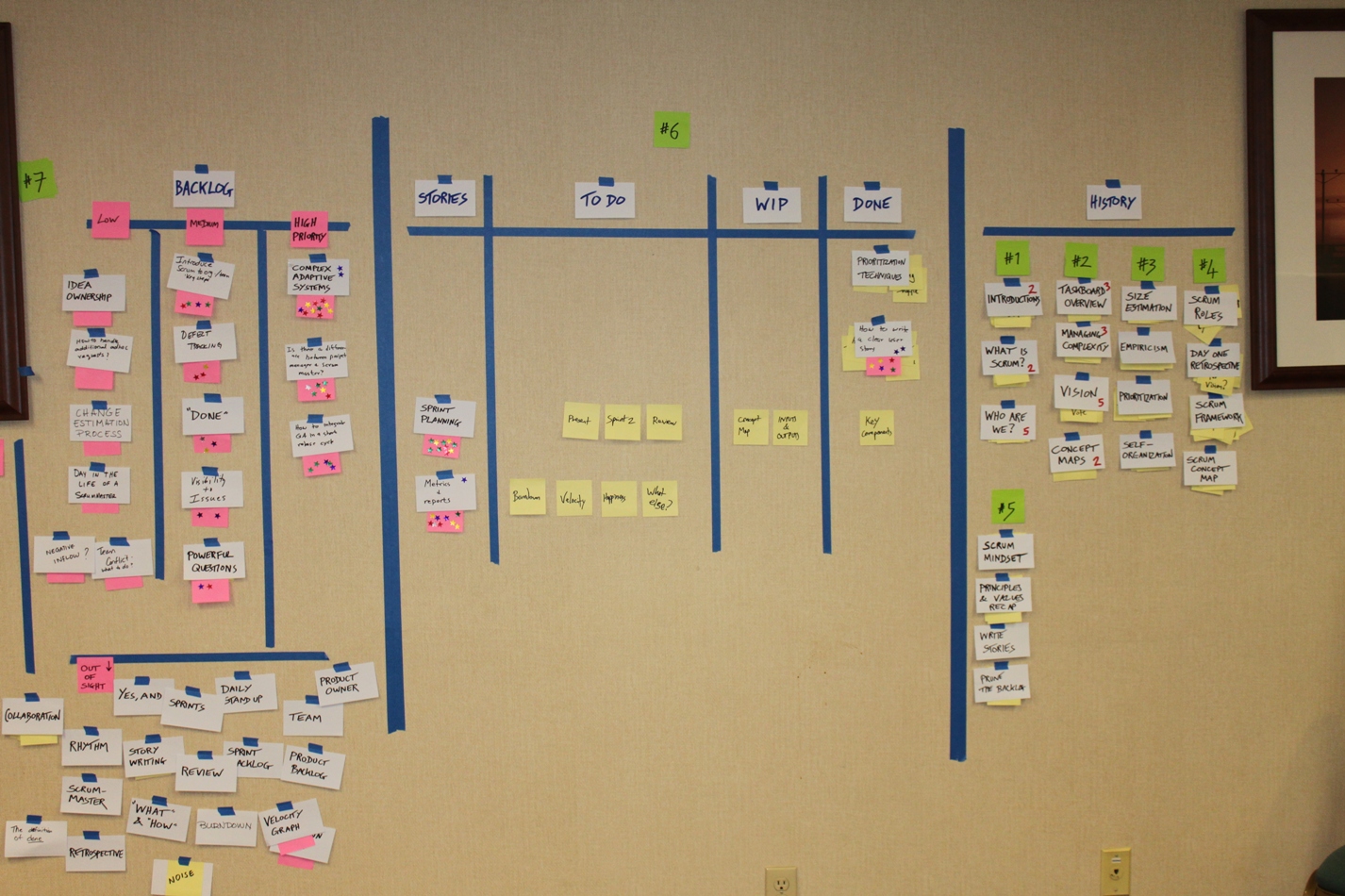
*NOTE: Your external actors* ***must*** *interact with* ***your system****.*

**Use Case identification**

* Use cases are functions that the system must perform
* To identify ask what each actor requires from the system. Ask questions like:
  + What are the goals that an actor expects the system to accomplish?
  + What information will an actor provide the system?
  + Will an actor require data to be stored, changed, or deleted?
  + Will an actor initiate functions on the system, like startup or shutdown?
  + What changes in the system do actors need to be notified of?
* Considering these questions will help identify what functions the system need to perform, which identifies the necessary use cases.

# Another method for identifying use cases is to look for verbs in the system description/documentation.

# LO4.3 - Describe User Stories



Example of a Story board used for organizing Agile projects.

