Module Feedback:

Formal Software Specification (UML Modeling)

Team 4:
Michel van der Hoek
Leoni Hurkx
Victor Versteeg
Benjamin Voss
Ralph Zoontjens

Feedback on our spec and the implementation of the other team

In general, the implementation as acted out was helpful, even refreshing, in showing us how our system would work. The lion's share was done like we intended it to be, although there was no creativity in acting out scenarios we had not generated. But visualizing it in this way did clarify some discrepancies between specification and implementation, which we will point out and elaborate further on here.

The comment was that a male as well as a female animal can give birth. This was not our intention, but we agree it was not specified clearly. We should have made an activity diagram involving two animals who are about to mate, containing the condition which animal is female. This animal would then have the method *giveBirth()* and the attribute *isPregnant*. Because a male and female animal have different behaviors, we should have also specified this in the class diagram, by introducing the abstract classes *Male* and *Female*. Also, apparently the condition for the *Dance()* method (an animal starts dancing if his stomach is full) was not clear to the implementation team.

There were some notational errors and inconsistencies, probably in all diagrams, which, however, did not cause major confusion of the implementation team, as they just acted out our sequence diagram. Had they also thought up new scenarios, more of these errors would have come to light.

Feedback on the other team's spec and our implementation

Unraveling all workings of the system was a big puzzle for us, partly due to our lack of experience in reading diagrams, part to inconsistencies between diagrams and actions that appeared unnatural to us. The latter are, for example, having a pet handing out a list of pets in the zoo (logically, if there were no pets at all, the list could not be handed over, obstructing the user from selecting a pet from the list), and having a pet attracting the person's attention instead of the person actively looking for it.

Concerning the use case diagram, we can say that a system boundary was lacking, although for this system this was no real problem.

The class diagram showed some dynamic relations (e.g. a person *can* exchange a pet with another person), which we consider irrelevant in a static diagram.

Critiques on the sequence diagram are, firstly, that concrete objects should be indicated, not just abstract classes. A class cannot do anything, only an instance can. An instance of the *Animal* class was used, instead of *Dog* and *Cat*. Secondly, some arrows were placed incorrectly. This caused confusion, which our 'Acting out' session clarified. It should be clear in the sequence what method of which object is invoked by which object. Finally, the origin of the object *system* was unclear to us as it was not declared in the class diagram. We interpreted that *system* was an instance of the class *Zoo*, which appeared not to be correct.

The state diagram has an excellent division between sub- and superstates. It can be mentioned, however, that some things could be improved. For instance, you have to be able to return to the previous state. In this diagram, you could not leave the shop without selecting a pet. Exchanging pets was not very clear too. There should have been more conditions in this diagram. We did make a misinterpretation of having a shop out of the zoo, while it was specified as being in the zoo. As these formal methods are not yet very natural to us, it is apparently hard not to make intuitive decisions, but stick to what is there in the diagram.

When it comes to the several activity diagrams we received, we can say that they were a bit confusing, as they seemed to be a mix with state diagrams. They should have been constructed while taking into account the perspective of the system more.

Feedback on the module in general

An important thing we noticed is that when using UML, you should have a clear structure. The given time period was short to find this, resulting in, among others, inconsistencies between diagrams. However, we believe this would improve greatly if we had the time to iterate the process. Hence we consider the spiral model very important.

Belief was varied among team members about if they would use UML in their future work. Of course, learning and using formal methods is hard, and takes much time, so in most projects it would not be necessary to use. However, if cooperating with somebody who masters UML, or to simplify complexity, these diagrams would be very explicative. It is also another way to make ideas explicit, rendering the methods useful in quickly 'sketching' parts of systems.

Acting out is a nice method. As mentioned before, it makes a system visible by letting it run not on a virtual machine, but a real one. Apart from the advantage of more efficient development (you do not have to program) there is the one that designers can better imagine the working of the system by 'being' it. And, of course, it is nice to do and greatly spices up a design process. Our suggestion is that the method could be used right from the start instead of only for verification.