UML for Industrial Design Engineers

Many products have software embedded to give them a kind of intelligence. The software describes the behaviour. The developers of the product have to think through the complete behaviour of the product and describe it.

There are many different programming languages with which you can write software. These different languages have, just like normal languages, different grammar, rules and dialects.

These languages are made for different hardware or operating system (OS). Software made for Apple computers, for example, differs from software that is made for Windows, Linux or Android. This means that the same software has to be made for different platforms.

You can learn of course, one or more languages but it is not necessary to know them all when you are developing new software. Software can be developed with Unified Modeling Software, UML, which later can be translates in a specific language.

UML is a general-purpose modelling language that gives a standard way to visualize the design of a system. It consists of different diagrams that describes the behaviour of the software and thus the product in which the software is embedded. An overview of these diagrams given in the next figure (fig. 1), but you don't need them all, it depends on the purpose.

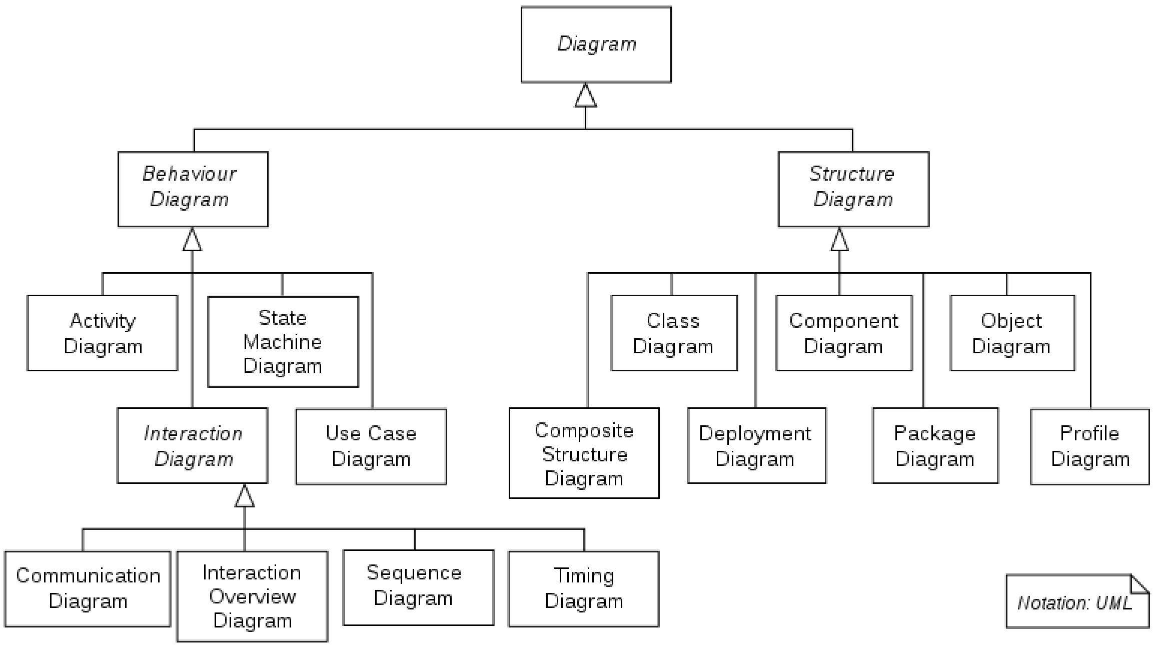


Figure 1 UML diagrams (Bron: "OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1". Object Management Group. 9 April 2014.)

How to start: Use Case Diagram

Where do you start when developing software for a product. As a designer you have probably already thought about use scenarios for your product. These use scenarios can be a starting point for your program. For example a coffee vending machine. Let's look at a part of the use scenario:

- User selects a type of coffee
- User chooses milk or no milk
- User has to pay with his card

From this scenario you can make a Use Case Diagram, one of the UML diagrams.

