INTERNATIONAL CONFERENCE ON ENGINEERING AND PRODUCT DESIGN EDUCATION
10-11 SEPTEMBER 2020, VIA DESIGN, VIA UNIVERSITY COLLEGE, HERNING, DENMARK

PROTOTYPING WITH A TEAM: ACTING MACHINES SUPPORT SHARED UNDERSTANDING

Aadjan VAN DER HELM and Pieter Jan STAPPERS
Faculty of Industrial Design Engineering, Delft University of Technology, The Netherlands

ABSTRACT
It is a great challenge for design students to learn to work with interaction and interactive technology. They need to learn about interactive prototypes, to work in teams as well as many other things, and within a limited time. In this paper we identify challenges related to maintaining a shared understanding of concept and implementation and propose a technique to keep teams aligned in the process. Existing conceptualisation tools, such as scenarios and storyboards provide little help to plan to code the control flow and algorithms for sensing the user and effecting feedback. User-friendly coding tools and technology as well as many online example projects are available for students to experiment with interactive behaviour. They are encouraged to jump in at the deep end, puzzling together existing code pieces without any planning. As a result, we found that student's resort to trial and error problem solving. This often leaves them with spaghetti code which makes it difficult to understand and improve the quality of the interactive behaviour in subsequent design iterations. Typically, student teams divide who is working on the concept and who is dealing with the code. The Acting Machine technique combines the language of State Machines with enacting techniques, and allows designers to first define and run through interaction scenarios using a visual representation that expresses the core design concept, allows for engagement of the whole team, and has a straightforward connection to implemented code. In this paper we describe the technique and discuss our experiences with it in MSc courses on Interaction Design.

Keywords: Interaction design education, interactive prototyping, programming, state machines

1 INTRODUCTION

The learning curve for designing and constructing interactive prototypes starts with a steep ramp. Prototyping requires knowledge from several disciplines such as design, computer science, electronics and systems engineering. In design education, there is no time to turn the students into experts in all of these fields, yet prototyping is considered an essential skill for designers [1, 2]. Prototyping involves the creation of user experiences with technology to evaluate design assumption with the intend to fail early on the wrong assumptions and to iterate forward to arrive at a successful design. To experience this in education, students typically have to work in teams, design and construct multiple prototypes for one project, learn about technology on-the-fly while at the same time managing course deadlines for presentations to coaches and other stakeholders.

Students collaborate as a multi-disciplinary team engaging with all tasks from user research to engineering the prototype. In our experience over more than 10 years [3-5], students tend to stay in their "comfort zone" to optimize efficient production of working prototypes. They choose to distribute the tasks in the team based on present skills rather than learning challenge. Often, they choose to collaboratively brainstorm about the user experience (they are still all design students). Once defined, the students switch to distributing the workload, tasking individual members or a sub-team with the engineering of the prototype behaviour. When these team members are new to the tasks of selecting components and programming, they tend to jump straight into coding, skipping designing the program structure, and losing themselves in cumbersome debugging of ill-structured and undocumented code, whose workings are understood by at most one team member. During this struggle, the technology sub-team solves technical details by making decisions which affect the overall user experience of the prototype. Sharing this with the rest of the team is difficult because there is no shared representation to support discussions. Once the prototypes were fixed and patched to a working state, mal-formed code

made it difficult to conceive or implement a next iteration. In summary, after the brainstorm, the team lacks a shared model of what they are trying to build.
In this paper we present a technique called “Acting Machines” that enables a team to collaboratively develop an interactive prototype. It is tied into the practise of design, offers a step-by-step approach that starts from an intended user experience, supports the definition, exploration, and development of corresponding system behaviour, and maintains a link to the software code that realizes the behaviour. Central to the technique is the Acting Machine Diagram, a visual representation of the programmed interactions, which supports shared understanding linking the conceptual stages with the implementation stages of prototyping.

