# Lab 05: Use Cases and Use Case Diagrams

In this lab we will extend our initial user stories into fuller use cases, trying to capture the detail of the work we require to complete the new HR System. We will also look at **Use Case Diagrams** to visually define our use cases and their relationship.

## Behavioural Objectives

* [ ] **Define use cases** using *Cockburn’s Use Case Template*.
* [ ] **Define use case diagrams** using *PlantUML*.

## Our Current User Stories

In [Lab 03](../lab03) we defined our eight user stories from our vision statement:

1. As an *HR advisor* I want *to produce a report on the salary of all employees* so that *I can support financial reporting of the organisation.*
2. As an *HR advisor* I want *to produce a report on the salary of employees in a department* so that *I can support financial reporting of the organisation.*
3. As an *department manager* I want *to produce a report on the salary of employees in my department* so that *I can support financial reporting for my department.*
4. As an *HR advisor* I want *to produce a report on the salary of employees of a given role* so that *I can support financial reporting of the organisation.*
5. As an *HR advisor* I want *to add a new employee’s details* so that *I can ensure the new employee is paid.*
6. As an *HR advisor* I want *to view and employee’s details* so that *the employee’s promotion request can be supported.*
7. As an *HR advisor* I want *to update an employee’s details* so that *employee’s details are kept up-to-date.*
8. As an *HR advisor* I want *to delete an employee’s details* so that *the company is compliant with data retention legislation.*

So far, we have implemented user story 1 and 6. As an exercise, you are expected to complete **all** the use cases defined above. This lab will only provide one example: user story 4. However, you should be able to complete the use case for 1 and 6 from our previous work, and we will be reviewing the other use cases through the rest of the lab series.

## What is a Use Case?

From [Wikipedia](https://en.wikipedia.org/wiki/Use_case):

In software and systems engineering, a use case is a **list of actions or event steps** typically **defining the interactions between a role (known in the Unified Modeling Language (UML) as an actor) and a system to achieve a goal**.

To simplify, a use case is:

1. a list of actions/events;
2. by an actor;
3. interacting with a system;
4. to achieve a goal.

Our user stories are a form of use case, sometimes referred to as a *casual* use case. The user stories we have defined two elements directly and one indirectly of a use case. For example, let us consider use case 4:

1. no list of actions defined.
2. HR advisor (actor).
3. interacts with HR system.
4. to produce a report on the salary of employees of a given role (goal).

Our user stories are lacking in the following two areas:

* there is no list of actions.
* the system interaction is opaque.

A fuller use case will allow us to address these two issues. The list of actions we are normally defining as we build a feature. Let us put some thought into the actions beforehand, but remember that **details can change!** We are planning but not putting our plan in stone until we have finished a feature.

## Defining Use Cases for Our System

[Lecture 10](../../lectures/lecture10) goes into more detail about use cases. Here, we are going to cover the application of these ideas. We are going to use a version of [Cockburn’s Use Case Template](https://cis.bentley.edu/lwaguespack/CS360_Site/Downloads_files/Use%20Case%20Template%20%28Cockburn%29.pdf). You can see a sample of [Use Case 4](use-case-4.md) using this template style.

Cockburn’s template contains the following sections of note:

* **Goal in Context** - we will use our user story.
* **Scope** - is discussed more in the lecture. Scoping is an important consideration in any work you do.
* **Level** - what level is the use case targeted at. This is discussed further in the lecture.
* **Preconditions** - what do we **expect** is true before the use case is executed.
* **Success Condition** - what will happen on completion of the goal.
* **Failed Condition** - what will happen on failure of the goal.
* **Primary Actor** - the main actor of the use case.
* **Trigger** - how is the use case started.
* **Main Success Scenario** - what are the steps leading to success.
* **Extensions** - what might happen at a given step to stop the use case.
* **Sub-variations** - any other branches that a step can take?
* **Schedule** - when does the use case need to be delivered.

### Exercise: Define the Other Use Cases

First, create a new folder in your project called use-cases. Copy the use-case-4.md file provided into this folder.

Your exercise is to complete the other seven use cases for the HR system. Write these in Markdown (.md files). IntelliJ comes with a default plugin to support Markdown. If you are unfamiliar with Markdown, then there [are](https://www.markdowntutorial.com/) [several](https://guides.github.com/features/mastering-markdown/) [tutorials](https://www.markdownguide.org/getting-started/) [available](https://learnxinyminutes.com/docs/markdown/). This is a good opportunity to work with your team.