## User Stories

Ron Jeffries, a well known Agile practitioner, defined User Stories as having three major components: the Card, the Conversation, and Confirmation.

### The card

The classic example of a user story uses a physical index card to keep track of all of the story information. The card serves as an easily manipulable placeholder and basic description of a single customer requirement. Index cards are used because the limited space keeps us from defining the requirement in too much detail. Instead we focus on open communication with the client and their changing needs.

Title

Example of a story card

|  |  |
| --- | --- |
| Agile Team Member documents requirements  Description | Business Level Acceptance Tests |
| As an agile team member, I want to document my user's requirements as stories so that I can see the progress of my project at a glance  Estimate  Priority  Prio: 18 Est: 4h | - Stories must include a descriptive actor name, rather than simply "user" or "person"  - Stories may include reasoning behind the described requirement in the form of "so that I"  - Stories should be marked with a priority and time estimate, decided in consultation with the team and stakeholders  - Stories must be written to avoid specific implementation details  - Stories may be written for functional or non-functional requirements |

**FRONT OF CARD BACK OF CARD**

**Title**

* Describe requirements expressed using active language
* Include an actor, a brief description of the feature.
* The title should be short and free of specific details which tie the team to a specific implementation.

Active language means that story titles don’t include words like: "can" or "may", but instead are phrased as "will".

Example: "Facilities department employees view all parking spots".

**Description**

* A short description of the requirement.
* Describe the who, what and (optionally) the why of the requirement.
* A common format for the description is:  
  "As a <user role>, I want to <action>, so that I can <reasoning>".

**Estimate**

* How much time the team thinks the story will take to complete.
* The unit varies, but in CST we use person-hours.

**Priority**

* How important the story is to the client.
* The unit varies. In CST it is up to the group. Recommend using either a priority number (with 1 being highest) or giving a more general low-high rating.

### The Conversation

User stories are often described as a "promise of future communication". Force the team to rely on an ongoing dialog with their client and more domain specific documentation in order to determine very specific requirements. They provide a way to keep ourselves organized and in tune with our client's needs.

### The Confirmation

These are frequently called:

* "acceptance criteria",
* "definition of done",
* "conditions of satisfaction", or
* "business level acceptance tests".

Include details of the conditions under which our client will consider the feature described by the story to be complete. The business level acceptance tests can be used by a product owner or product team during a demo of the story to see whether or not it can be considered done.

The confirmation can also include implementation details commonly includes non-functional requirements, such as security or performance implications, which can be used as part of the tests.

A common system for describing the attributes of a good user story is the "INVEST" acronym, introduced by Bill Wake. INVEST stands for:

**Independent**

* Minimize the number of other stories which must be created first in order to implement other stories. This might mean combining several smaller stories to make one independent story.
* Can be dealt with through customer priorities. Having a high priority story dependent on a low priority story can make development difficult. Sometimes explaining the dependency to the client can help prioritize the stories.
* Always attempt to minimize story dependencies.
* Sometimes stories may seem dependent on another when they’re not. Example: You may have 2 stories:
* 1) An attendant adds a parking space to an employee
* 2) An attendant adds parking spots

It might seem that you must do #2 before 1 but in reality, you can just add the data needed to complete the story for #1 to the underlying database.

**Negotiable**

* Minimize the number of implementation details, or forced pre-choices when writing the stories.
* Stories should provide enough detail to be implemented, but not so much detail that you are unable to change or refine the requirements.
* Details should be determined through future communication.
* "Commissionaires will view a list of vehicles in a parking spot in a ListView ordered by vehicle make" is far too specific, making the story difficult to change if a client requirement (ie that we create an android app) changes. It contains implementation details forcing the team into a corner.

**Valuable (to users and customers)**

* A common mistake in creating user stories is to only create stories which affect the end-users of the software. While these stories are important, the frequent result of this mistake is that non-behavioural requirements, like performance or security, are left unimplemented.
* End users and purchasers or owners can have differing requirements.
* Another common mistake is to write stories that only the team will care about. These stories will often contain obvious implementation details. For example "The system will use MySQL connection pools" does not provide business value to the customer. It may be a requirement all the same, but on its own does not make a good user story, and would be better suited to acceptance criteria or a team standard.

**Estimable**

* Every story must be assigned an estimate of time to be completed by the team. In order to estimate the story the team needs enough information and background in order to provide an accurate idea of time to complete.

**Sized appropriately**

* Stories which are too small will often introduce temporal dependency,   
  , (separate stories which must be implemented in a specific sequence in order to make it possible to implement).  
  For example, if a team were to break up the stories about logging-in to a system to:

1. "Visitor views login form",
2. "Visitor enters username",
3. "Visitor enters password",
4. "Visitor submits login form",
5. "Visitor redirected to homepage if credentials are correct",

It would be difficult to implement the stories in just any order. Having tiny stories like this can also add overhead which can slow down the team's progress.