2 PRIOR RESEARCH

During prototyping, a designer continuously shifts attention between detailed activities such as writing software and connecting sensors (zooming in) and the overall concept, goal, and development process (zooming out). When executed with a team the zooming out involves communication to arrive at a shared understanding what the best next steps are. From the field of human-centred design several techniques are available to support obtaining a shared understanding about the user experience [6]. Methods such as Interaction Scenarios and Storyboards can be used to communicate about how the interaction between user and the design unfolds over time. From the field of engineering visual notion techniques exist that support the communication of implementation of the prototype, e.g. State Machine Diagrams (STD) and Flow Charts [7, 8]. STDs have a very simple visual language, offer a model of execution that can readily be transcribed into program code and seem to be most suitable for modelling interactive prototype behaviour [9-11]. While Flow Charts are useful and have wide adoption in education and beyond, for our purpose here they are less optimal. Flow charts have a large set of constituent symbols, which makes creating a sound diagram difficult for novices. In addition, there is no one-to-one mapping with program code that models the described behaviour thus requiring an unwanted extra step in an already overloaded education experience of our design students.
Now we turn to a teaching strategy for introducing the STDs in the team. Sorva [12] reviews the challenges that novice programmers have an understanding how to program, and advocates the development of ‘notional machines’, descriptions that express the workings of the program while abstracting from many of the distracting issues, such as coding syntax or language features that are not play. In several studies it is mentioned that unplugged activities (away from the computer) and role-playing can be successful strategies to engage students and to scaffold an understanding of computing [13, 14]. In the field of design role-playing is also acknowledged to help in team communication and to e.g. leverage the complexity of interactive experiences [6, 15].
In this paper we combine the concept of notional machines with the STD and develop an unplugged activity to role-play the system behaviour of a prototype with a previously designed user experience.

3 THE ACTING MACHINE TECHNIQUE

The name “Acting Machines” was chosen to emphasize (and motivate) students to act-out interactive behaviour as part of exploring a design brief and to try to first create (or fake) user experiences with a human in the loop (the so-called Wizard of Oz technique) [2]. In a sense the user and the interactive device are both actors in the interaction scenario as it unfolds over time. The task of the design team is to conceive how that interaction takes place.
After defining the user experience (Figure 1, rows 1 to 4), the team starts with defining the system behaviour of a device that can perform in the interaction (Figure 1, row 5 and 6). Here the Acting Machine technique is positioned, from the storyboard the team can develop an AMD and can subsequently engage with role-playing the AMD to check its consistency (and validate that the designed user experience holds). After arriving at a verified plan for the implementation, the team proceeds with transcribing it into code and constructing a prototype that can be made to work for real (Figure 1 row 7). As a last step of the process the actual prototype is tested with the team or others (Figure 1 row 8).

3.1 Creating the Acting Machine Diagram

An AMD consists of a collection of one or more states. A state is represented by a circle with a name inside it. There is the notion of a default state (depicted by a double-edged shape), this is the state that the system starts in. There also is a notion of the current state, this is initially the default state, but subsequent events may cause a change to the current state. The states are connected by transitions, a transition is represented by a unidirectional arrow. A transition is guarded by a cause, which is an event that can occur either by way of the user (pressing a button) or some internal event (a timer expiring). Because of a cause ringing true, an effect is realised. The effect can be one or multiple reactions effected by the device, such as changing the colour of a LED or some invisible reaction like starting a timer.