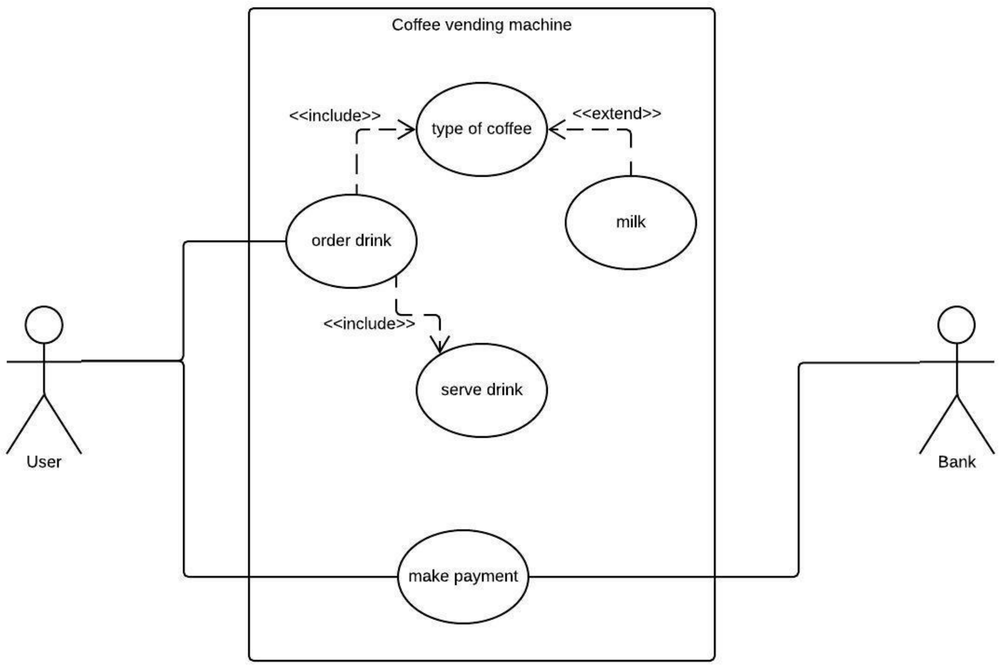


Figure 2, Use Case Diagram, Coffee vending machine.

How to make such a diagram is explained in the following video:

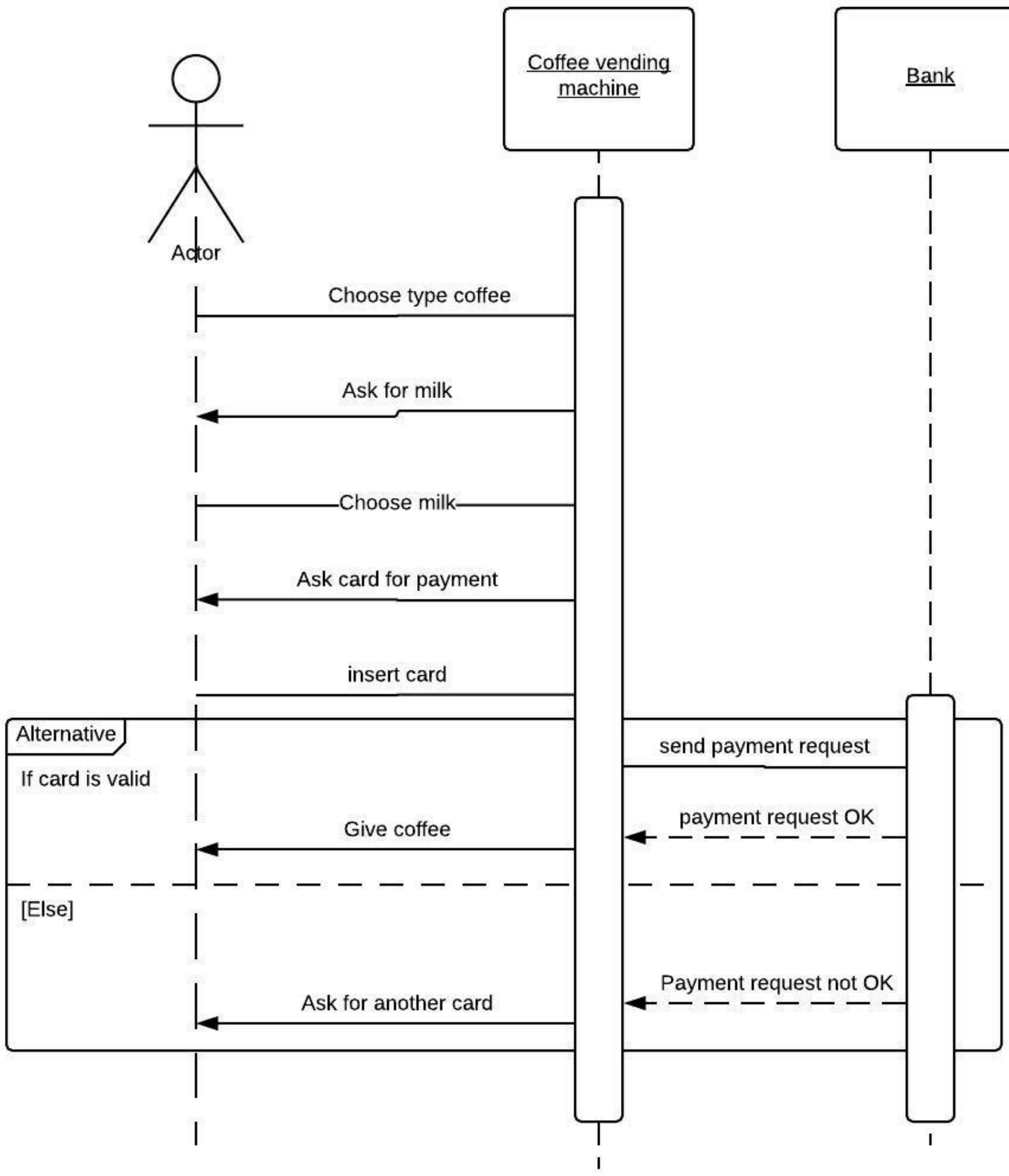
https://www.youtube.com/watch?v=zid-MVo7M-E

Sequence Diagram

“Sequence diagrams are sometimes called event diagrams or event scenarios. A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.” https://en.wikipedia.org/wiki/Sequence_diagram

A sequence diagram is not static. It illustrates the interactions between different parts of a system and shows also different possibilities in interaction (for example if then statements).

It gives insight in the use of the product and situations that occur.



Instructions on making a sequence diagram:

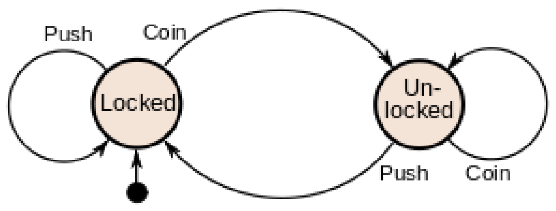
https://www.youtube.com/watch?v=pCK6prSq8aw

State Machine

A state machine is a part of code that defines a state at a certain moment. An external input can change the state at a given time. This is called a transition. An example with a turnstile taken from Wikipedia illustrates this: ”A turnstile, used to control access to subways and amusement park rides, is a gate with three rotating arms at waist height, one across the entryway. Initially the arms are locked, blocking the entry, preventing patrons from passing through. Depositing a coin or token in a slot on the turnstile unlocks the arms, allowing a single customer to push through. After the customer passes through, the arms are locked again until another coin is inserted.

Considered as a state machine, the turnstile has two possible states: ***Locked*** and ***Unlocked***.[3] There are two possible inputs that affect its state: putting a coin in the slot (***coin***) and pushing the arm (***push***). In the locked state, pushing on the arm has no effect; no matter how many times the input ***push*** is given, it stays in the locked state. Putting a coin in – that is, giving the machine a ***coin*** input – shifts the state from ***Locked*** to ***Unlocked***. In the unlocked state, putting additional coins in has no effect; that is, giving additional ***coin*** inputs does not change the state. However, a customer pushing through the arms, giving a ***push*** input, shifts the state back to ***Locked***.

