UML in Action: Teaching Design students Formal Methods

Jun Hu[1] and Philip Ross[1] and Yuechen Qian[2] and Loe Feijs[1]

[1] Eindhoven University of Technology, 5600MB Eindhoven, NL
[2] Philips Research, 5656AE Eindhoven, NL

Abstract. When designing product behavior, the designer often needs to communicate to experts in computer software and protocols. In present-day software engineering, formal specification methods such as the Universal Modeling Language have been widely accepted. Teaching design students these formal methods is non-trivial because most of design students often have difficulties in programming the behaviors of complex produces and systems. Instead of programming, this paper presents a technique, namely "acting-out", for design students to master the formal methods. The experience shows that acting-out not only worked out very well as a teaching technique, but also showed the potential for bridging the processes of industrial design and software engineering.

1 Introduction

1.1 UML in Industrial Design Teaching

A product with sensors, actuators and network connections can offer an interesting, useful, or playful behavior to its users and to the other products, systems and services to which it is connected. The master students from the department of industrial design (ID), Eindhoven University of Technology (TU/e) take responsibility for the creation of this behavior. In many cases. if the product isn't stand-alone, neither is the designer. Whenever product behavior is realized through computer software and protocols, the designer often needs to communicate to experts in these matters. In software design, formal methods are often used for this purpose [1–3]. In present-day software engineering, formal specification methods such as the Universal Modeling Language (UML) [4] have been widely accepted. It contains "activity diagrams", "use case diagrams", "class diagrams", "state charts" and "message sequence charts" that are useful for product designers and software engineers to exchange their ideas and concepts to reach a common understanding. A module called "Formal Software Specification in Action" is now taught at ID, TU/e to help the students understand and use such formal methods for this purpose.

The knowledge and skills that students gain by participating in this module helps them to express the structure and behavior of the software components of their design in a way that is understandable to other parties.

The students develop an understanding and appreciation of what it means to master complexity through formal methods. The scope is widened from small programs to complex software systems. Although developing and maintaining such systems usually involves computer scientists as well, the ID Master students are expected to be well equipped to use formal methods such as UML and thus specify system structure and desired behavior.

UML as a formal specification tool is widely used in an object-oriented design process in software engineering. Such a process can follow a traditional waterfall model going through object-oriented analysis (OOA), object-oriented design (OOD) and object-oriented programming (OOP) [5], testing and delivery. It can also following a spiral process that includes fast iterations, each of which going through analysis, design, implementation and evaluation. The later is more and more used in software industrial, often being referred as rapid prototyping, or in more recent buzz word, Agile design [6].

Either method has a phase of implementation or iterations of implementations so that the ideas and concepts under design can be visualized and tested. This often requires the designers or the engineers to have enough programming skills to implement the system (waterfall model) or to prototype the behavior (agile design). In a teaching course of such methods, students with no experience of programming would find it difficult to fully understand the entire process.

This is exactly the difficulty one may expect in teaching industrial design students a formal specification language such as UML. Most of the students in an industrial design education are not trained extensively in programming tasks, neither do they need to. They are not software engineers, they are designers. But as a designer who is expected to design "intelligent products and services", understanding of and communicating about the structure and the behavior of a complex system are certainly unavoidable. To deal with this dilemma, instead of pushing extensive programming courses to the design students, we need to find a method that can be better integrated in an industrial design (not software design) process. Hence a technique called "Acting-out" was proposed and tried out in our module for the master students at ID, TU/e.

1.2 Acting-out as a design technique

In design of interactive, networked products and systems the intended interaction experience often emerges as a main design criterion. "Traditional" design techniques, like storyboards, or on screen simulations seemed unfit to deal with the multifaceted and dynamic character of interaction experience. Several design and design research communities have developed approaches to deal with this complexity in different stages of the design process, based on introducing real, bodily involvement in the design process. Different communities gave different names to these approaches, e.g. "Experience Prototyping" at IDEO [7], Informance Design [8], Designing actions before product [9], Choreography of Interaction [10], Gesturing out [11]. In this paper, we call these approaches "acting-out"-approaches. These approaches have in common that they allow designers to make aspects of a product or system experiential and vivid

by physically acting out (elements of) interaction scenarios. Buchenau et al. [7] describe the following advantages of acting out techniques: understanding existing interaction experiences and contexts, evaluating design ideas, exploring new design ideas, and communicating designs to an audience.