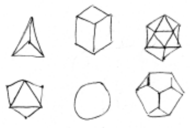
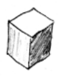
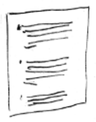
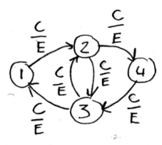
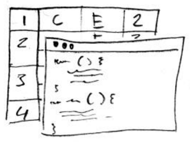
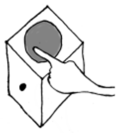
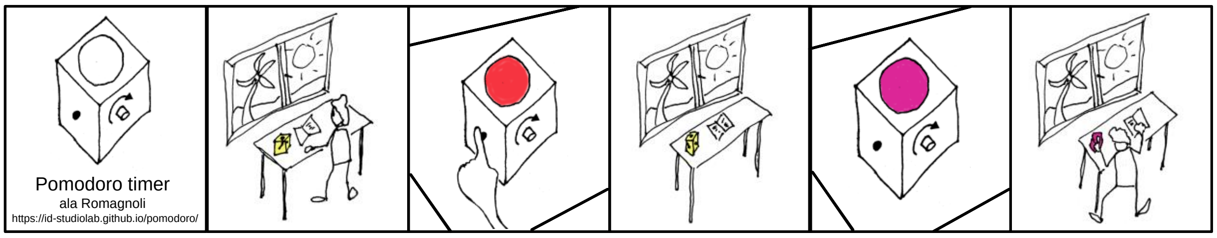
design user experience		ideas	Generate ideas of device mediated interactive experiences
		concept	Select a viable idea making the design assumptions explicit
		interaction scenario	Write a timeline based story of people interacting with the device
		storyboard	Draw the interaction scenario showing context and people interacting with the device: 1) start with what happens before, 2) draw a couple of possible interactions, 3) show what happens after
design system behaviour		acting machine	Create an AMD: 1) identify states from device behaviour changes in the storyboard, 2) link the states with transitions to allow for described changes, 3) describe behaviour changes exactly, these are the effects (E) 4) identify the causes of changes in behaviour (C)
		enactment	Evaluate the designed behaviour by acting out the diagram with a team (see section 3.2 for full description)
design the code		implementation	The designed behaviour has been validated, many potential errors are filtered out and the resulting AMD can readily be transcribed into an executable program.All the electronic component are collected and fixed in an embodiment.
		evaluation	With the prototype ready to try out, now it can be experienced and evaluated. At this point the AMD serves the purpose of providing a roadmap connecting the design and the implementation. A decision can made to improve the current design or to iterate further with one of the earlier discarded ideas.

Figure 1. Overview of the steps in a design process that employs interactive prototypes

Constructing an AMD from the storyboard in Figure 2 takes 4 steps, the result is visible in Figure 3. Going through the steps is an activity that ideally involves all team members. It may take a couple of iterations going back and forth from the storyboard to the AMD. It might even require iteratively improving the storyboard. Results shown here are the result of these iterations.

Step 1: involves identifying where in the storyboard the device exhibits different behaviours. Reading the storyboard in Figure 2, a number of different device behaviours can be identified, e.g. in the beginning, the device is off but as soon as the button is pressed, the device shows a green colour and the clock hand starts moving to signify the passing of time. These types of behaviour changes are the cues to finding states. We identify six states in total to capture all the behaviour changes: idle, work, break alarm, short break, work alarm and long break (see Figure 3).

Step 2: involves connecting all states with transition arrows according to the interaction sequence in the storyboard. For example, in the first frame of the storyboard, the device is off until the user presses the button starting a work period, this connects the idle state to the work state.



Lorenzo is an actor, he has to memorise a text for a play he is performing in. He is using the Pomodoro technique at home to structure his study time in a productive and healthy manner.

He starts by pressing a button on the side of the device. A green light appears on the top of the timer and the passage of time is made visible by a small clock-hand.

After 20 minutes an audible alarm sounds with an accompanying light animation, indicating it is time to have a short break. Lorenzo stops the alarm by pressing the button on the device.

Lorenzo walked to the kitchen and is making a cup of coffee.

After 5 minutes another alarm is played signalling it is now time to start another work period.

Lorenzo returns and presses the button. The alarm then stops and another 20 minutes work period is started. The timer repeats this process 4 times and then the it switches itself off. In Pomodoro terms it is now time for a longer break.

Figure 2. Example of a storyboard with a concept for a Pomodoro timer

Step 3: requires an investigation of each transition and listing all the changes in behaviour. These will be the effects that are performed each time the transition is activated. For example, as soon as the user presses the button a green LED turns on and the clock hand on the device starts showing the time ticking away. These effects will appear below the arrow of the transition in the AMD.
Step 4: requires determining for each transition the cause that activates it. For example, in the first frame of the storyboard the button being pressed is apparently a cause, once that happens, we advance through the AMD from state idle to state work. The causes appear on top of the transition arrow in the AMD.

