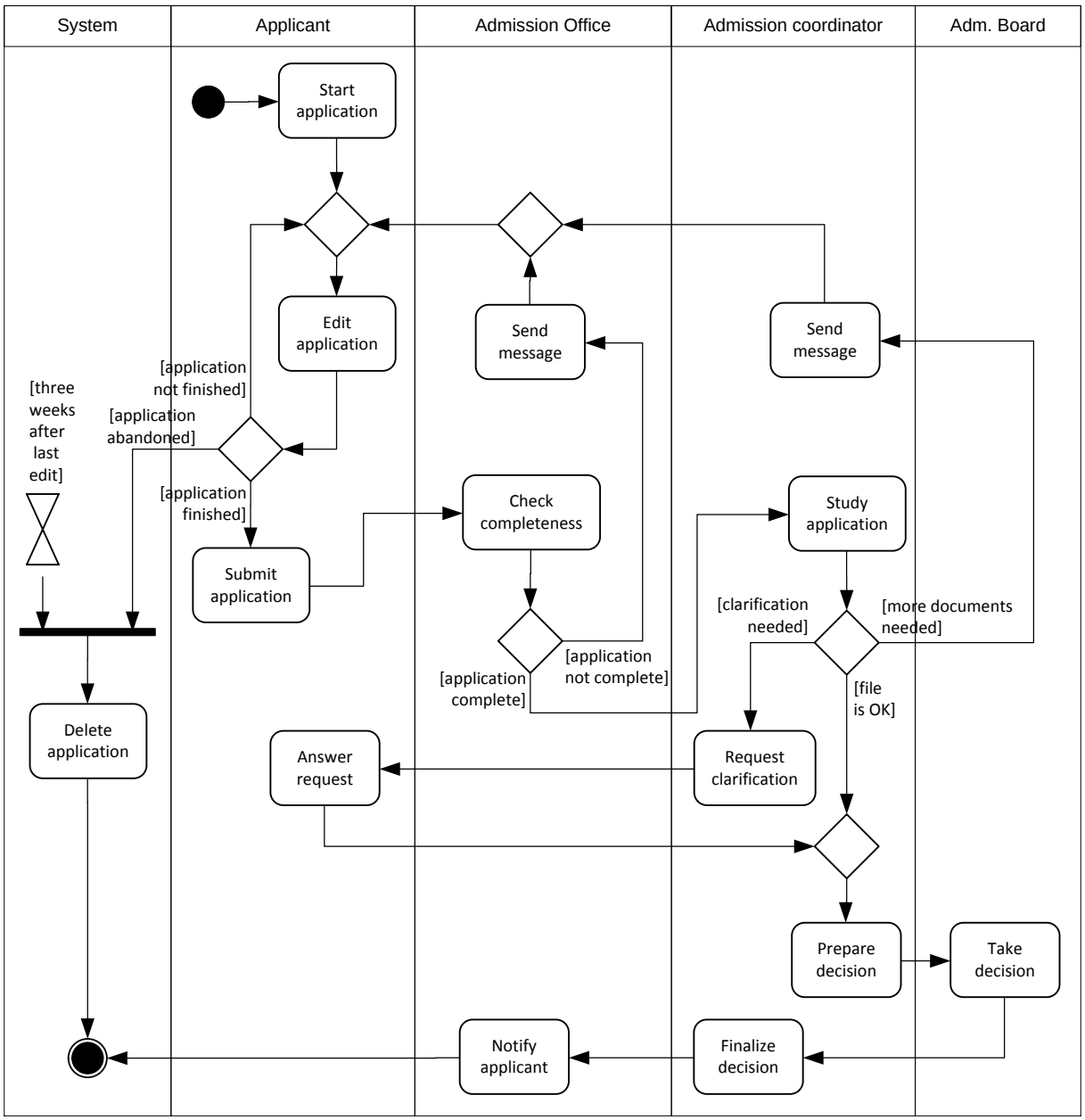
Solutions to the Design Test, 10 December 2015, 8:45–11:45

1. Activity Diagram



Remarks:

- The above solution has some three-way branches. You can model these as a single branch with three exits (as above) or as a combination of two binary branches. These are equivalent
- When the admission office or the admission coordinator sends a message, asking for additional information, the status of the application is also changed. This is not represented in the diagram (it would be represented in a state machine), but it is okay of you explicitly included this.