## What is a Use Case Diagram?

Use cases can also be visually represented using a **Use Case Diagram**. Typically seen as part of the **Unified Modelling Language** (UML) (see [Lab 6](../lab06) and [Lecture 11](../../lectures/lecture11)), use case diagrams allow us to see how use cases interact simply. However, they do lack the detail required to fully implement and understand features, and therefore should be seen as a support tool for software development. In particular, they can communicate with stakeholders quickly about how the engineers see the system working.

### Use Case Diagram Symbols

Use case diagrams are quite simple, requiring only stick men, arrows, and ellipsoids at the most basic level. We will cover the common use case symbols from UML below.

#### Use Case

To illustrate a use case we use an ellipsoid with text as below:

Use Case

Use Case

#### Actor

Actors are represented by stick figures:

Actor

Actor

#### Use Case Relationships

Use cases can also relate to each other, typically in **include** and **extend** cases. Below is the diagram:

Include Use Case

Include Use Case

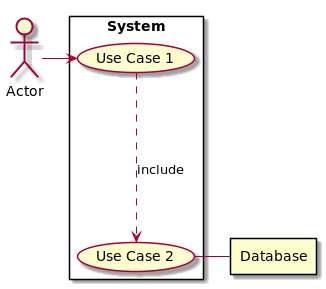
Extend Use Case

Extend Use Case

An *include* relationship is one where a use case includes (i.e. *uses*) another use case to perform its functionality. An *extend* relationship is one where a use case extends (e.g. supports an edge-case) from another use case. It provides a special version. These should have been identified in the **Extensions** section of the use case.

#### System

A use case typically exists within a system, or communicates with another system. For example, see below:



System

*Use Case 1* and *Use Case 2* both exist within the *System*. *Use Case 2* also communicates with an external system - *Database*.

## Using PlantUML in IntelliJ

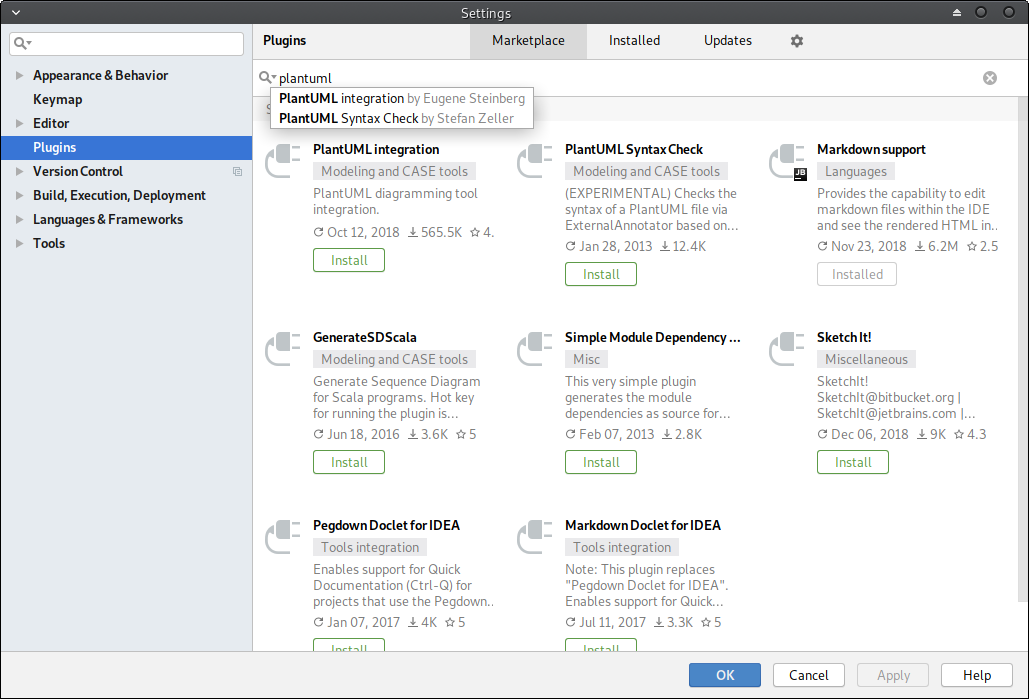
There are quite a few UML diagramming tools out there. However, we want to store our diagrams in our GitHub repository. As Git repositories don’t like binary files, we will use a textual representation via [PlantUML](http://plantuml.com/). This is a common textual standard to describe UML diagrams and can be used to generate images. We will do this via an IntelliJ plugin.

### Install GraphViz

To use the PlantUML plugin in IntelliJ you will first need to install GraphViz on your machine. GraphViz is a [graph drawing tool](https://en.wikipedia.org/wiki/Graph_drawing) that several tools use to layout diagrams. Download instructions for GraphViz are available from [here](https://www.graphviz.org/download/). For Windows users, install the **stable release**.

