# Component Design

Designing the architecture is an iterative process. When we start modeling the components, we might have to come back and revise the architecture. The output of the component design might be something like slide 4.

For the sub-activities of this activity, see slide 5.

We now have problem domain, application domain, and we have specified our general component architecture. Now we are going to specify the architecture.

The model component is the part of the system that implements the problem domain model.

Here, we represent events and classes, structures and attributes. We also choose the simplest way we can do this. The result of this sub-activity is a class diagram of the model component seen on slide 10.

Activities: We represent the private events of our different classes. There are two kinds: sequence and selection events marked with a +, and iteration events marked with a \*.

Then we represent common (public) events. We are looking for the simplest possible representation, so choose between the alternatives you see.

We then restructure our structure of the problem domain using generalization, association and looking for embedded iteration. This is only really possible if we’ve made changes.

The Bank System

If we look at the event table, we have both private and common events, and sequential and iterative events.

## Private events

**Sequence and selection events** are represented as an attribute in the class described in the state chart diagram. We assign the attribute a value when the event is triggered, and we integrate that attribute in the class (we can do this because we know the event will only occur once).

**Iterative events** are represented as new classes connected to the class described in the state chart diagram with an aggregation structure. When the event occurs, a new object of that class is instantiated.

Again, integrate the attributes of the event in the class.

If the event doesn’t contain any attributes, consider not making a new class. The argument is that the new class doesn’t contain any information, so there is no need to create objects to represent the event. This is applicable for “updated” events.

**(Question)** How do we actually code this?

An iterative event can happen multiple times, and we want to keep track of this. If it was just an attribute, we would lose all previous events. When modeled as a class and objects, we will always have previous information.

See how the private events from the Customer class modeled on slide 15.

## Common events

Consider structural connections between objects to model access to attributes.

If the event has a + and a \* on the same line (different classes), we represent it with connection to the class, which gives the simplest representation.

If it has the same sign across all classes in a line, consider other representations.

See slide 17. The account opened has a \* for Customer, but a + for Account.

The simplest way to do this is to give the Account class an attribute called open-date and close-date. This is the simplest representation.

When the event is iterative for all involved objects, then the event can be represented as the new class(es) under Account on slide 18. We could also choose to model it under Customer, then it would look like slide 19. See that this is more complicated than slide 18.

## Restructure classes

The class diagram can often be restructured and simplified without any loss of information.

We don’t really need two classes Deposit and Withdraw, they can be combined into a Transaction class. Simplify.

Another way is to look for associations. In a gas station, we need to keep track of Customers filling their cars with gas. To model the event that the customer fills the car with gas, this is iterative for both classes, so we need to add another class, which aggregates to both classes. Then we can actually remove the association.

**Embedded iterations**

See slide 22. We have three events aggregated by Person, which are events modeled as classes. This does not represent the association between the events. You cannot receive treatment after discharge. In cases like this, we choose to move on to hierarchical structure instead. See how the hierarchy makes much more sense as to what is actually happening. The discharge class is modeled as an attribute in the hospitalization class now.

## 

## 

## Group assignment

We are doing the group assignment in the lecture. See slide 23 for the class diagram.

The Movie Played event is common and has different signs across classes. This can be represented as an attribute for the Show class, seeing that this is where the event is sequential. The attributes are seen in the attributes for the event itself in the behavioural diagram.

Same thing goes for Show Planned.

Customer closed and opened are both attributes for the Customer class.

Movie Rated can be a new class aggregated by Customer and Movie, that has attributes rate and date. The Movie Viewed event could be a new class, but it’s way more simple to make it an attribute “view date” in the new Rating class. This is given that we only want to keep track of ratings, and not when people watch the movies. If it was the other way around, we could make a new class Viewed with a rating attribute.

Cinema opened and closed are both attributes.

## Exam example

For class 1, we have a private sequential event H1 with attribute A1, so the class should have the attribute A1. Furthermore, it should have an attribute called C1State, as we want to keep track of the states of the class, for example when it is destroyed.

For class 2, it should have attribute A4, as well as a state attribute. For H2, it is iterative for both C1 and C2, and as such should be made into a fourth class aggregated by C1 and C2, and should have attributes for the state and for A2.

H6 is iterative for C2 only, and should be made into a new class aggregated by C2. It should have attributes state and A6.

Note that H5 is iterative for C2, and sequential for C3. We give C3 the attribute A5, but as it is iterative for C2, we need to show this connection between the two classes. We do this by letting C2 aggregate C3, which is the class that has the attribute it needs.

For class 3, we want a state attribute. Furthermore, it should have A5, A9, and A10 as attributes.

**Always remember multiplicity when doing exam exercises like this.**

# Exercises