These last three advantages, i.e., evaluation, exploration and communication in the design process, make acting out especially valuable in the Industial Design UML module. UML is a highly abstract language, but it is a tool that, in Industrial Design practice, referes to real products and systems that make real life experiences happen. Our assumption in this module is that through acting out a concept UML diagram, its structures, objects, properties, connections and restraints gain an experiential dimension in an early stage. This experiential dimension could help identify possible inconsistencies and flaws in the UML diagrams and help suggest improvements. Furthermore, it may help communicating to other designers or software engineers what the problems or ideas for improvement are.

In the next section, the module is presented, followed by the feedback from the students.

2 Zoo of Tamagotchi animals

The module was a one-week (5 full days) course for 32 master students, including morning sessions for lectures and afternoon sessions for a project. The lectures gave introductory information about software engineering and formal methods in general, object-oriented design process, selected UML diagrams for the project and acting-out as a design approach. The students were expected to use what they learnt from the lectures in their projects, going through a single process of analysis, design and acting-out.

2.1 The project

Students were divided into five teams (about six students each). Each team received the same task: Designing a system called Zoo of Tamagochi Animals:

- The first half of the project was for analyzing the requirements, making and specifying the object-oriented design of the system;
- In the middle of the project, the teams swapped their specifications for acting-out each other's specifications;
- The second half of the project was for acting-out the specifications received.

The students were asked to "implement" the system by acting the specification out to show how the system should work according to the specification. Students could play the objects, showing their behaviors and the communication in between. Stage props could also be used to represent objects, interfaces, and events, etc. Students were asked to use imagination and creativity in acting-out, since we did not have experience in acting-out UML in a design process.

2.2 Requirements

A preliminary description of the Tamagochi Zoo requirements was handed out at the beginning of the module. Students ware asked to design the object-oriented structure of the system - Zoo of Tamagochi Animals:

- *All animals live in a zoo.*
- *An animal has got a life after it is born. While the life goes on, the animal moves and sleeps (if it is still alive), eats (if it moves and finds food), grows (if it eats) until one day, it dies because of hunger, illness or age.*
- *Every animal has got a body. Different animals look different because of their different bodies. Every animal has got two eyes and one mouth on its body. When eyes are pinched, animals scream and can be hurt.*
- *There are male and female animals. When grownup males and females meet, they may fall into love and the love may result in baby animals.*
- *Some animals are pets. Pets have names and they wear their name plates on their bodies. People (the users) take care of their pets and feed them with food.*
- *People may play with their pets to keep their pets fit.*
- *Some baby animals will be selected by people and they become pets. The rest are left free in the zoo and they have to strive for food and try to survive by themselves.*
- *When a zoo is created, it is empty, until people get some animals from somewhere else (from shops, for example).*
- *People may exchange their pet animals.*
- *Dogs and cats will be our favorite animals for the time being. Dogs woof and cats meow. Cats are scared of dogs and can be bitten by dogs. When big dogs bark, cats scream and start running away. When big dogs fall into sleep, cats start getting together and partying.*
- *The zoo is open for other animals, including unknown ones.*

2.3 Results

As mentioned before, during the second half of the project the teams swapped their specifications for "acting out". They were not allowed to consult the teams that made the specification about the design. The teams were only allowed to read the specifications in order to understand the design. At the end of the project, each team presented (acted out) the specification together as a theater play (Fig. 1). After that students were asked to reflect on the specifications they got from other teams, their own specifications, and their implementations (acting-out) for other teams. Here an example is included as follows:

(a) Implementing the *Pet* interface on a *Dog*

(b) Feeding a *Croc* only if it is a *Pet*

Fig. 1. Acting-out

Feedback of Team 4 on Team 2's specification

. . .

The state diagram (Fig. 2) 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.

. . .

Team 2's reaction on Team 4's feedback and acting out

. . .