### Install PlantUML Plugin

Next we need to install the PlantUML plugin for IntelliJ. Go to **File**, **Settings** then **Plugins** to open the **Plugins Window**. Search for **PlantUML**. The plugin you want is **PlantUML integration** as shown below:

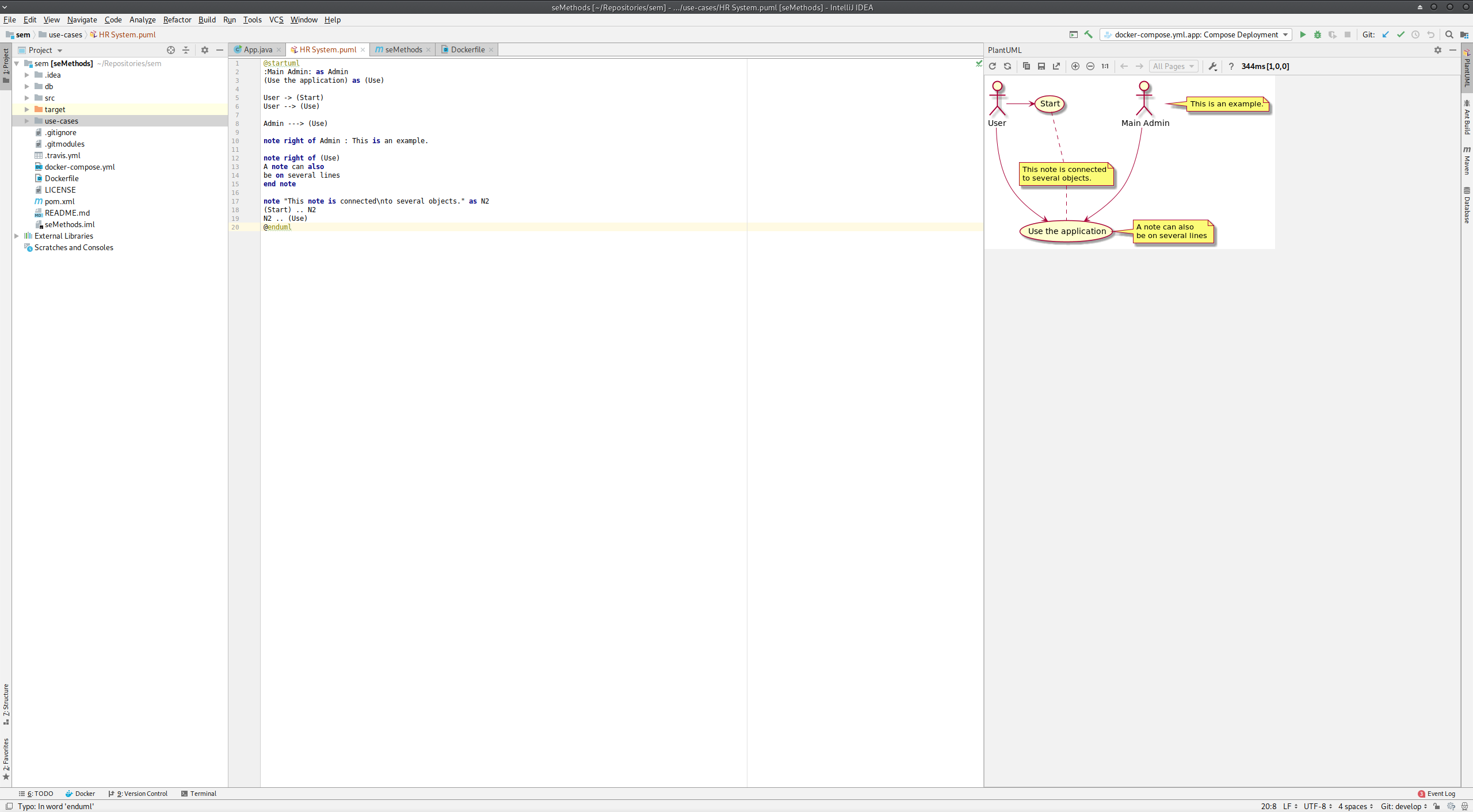


IntelliJ PlantUML Plugin

Click **Install**, then **Accept** and finally **Restart IDE**. Once IntelliJ has restarted, PlantUML should be available.

### Creating a Use Case Diagram

To see the PlantUML is set up correctly we need to create a diagram. **Right click** on the **use-cases folder**, select **New**, and the **UML Use Case**. Give it the name **HR System** and click **OK**. This should provide you with the following window:



IntelliJ PlantUML Diagram

**If you don’t see screen as above a couple of things to check**:

* did you install GraphViz?
* if so, open the settings for PlantUML (click the small spanner above where the diagram should be), and browse for the **dot** executable which will be where you installed GraphViz.