The turnstile state machine can be represented by a state transition table, showing for each possible state, the transitions between them (based upon the inputs given to the machine) and the outputs resulting from each input:

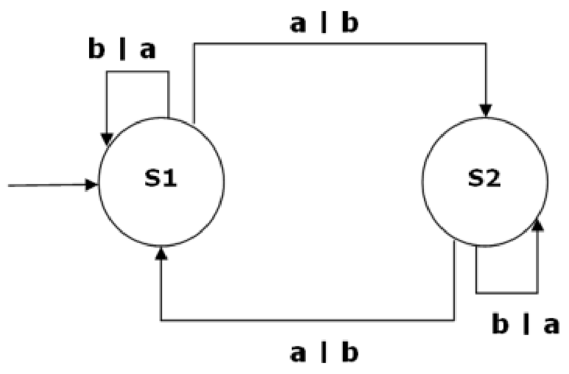


- The turnstile has two states: ***Locked*** and ***Unlocked***
- Two inputs that affect its state: putting a coin in the slot (***coin***) and pushing the arm (***push***)

Current State	Input	Next State	Output
Locked	coin	Unlocked	Unlock turnstile so customer can push through
	push	Locked	None
Unlocked	coin	Unlocked	None
	push	Locked	When customer has pushed through, lock turnstile

"(source: https://en.wikipedia.org/wiki/Finite-state_machine, December 2018)

A state machine can be drawn in UML like this:



- Circles labelled **S1** and **S2** represent the **states** of our machine
- The arrows between the states are called **transitions**.
- The transitions are labelled with two symbols (**a | b**) where a is the **input symbol** or the **trigger** for the transition.
- The starting state is indicate with an arrow on the left of the diagram.

Current State	S1	S1	S2	S2
Input Symbol	a	b	a	b
Next State	S2	S1	S2	S1
Output Symbol	b	a	b	a

So output depends not only on input, but also state you are in.

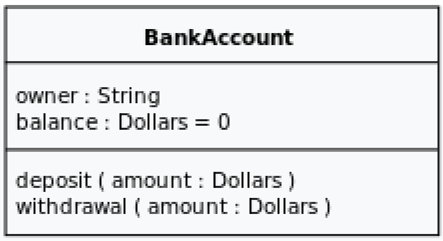
Class diagram:

A **class diagram** describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram can be seen as a building block of object-oriented modeling. These diagrams can be used by the actual programming.

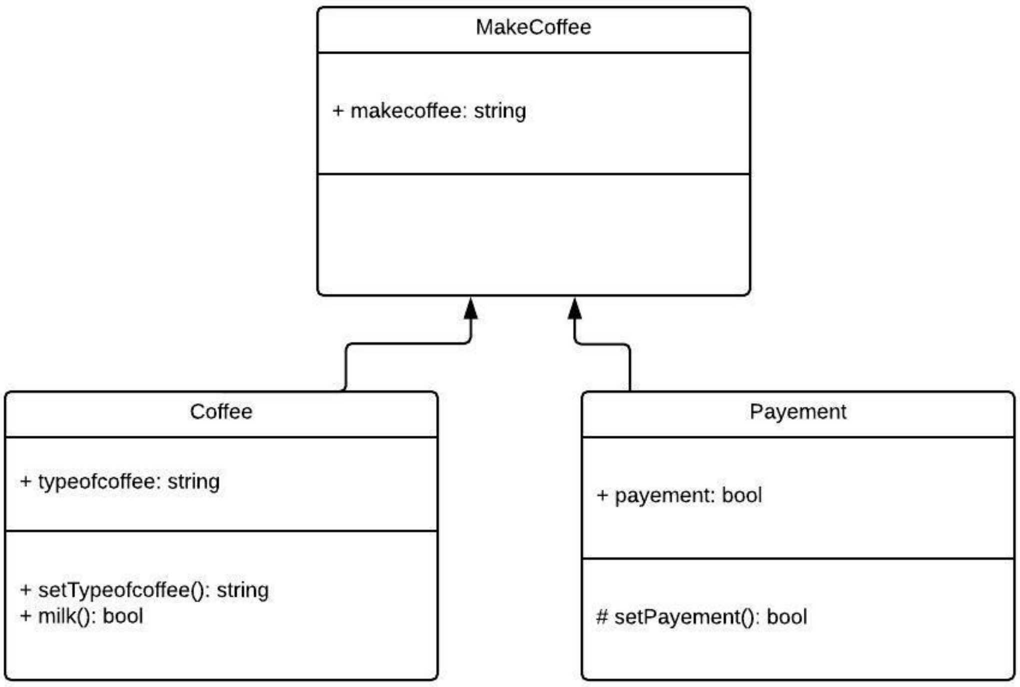
In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class.
- The middle compartment contains the attributes of the class.
- The bottom compartment contains the operations the class can execute.