* Stories which are too large will be difficult to reliably estimate.
  + It's easy to be confident about saying that something will take 2-4 hours, but much more difficult to estimate with confidence something that might take several weeks.
* Reasonably small stories can help with team morale. Accomplishing more tasks more quickly, which is easier to do with small tasks, can often lead to a better feeling of accomplishment and makes upcoming work feel less daunting.
* An appropriate story size depends on the team size, knowledge, technological confidence, and will generally be arrived at over time as the team becomes familiar with how each-other works.

**Testable**

* The story must be written so that it is possible to test.
* Untestable stories are seen when used to test non-functional requirements.
  + ie "The system must be use the latest web technologies" or "The system must be accessible to users at all times"
* When creating stories, always think of how you will test them. Tests can come in a variety of forms
  + Unit tests - Code that tests the code itself
  + Business level acceptance tests (acceptance criteria)
  + Formal acceptance tests

Bill Wake, the creator of the INVEST acronym, has created a video describing, in his own words, the reasoning behind each of the parts of INVEST. <https://www.youtube.com/watch?v=uj5iUbDf-iw>

## Use Case Scenarios

* Also called user scenarios, scenarios, or use cases.
* Use case scenario is a narrative, sequential description of a single occurrence of a use case. Use case scenarios are used to illustrate alternative paths through functionality of a use case.
* Use cases are generally very specific- they lay out the exact sequence of events from the beginning to the end of the user’s interaction through the system.
* Closely intertwined with the UML activity model.

### Things to include in scenarios:

* give the scenario a unique ID (ex., Scenario #2)
* give a short description of the scenario (the goal)
  + ie "Successfully logging in to the system with valid credentials"
* give a numbered list of the steps the user takes in scenario, in sequence (the actions)
  + with these steps, give brief details of the system's responses (the events)
* use specific people or systems (the actors)
  + this is where personas come in handy

### Things to not include in the scenario:

* Multiple failure conditions
* Multiple branching paths
  + Use multiple scenarios instead. For example, have one scenario for the process of logging in successfully, one for a failed login due to a bad password, one for a failed login due to the AD domain being unavailable, etc.
  + Keep scenarios as simple as you can, within reason.

In short, user scenarios are an expanded form of the use cases described in the UML use-case diagram.

# LO4.4 - Identify User Stories and Scenarios

It is common on agile teams to document requirements in the form of user stories and user scenarios while performing interviews, brainstorming sessions, shadowing and reviewing the results of questionnaires. While many project teams will also document requirements using a formal requirements document, some will use stories as their primary documentation of requirements.

The following user story and scenario were created during the analysis process for a software project intended to automate the assignment of parking spaces to employees at a local institution, named CST Ventures here, as well as to manage those spaces once assigned. In summary, this story was created to encode a requirement discovered during a conversation with employees of the organization (Using the "Interview" technique for discovering requirements). Employees stated that they had in the past been issued a ticket when parking in their own spot, but using a different vehicle than they had previously registered. This requirement could be initially documented in several ways, but in this case was first included as one of the user scenarios for the "employee" actor discovered in LO4.2. Use case diagrams are an excellent way to quickly jot down future user stories, as they include both the actor and the basic behaviour which are also recorded in stories.

|  |  |
| --- | --- |
| Employee adds a vehicle to their spot |  |
| As an employee of CST Ventures, I can add a vehicle to my own spot so that I don't get a ticket when I forget my parking pass at home  Prio: 18 Est: 4h | - Acceptance tests are described in a later learning step |

**FRONT OF CARD BACK OF CARD**

While the text of this story, the estimation and priority could change depending on the project team and their standards, in this case it was following the standards of the project team. The story title describes the action the software team will implement. The description describes the specific actor who will receive value from the story being implemented, enough detail about the activity being performed to spark future conversations and a brief narrative around why the story would be valuable.

Use case scenario for "Employee adds a vehicle to their spot" is below. Remember that use case scenarios describes the steps a specific instance of an actor would go through during a single, specific interaction with the system. It is possible to create multiple use case scenarios for a single case from a use case diagram or for a single user story.

|  |
| --- |
| ID: #5.4 |
| Description: Employee adds a new vehicle to their own parking spot |
| Employee:   1. Logs into the system using their own, valid credentials 2. Navigates to the "Manage vehicles" menu item 3. Chooses the "Add vehicle" option 4. Fills in vehicle model (civic), make (honda), colour(black) and valid license place(777GGG) 5. Submits form 6. Since user currently has less than 3 vehicles, the system saves the vehicle and displays a "Success" message |