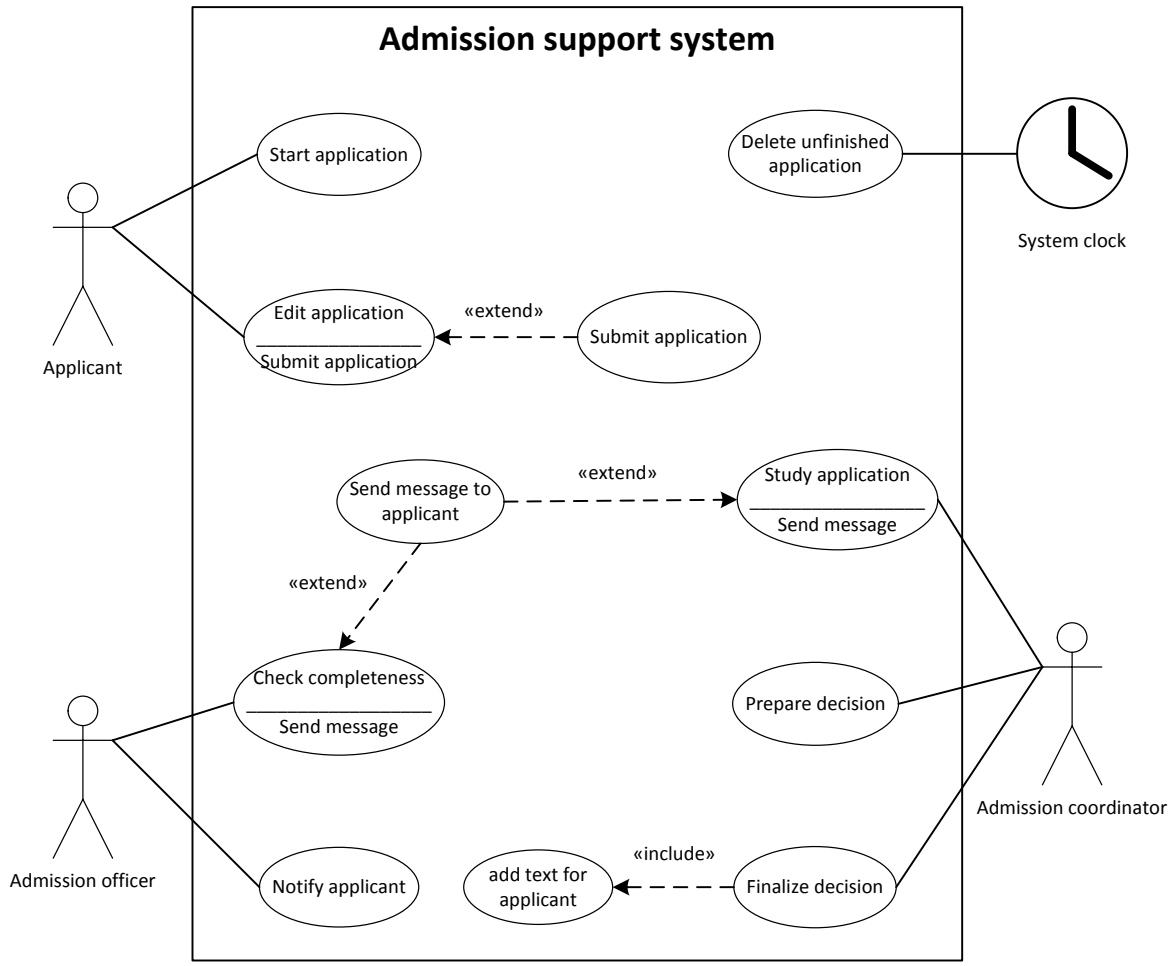
2a. Actor list

Actor	***Description***
Applicant	A person who seeks admission to a study program
Admission office(r)	(A person working in) the central office of the University that handles the registration of admission applications
Admission coordinator	The person who assists the *admission board* of the program to which the applicant seeks admission with preparing and finalizing decisions
System clock	An automatic trigger to delete unfinished applications when they have not been edited for three weeks

Remarks:

- The actor list should be consistent with the Use Case Diagram. The case description does not mention the Admission board as an actor (there are no use cases for it), but it is reasonable to assume that board members at least can read documents, so it is not wrong to include it in the list.