A class with three compartments.

A class is one piece of program, a building block. You can combine different classes to describe the complete product. This is called object orientated programming. These classes can interact in different ways with each other.



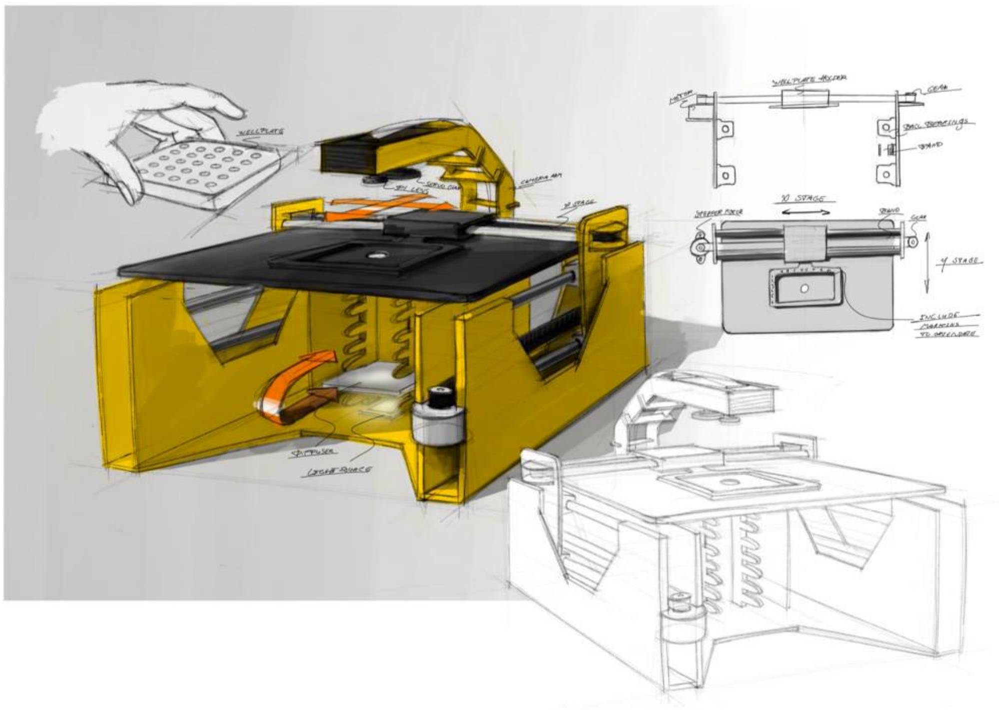
Instructions on making a class diagram:

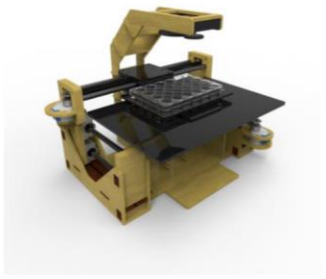
https://www.youtube.com/watch?v=UI6lqHOVHic&t=311s

Example:

From graduation report: Ernst Paul Swens, High-throughput screening device for Caenorhabditis elegans, June 2018

Ernst Paul Swens graduated (grade: 10!) on a screening device that can be used in a laboratory setting. He also developed software for the apparatus. He made use of UML schemes to do this. These are shown here:





