# Analysis - use case structure

This document shows the progression of the workflow in the creation of a set of use cases. The example domain is a restaurant and we are focusing on the front end of the house activities.

## Find the actors/roles

Discover the actors from the many types of **job titles** that interact with the activities. These can be your first candidate actor names. Actors will have the same access privileges or security group settings as each other allowing them to do the same tasks in an automated system. In a business system, one that is a workflow to produce value with automation or without but has people taking actions, the privileges are the authority to act or job responsibilities.

* Server
* Cook
* Customer
* Manager
* Expediter
* Host

## Find candidate use case names

The name of the piece of scope that is to be performed comes is different amounts of work. This is the granularity of the process. A set of use case names should all be at the same level of granularity. For now, you will use a brainstorming style of collecting possible names of use cases. It should be any small or large process that you can think of and worry about the amount of scope later if you don’t know. You can also add others later. Follow the rule of writing them with verb-noun syntax.

* **Take** order
* **Cook** food
* **Validate** order is correct
* **Plate** food
* **Deliver** food
* **Clean** table
* **Ask** about dessert
* **Greet** customer
* **Prepare** bill

## Validate use case names – rules

Now you take the use case names and ask questions to see if they fit the rules of a goal level use case. That’s all you look for. Some may be smaller, some may be larger, some may be rules. Here’s a listing of the rules applied to bad examples and how to fix them, but you will likely confirm what each use case is sequentially.

|  |  |  |  |
| --- | --- | --- | --- |
| Rules | Bad examples | What to do if it fails and testing | Rewritten |
| Written in verb-noun syntax | Ketchup bottle on table  Station is cleaned often | Rewrite process you had in mind with strong verb first. In many cases you will want to describe with a clause like “for someone” or “with data” | Put ketchup on table for customer  Clean station for servers |
| Describes one path / no combination | Get menus and give to customer | Rewrite into two or more.  Try asking “Can I propose a test that will pass or fail for this?” What if you get the menu but don’t give it to the customer? It neither passes or fails. | Get menus  Give menu to customer |
| Describes one path / not vague | Take care of orders  Manage customers | Break down the process into several goal level workflows that make sense.  Try asking “If I told an employee this, would they be sure about what I was thinking?” | Take food order from customer  Serve ordered food to customer |
| Describes one path / not a rule | Check food is correct before serving  Include specials when you greet customer | Rules are associated with a process and are not functional. They are tests to be run to go to the next step or they validate data. | Serve ordered food to customer |
| Branches to two separate use cases | Ask about dessert and get dessert menu if they do or give bill if they don’t  Seat customer at a table if possible unless they ask for a booth | Any conditional statement that has two outcomes points to two separate use cases. Find the use case that it rolls up to and rename it. | Take food order without dessert  Take food order with dessert  Seat customer at table  Seat customer at booth |
| Ends in value for service owner / not a task | Greet customer  Plate food | Think of the process that it rolls up to. The candidate will be a part of that use case but as a task in that use case. It contains less scope or amount of work than necessary to achieve value.  Try asking “If done a hundred times, will there be real value?” | Take food order (value = commit to sales transaction) |
| Ends in value for service owner / can repeat | Serve food  Put garnish on plate | Some use cases are dependent on other use cases and can’t start until another one has completed. This is the only exception to repeatability. Here Serve Ordered Food is dependent on Take Food Order.  Try asking “Can this be done immediately for another person after this completes?” | Compute bill  Add item to order |
| Ends in value for service owner / process state is reset | If table is dirty, clean table before putting setup on it | This rule usually points to tasks that must be added to the use case. If you got menus to the table, did you return the menus back where you got them? If you started with a clean table, is the table cleaned at the end of the service? | Clean table  Put setup on table |

## Validate use case names – results

After applying the rules, you should have a very small list of use cases. I see around 90% of candidate use case names as not being valid.

* Seat customer
* Take food order from customer
* Serve ordered food to customer
* Bill customer
* Reset table for customer

## Validate actors with use cases

After the use case names become clear, the actors become clear also. Associate them. More than one actor can do a use case. But the only rule here is if you can’t think of something that an actor does differently than another actor, they really are the same actor even though they may have two different job titles.

|  |  |
| --- | --- |
| Actor | Use case |
| Host, server | Seat customer |
| Server | Take food order from customer |
| Server, manager | Serve ordered food to customer |
| Server | Bill customer |
| Server, busser (new) | Reset table for customer |

## Draw use case diagram

Now you design the diagram so it’s easy to read. This one uses the online diagram generator from <https://yuml.me/diagram/scruffy/usecase/draw>

### Yuml code

[Host]-(Seat customer),

[Server]-(Seat customer),

[Server]-(Take food order from customer),

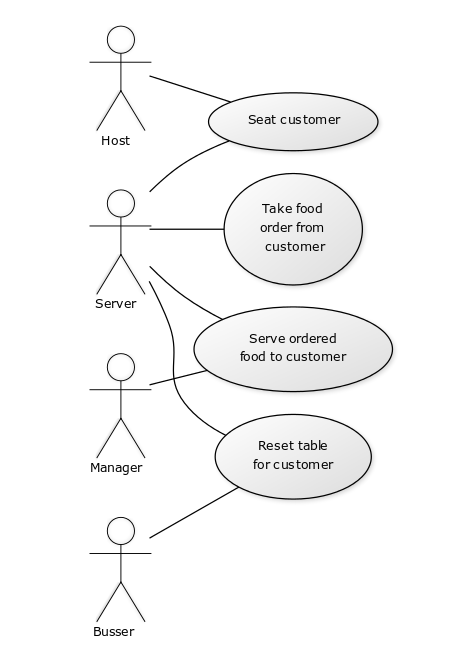
[Server]-(Serve ordered food to customer),

[Manager]-(Serve ordered food to customer),

[Server]-(Reset table for customer),

[Busser ]-(Reset table for customer)

### Yuml diagram



## Yuml diagram with extension and include

The extension shows an **optional** part of a goal level use case that may or may not be added. But it doesn’t cause the flow of the use case to end differently. It’s an add-on.

The include shows a **required** part of a use case as a possible group of tasks but is not large enough scope to be a goal level use case.

The dotted-line arrow is about a dependency and not about order of activity like in a flowchart. What the arrow comes from depends on what the arrow goes to.

### Yuml code

[Host]-(Seat customer),

[Server]-(Seat customer),

[Server]-(Take food order from customer),

[Server]-(Serve ordered food to customer),

[Manager]-(Serve ordered food to customer)

(Take food order from customer) < (Take dessert order)

(Seat customer) > (Give menu to customer)

### Yuml diagram

### 