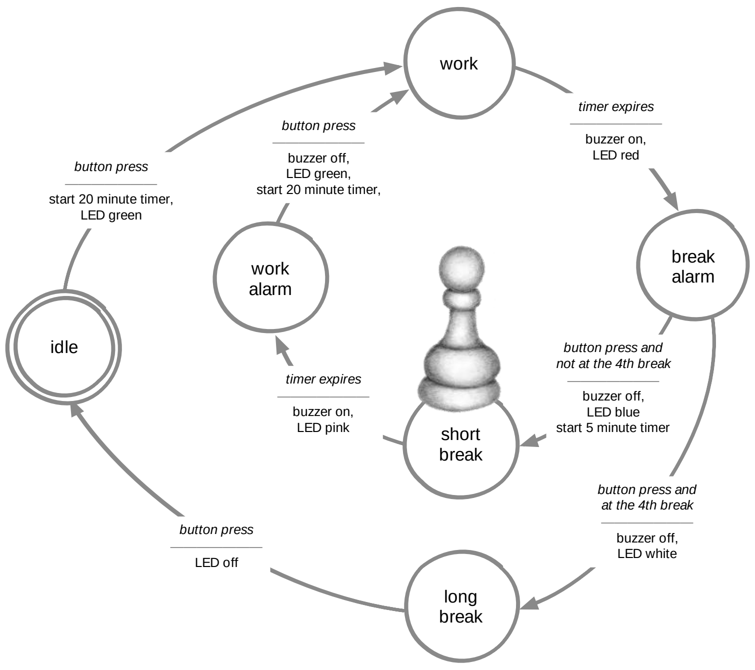

work
timer expires
buzzer on, LED red
break alarm
button press and not at the 4th break
buzzer off, LED blue start 5 minute timer
short break
timer expires
buzzer on, LED pink
work alarm
button press
buzzer off, LED green, start 20 minute timer,
button press and at the 4th break
buzzer off, LED white
long break
button press
LED off
idle
button press
start 20 minute timer, LED green

Figure 3. A complete AMD with the states and transitions. Each transition has causes (as the numerator) and effects (as the denominator). The current state is marked here by (the image of) a pawn on state short break

3.2 Evaluating the designed interactions through role-play

At this point in the process we encourage evaluation by role-playing the designed interactions (see Figure 4). The team sits around a table. On the table, the Acting Machine Diagram lies as a gameboard; a pawn is placed on one of the state circles to indicate the current state. The team members play four roles: *user*, *sensor*, *machine*, and *observer*. The *user* has a script with several tasks to perform with a mock-up of the interface of the designed device. The *sensor* watches the user actions and flags the occurrence of cause, such as ‘the user presses button A’, or ‘the light sensor picks up a shadow’. The student playing the *machine* enacts the designed behaviours as found in the effects on the AMD. She listens to the *sensor*, moves the pawn to reflect the state transition triggered by the indicated cause, and announces the effect (e.g. LED turns green). Finally, the *observer* focuses on seeing if the interaction unfolds according to the designed intention and keeps track of pain-points in the concept or its implementation. When such an ‘error’ is seen, the whole team can discuss how the diagram can be changed to obtain the desired sequences of cause and effect. As soon as the team is satisfied that the AMD is working well, they start with the implementation of the AMD to deliver a working prototype.