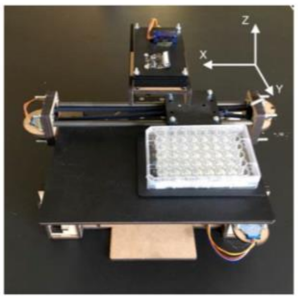
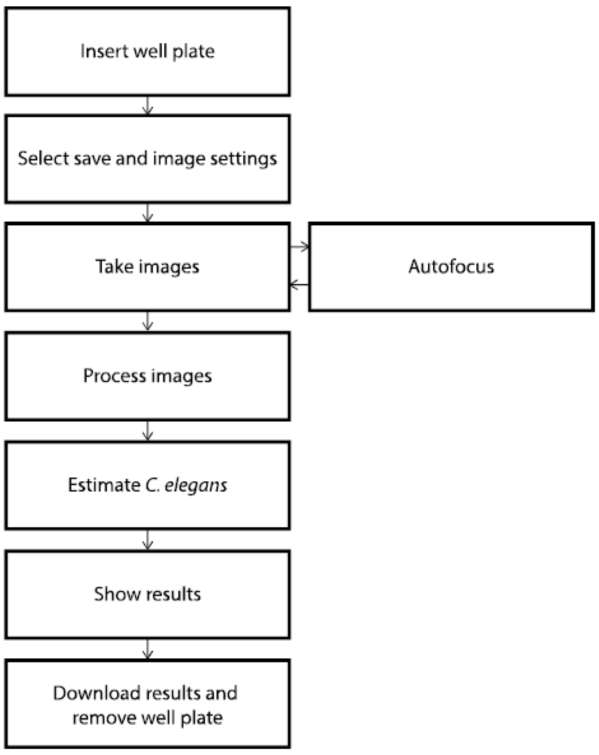


FIGURE 29 Realization of concept variation I

TABLE 10 Co-creation matrix results

	Must have	**Nice to have**
Implement now	The user can clean the machine parts in contact with the well plate with ethanol. The user can insert a 48 well plate. The user can activate capture mode. During capture mode, the product makes an image of each well of the 48 well plate. *The user can adjust the number of photos taken from each well before enabling capture mode.* The device automatically labels images according to the well plate index. After capture mode, the device automatically starts processing mode. During processing mode, the product gives the user a prediction of the amount of *C. elegans* in each image. During processing mode, the product gives the user a prediction of the amount of *C. elegans* in larval stage four in each image. After processing mode, the device provides an overview for the user of all the recorded images and predictions. Allows the user to save and store the data on an external hard drive. It does not matter if there is a delay between the capture and processing mode.	The user can select and make different traveling paths for capture mode. The user can change the diaphragm to 10/11/12/13 mm. Multiple microplates can be screened in one go.
Implement in the feature	In the future, the device can support 24/96 well plates. In the future, the device can also be applied for microplates with liquid medium In the future, the device can support other light sources. In the future, the user can select and make different traveling paths for capture mode. In the future, the device uses more *C. elegans* parameters to estimate the life stages.	A website lists all the documentation of the hardware and software.



It is interesting to note that users only interact with three out of eight steps described in the flowchart. This implies that the screening devices automates the majority of the tasks. However, to make the product useful, users need to be aware of the process steps and feel that they are in control. This inherent conflict between control and automation can be solved by applying feedforward and feedback. For instance, a process bars can show the current process steps and upcoming steps.