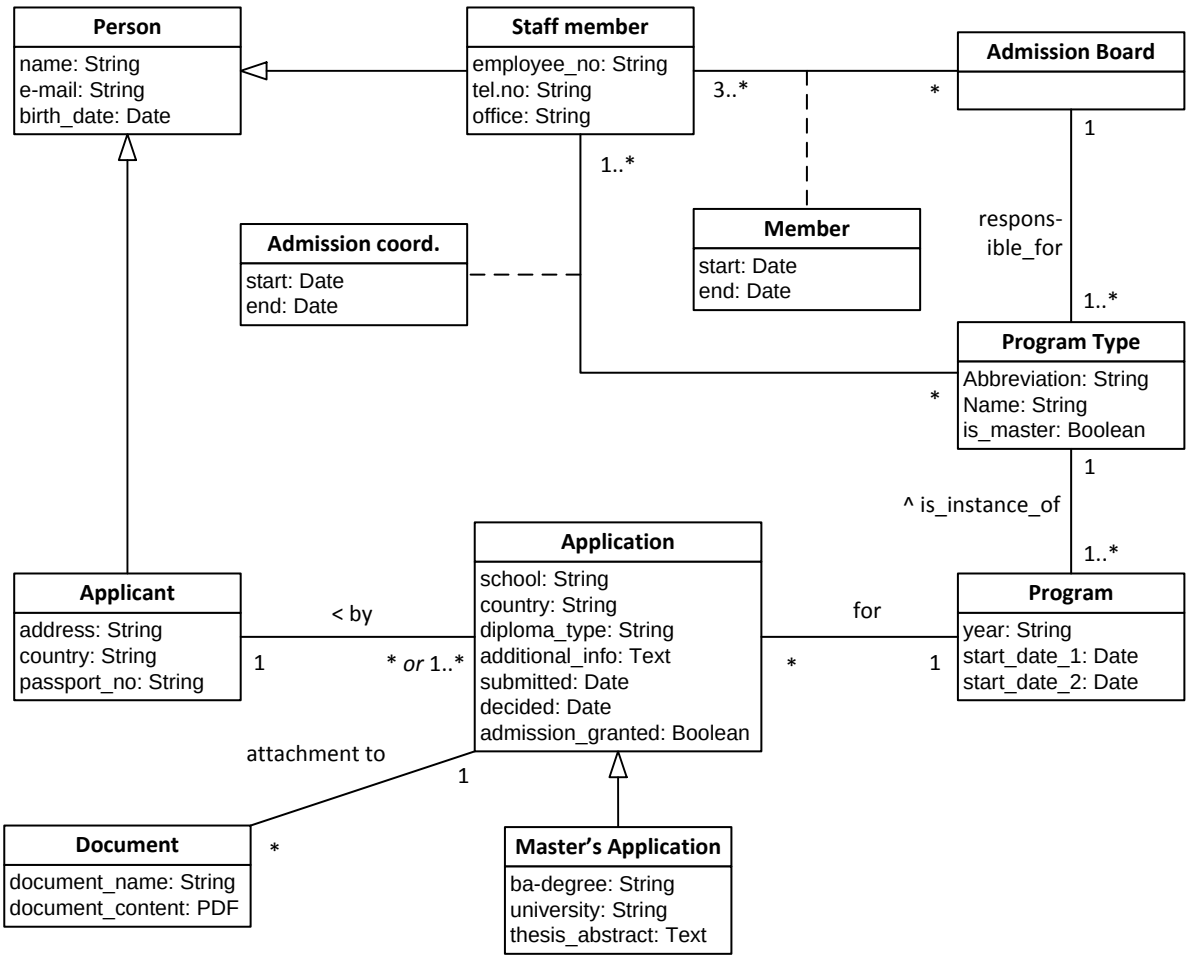
2b. Use Case Diagram



Remarks:

- "Submit application" could also be modelled as a separate use case, but even then it should be an extension to "Edit application".
- Any icon is OK as a representation for system clock. (There is no example in the lecture slides and lab exercises.)

3. Class Diagram

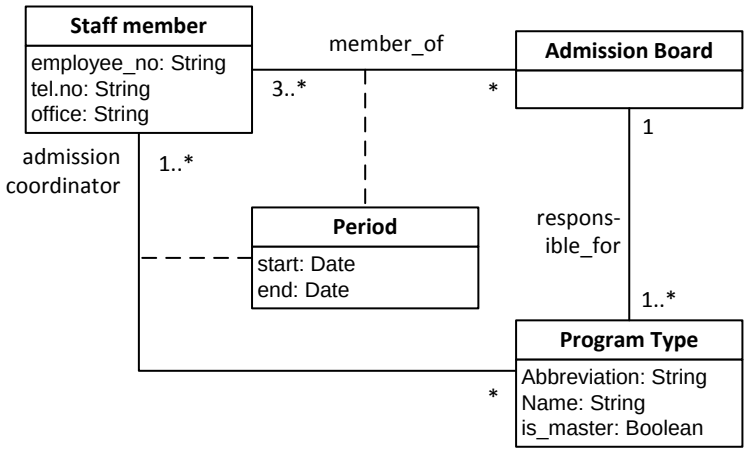


Remarks:

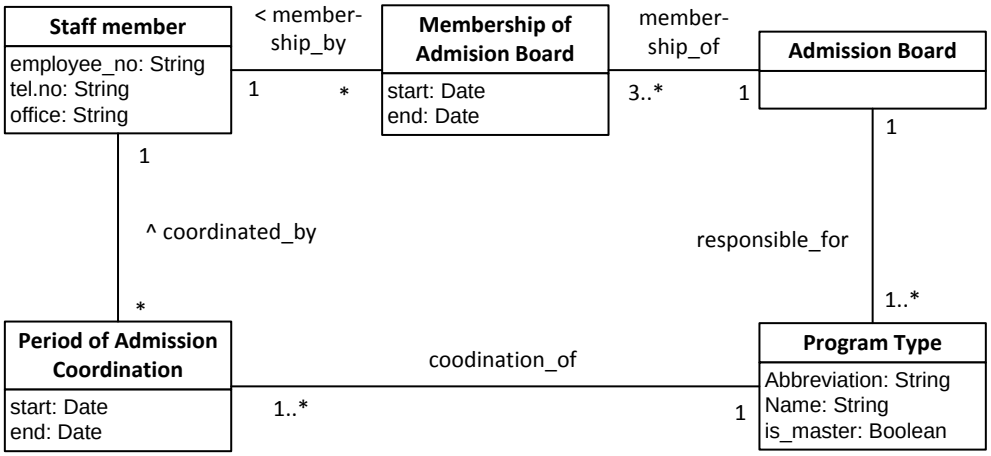
- "Application" could have subclasses "Bachelor's application" and "Master's application" – but the former has no attributes and associations, therefore it can be discarded.

- In some cases different variants of multiplicities are OK. It it necessary that an applicant is associated with at least one application? Probably yes (implying that the applicant information is deleted when the application is deleted; probably true but not mentioned in the text). But it is of marginal significance for the model.
  Note that an Admission Board has at least 3 members, reflected in the mulitiplicity "3..*". Other multiplicities like "*" are also considered correct.

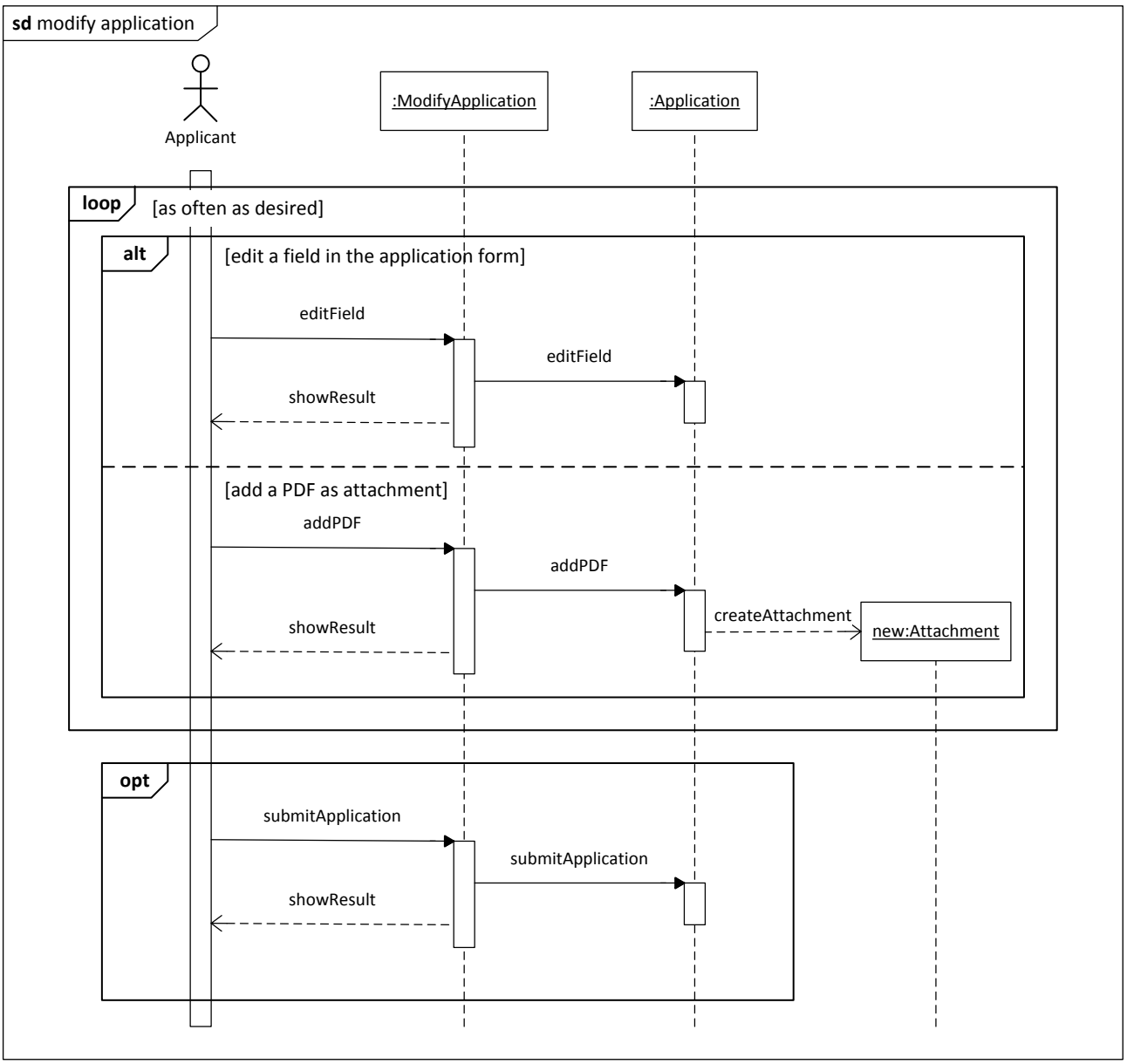
- The period for which a staff member serves on admission board is a typical case for an association class (identified by the combination of a staff member and an admission board).
  Similarly, the period for which a staff member is admission coordinator for a program (type), can be modelled as an association class.
  Note that these two classes have exacly the same attributes. The can be merged in to one class, as shown below. In that case, however, the assocations should have names, because the assocation class "Period" does not specify the nature of the association:



- In the above solutions it is assumed that someone will not serve as admission coordinator multiple times, for different periods, with breaks in between. Similar for member of an admission board. However, if you do want to take these possibilities into account, this can be modelled as follows.



4. Sequence Diagram



Remarks:

- Instead of **alt** you could also have two times **opt** within the loop. What counts is that these two actions can be done any number of times in any order.

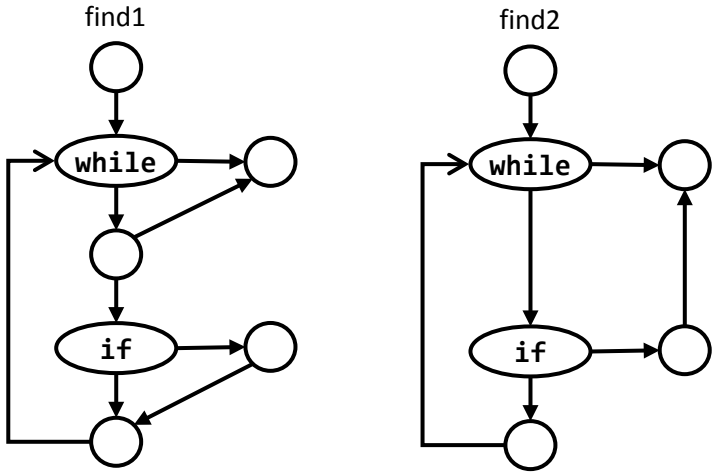
5a (Software Metrics: Coupling)

1. a. Both couplings will typically decrease. The affarent coupling decreases because where possible other classes will now depend only on A instead of on both C1 and C2. The effarent coupling decreases because the functionality of C1 (and C2) is split with A, and so they both receive part of the overall efferent coupling.

2. b. If you call the method of another class directly, that class counts towards your efferent coupling (you depend on that class). On the other hand, if you register as listener to an observable, you still depend on that class because you call its addObserver method, so the efferent coupling does not change.
   But now the observable calls back one of your methods. Does this increase your afferent coupling? No, in fact it doesn't, as (due to abstraction, see above) the observable does not know whose method he is calling; there is therefore no dependency.
   In conclusion, use of the observer pattern does not change the coupling.

5b (Software Metrics: Cyclometric complexity)

1. We have seen various methods to compute the cyclometric complexity. All of them to some degree depend on the *flow graph* of the method. The flow graphs for the methods above are as follows:



   The **`while`**-condition in the first flow graph is split in two to faithfully reflect the behavior of the (conditional) &&. The complexity can for instance be computed as `#edges - #nodes + 2`; this yields

   - `find1: 9 – 7 + 2 = 4`
   - `find2: 7 – 6 + 2 = 3`

   The calculation as `#decisions + 1` yields the same outcome.

2. In `find1`, the **`while`**-condition has to recompute whether the previous instance of the loop went into the `then`-part of the **`if`**-statement. This causes additional complexity.