In general, the implementation as acted out was helpful, even refreshing, in showing us how our system would work. 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.*

. . .

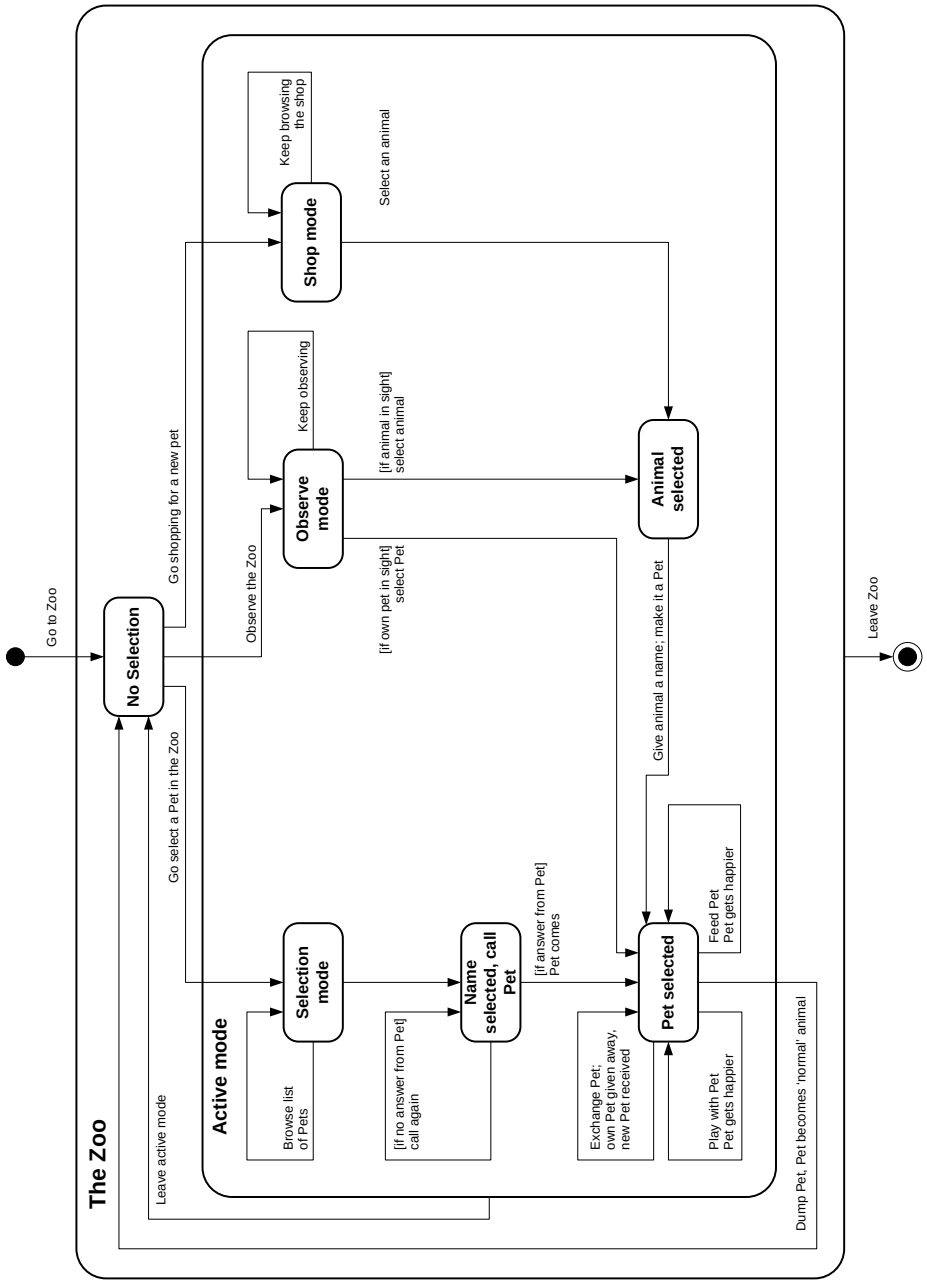

Fig. 2. State diagram of the Zoo, designed by Team 4

2.4 Students' reflections on acting-out

- We suggest for the next module to plan the acting out method earlier in the process to use it as inspiration for thinking of new interactions or for in between validation of the UML diagrams rather then as an end stage. ··· Overall a proper module; good introduction into UML diagrams and acting out as a design (validation) method. – *Team 2 (Thomas Visser, Michel Peeters, Carl Megens, Marcel Verbunt, Fabian Koopman, Sander Kruitwagen)*
- ··· The acting out of the other group's specification was a nice way to end the module, and made it easier for us to understand where the flaws were in the system. This acting could also be a good way to remove bugs from the diagrams earlier on in the module. – *Team 5 (Sibrecht Bouwstra, Sander Kouwenberg, Bart-Jan van Putten, Jos Verbeek, Alice Verdonk)*
- 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. – *Team 4 (Michel van der Hoek; Leoni Hurkx; Victor Versteeg; Benjamin Voss; Ralph Zoontjens)*
- We think the acting out method could be very useful in the beginning of the module, where interpretation, expression play a more important role. Secondly, it would be handy to first act out the logic in the diagram, and after that write the diagram down. We discovered most flaws by reasoning, but acting out was a good way to communicate flaws in a diagram to the audience. – *Team 3 (Pei-Yin Chao, Bas Groenendaal, John Helmes, Philip Mendels, Rik Wesselink, Mehmet Yalvac)*

3 Concluding remarks

The method of acting out the UML diagrams instead of implementing them using programming worked remarkably well. Within one week of time, students mastered the basic principles and methods of object-oriented design, and UML as a formal specification language. Without involving the students in heavy programming activities, this module gives more time and space for the students to concentrate on the essential issues and disciplines.