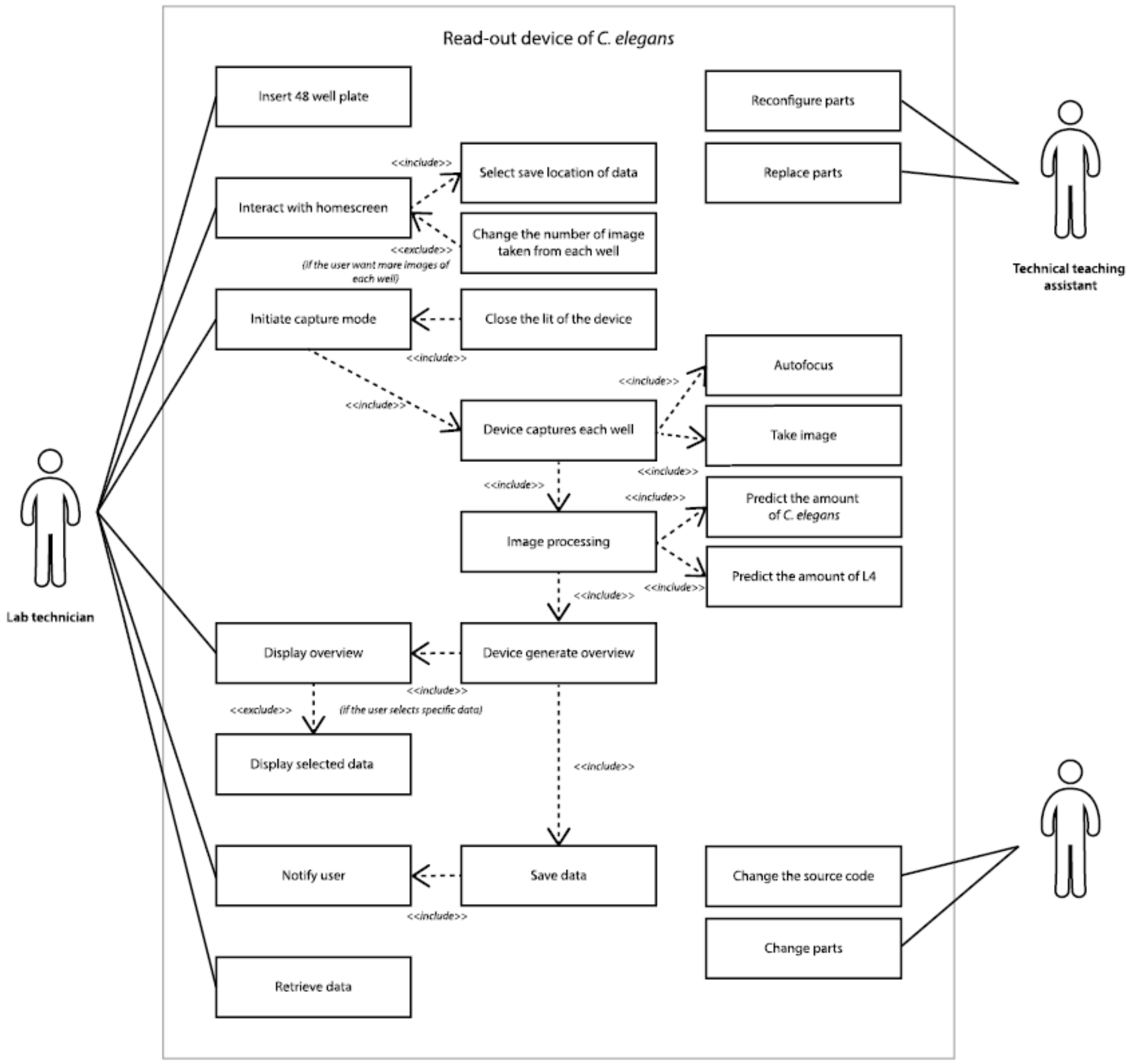


FIGURE 12 Use case diagram

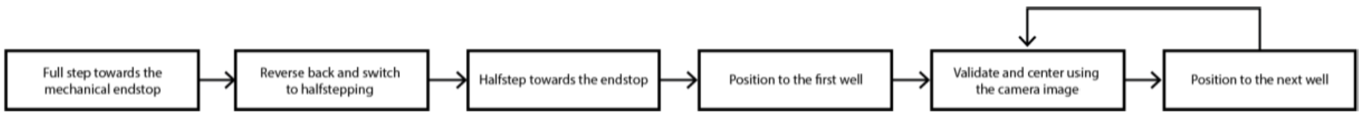

Full step towards the mechanical endstop
Reverse back and switch to halfstepping
Halfstep towards the endstop
Position to the first well
Validate and center using the camera image
Position to the next well

FIGURE 41 Positioning system block diagram