### PlantUML Syntax

A more comprehensive guide is available [here](http://plantuml.com/PlantUML_Language_Reference_Guide.pdf). We will examine the basics.

A PlantUML file starts and ends with the following:

@startuml  
  
@enduml

We define a use case as follows:

usecase "Use Case"

We can also provide a name for the use case. This makes it easier to connect them later:

usecase UC1 as "Use Case 1"  
usecase UC2 as "Use Case 2"

Actors are defined as follows:

actor "Actor"

And they can likewise be named:

actor A1 as "Actor 1"  
actor A2 as "Actor 2"

There are numerous methods to lay out arrows - [see the tutorial](http://plantuml.com/PlantUML_Language_Reference_Guide.pdf). For example:

actor A1 as "Actor 1"  
usecase UC1 as "Use Case 1"  
  
A1 --> UC1

Systems can be defined using rectangles:

rectangle Database  
  
rectangle System {  
 usecase UC1 as "Use Case 1"  
 UC1 --> Database  
}

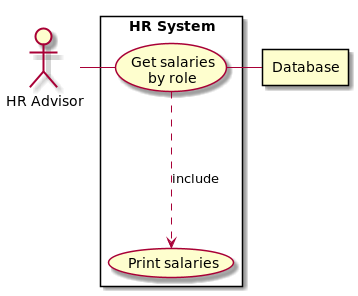
Anything defined within the rectangle curly braces are part of the System.

### Diagram for Use Case 4

Let us look at an example from our system. Below is use case 4:

@startuml  
  
actor HR as "HR Advisor"  
  
rectangle Database  
  
rectangle "HR System" {  
 usecase UC4 as "Get salaries  
 by role"  
   
 usecase UCa as "Print salaries"  
   
 HR - UC4  
 UC4 ..> UCa : include  
 UC4 - Database  
}  
  
@enduml

This will produce the following diagram:



HR System Use Case Diagram

### Exercise: Complete the Use Case Diagram

Your task now is to complete the use case diagram for the entire set of use cases defined. You only need one diagram for all the cases. Again, work with your team, and seek feedback. We will revisit various parts of the diagram throughout the module so you can adjust as you are going along.

## Next Feature: Salary By Role

Now it is time to work on our next feature - user story 4: As an *HR advisor* I want *to produce a report on the salary of employees of a given role* so that *I can support financial reporting of the organisation.*

Remember the steps you took last week for executing a Sprint:

1. Decide which user story/stories to work on for the next Sprint.
2. Create a new Sprint on Zube.
3. Add the user story card(s) to the Ready column in Zube.
4. Add any additional task cards to Zube and put in priority order.
5. Pull the latest develop branch.
6. Start a new feature branch for the task(s) or user story.
7. Select task to work on in Zube.
8. Work on task.

We only have one task to do this week: get the salaries by department. This is very similar to the last feature - get all salaries - but with an additional restriction. Therefore it is your task to implement this feature on your own.

### Exercise: Implement Salaries by Role Feature

The SQL required for this query is below:

SELECT employees.emp\_no, employees.first\_name, employees.last\_name, salaries.salary  
FROM employees, salaries, titles  
WHERE employees.emp\_no = salaries.emp\_no  
AND employees.emp\_no = titles.emp\_no  
AND salaries.to\_date = '9999-01-01'  
AND titles.to\_date = '9999-01-01'  
AND titles.title = '<title>'  
ORDER BY employees.emp\_no ASC

<title> is replaced by the name of the role (title). For example, the end of the Engineer salary information is:

...  
499838 Annemarie Peroz 53972   
499843 Vitaly Zucker 66847   
499855 Constantine Michaels 49559   
499856 Yoshinari Theuretzbacher 50966   
499857 Leszek Tempesti 60478   
499896 Gianluca Rando 59952   
499900 Leon Baba 51414   
499904 Kazuhiro Velasco 47104   
499913 Masako Heiserman 73788   
499918 Hilary Rodiger 55843   
499927 Manohar Heemskerk 83769   
499935 Ymte Perelgut 77520   
499936 Chiranjit Himler 54253   
499948 Cordelia Paludetto 45625   
499962 Yongqiao Dalton 57667   
499973 Lobel Taubman 61400   
499979 Prasadram Waleschkowski 54088   
499990 Khaled Kohling 45512   
499993 DeForest Mullainathan 44305   
499995 Dekang Lichtner 52868   
499999 Sachin Tsukuda 77303

## Cleaning Up and Committing

And now end your Sprint and clean up. Follow the process as defined at the end of [Lab 4](../lab04).