Acting-out seems to be an effective and enjoyable method for students to learn formal specification methods. Furthermore, we are looking for methods to integrate industrial design processes with software design processes in designing "intelligent products and services", and we argue an acting-out design approach may provide a good bridge in between. However, establishing an acting-out based method for this bridging purpose, requires more research and experiences in intelligent product and systems design practice .

References

1. Hu, J.: Design of a Distributed Architecture for Enriching Media Experience in Home Theaters. Technische Universiteit Eindhoven (2006) ISBN:90-386-2678-8.
2. Feijs, L., Hu, J.: Component-wise mapping of media-needs to a distributed presentation environment. In: The 28th Annual International Computer Software and Applications Conference (COMPSAC 2004), Hong Kong, China, IEEE Computer Society (2004) 250–257 ISBN0-7695-2209-2, DOI:10.1109/CMPSAC.2004.1342840.
3. Feijs, L.M.G., Qian, Y.: Component algebra. Science of Computer Programming **42**(2–3) (2002) 173–228
4. Booch, G., Rumbaugh, J., Jacobson, I.: Unified Modeling Language for Object-Oriented Development (Version 0.9a Addendum). RATIONAL Software Corporation (1996)
5. Taylor, D.: Object-Oriented Technology: A Manager's Guide. Addison Wesley (1990)
6. Martin, R.C.: Agile Software Development: Principles, Patterns, and Practices. Prentice Hall (2002)
7. Buchenau, M., Fulton Suri, J.: Experience prototyping. In: Designing interactive systems: processes, practices, methods, and techniques., New York, ACM Press (2000) 424–433
8. Burns, C., Dishman, E., W., V., Lassiter, B.: Actors, hairdos & videotape-informance design. In: CHI, New York, ACM Press (1994) 119–120
9. Buur, J., Vedel Jensen, M., Djajadiningrat, T.: Hands-only scenarios and video action walls: novel methods for tangible user interaction design. In: DIS, New York, ACM Press (2004) 185–192
10. Klooster, S., Overbeeke, C.: Designing products as an integral part of choreography of interaction : The product's form as an integral part of movement. In: the 1st European workshop on Design and Semantics of Form and Movement, Newcastle, UK (2005) 23–55
11. Ross, P., Keyson, D.: The case of sculpting atmospheres: towards design principles for expressive tangible interaction in control of ambient systems. Journal of Personal and Ubiquitous Computing (In press) DOI:10.1007/s00779-005-0062-3.