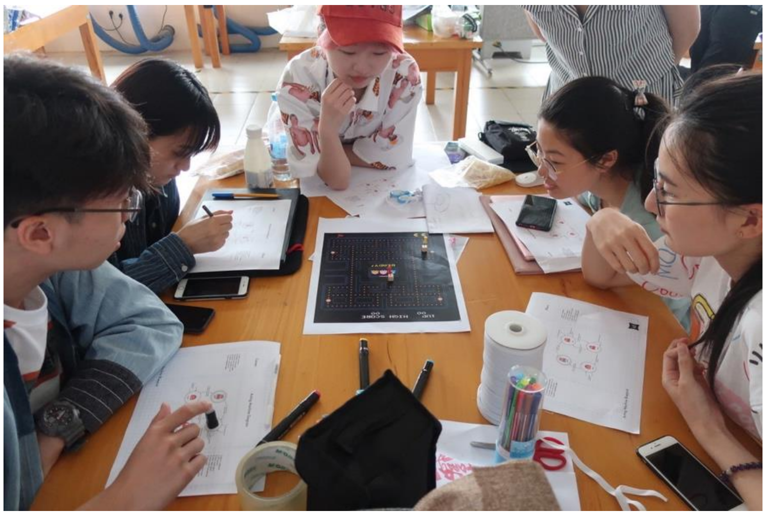


Figure 4. Role-playing the designed interactions. The roles are distributed from left to right: machine, observer, user, another observer and sensor

3.3 Implementing the interactive behaviour

At this stage mistakes or omissions in the interactive behaviour have been identified and fixed in the AMD. The team is now in a good position to start coding the interactive behaviour. This involves working with a computer, the technology for the prototype (development environment, microcontroller, sensors and actuators). We suggest this task is handed to a sub-team to streamline activities; it does not make sense to put more than 2 people at a computer. The AMD helps in constructing the code in several ways. The states and transitions, causes and effects in the diagram are labelled, and these labels can be used in the program code. This makes it possible for the whole team to eventually reach a shared understanding of how the behaviour is programmed.

4 DISCUSSION

We have over 5 years of experience with the AMD as a means to introduce novices to programming with Arduino[4]. In this paper we focus on the contribution of our Acting Machine technique to engage the whole team in designing the system behaviour and reflect on whether we are scaffolding an understanding of computing in the context of HCI. For evaluation we organised several workshops and embedded an introduction of the technique into existing full day workshops of our master’s programme. The number of participants varied between 30-60 students.

In all instances the Acting Machine Technique was presented in two stages. First a completely worked out example of the Pac Man game was shown and after a plenary demonstration of role-playing the AMD, the teams were asked to do the same (30 minutes). In the second stage we distributed the storyboard of the Pomodoro timer and asked the teams to derive an AMD that would model the system behaviour of the Pomodoro timer (60 minutes). All the time staff was walking around observing and helping when teams got stuck.

Role-playing the Pac Man game was entirely do-able for the teams. Enforcing turns (starting with the machine role) was sometimes not intuitive, however once enforced they realised it naturally models the main loop of a system. The Pac Man solution contained a couple of deliberate flaws with the intend to steer them to start considering alternatives. The teams all naturally moved into reconsidering features of the AMD and started redesigning the blueprint of the interaction as proposed by the AMD.

The Pomodoro timer assignment of deriving a new AMD was more challenging for the teams. Breaking the construction of an AMD down into 4 steps seemed logical during the design assignment. However, what we observed in class was lively discussion and sketching, teams did not stick to the strict order. Students started sketching, highlighting aspects in the storyboard and discussing about what could work.

Most teams were able to deliver a (partial) working AMD in time. Since there is no one optimal AMD for the system behaviour, we saw many different and surprising solutions.
During the workshops, our observation confirmed that we were scaffolding an understanding of HCI technology in the students. Whether this also has an impact in their education later programme remains to be determined.

5 CONCLUSIONS

With our Acting Machine technique. we are helping student to leverage the complexity of all tasks involved with prototyping interaction. The main contributions are: we offer a technique that connects to the user experience domain that our students already know; we offer scaffolding for the understanding of HCI computing and we offer a way to design the interactive system collaboratively with the entire team involved. Our experiences expose the need for better longitudinal evaluation methods to show the effectiveness of teaching prototyping techniques in the long run. The style of AMDs used in this paper is effective for modelling simple HCI systems. With HCI concepts moving into more complex technology (e.g. Internet of Things and Artificial Intelligence), we need stronger visual languages that can scale to the increasing complexity of concept designs.

REFERENCES

[1] Voûte, E., et al., *Innovating a Large Design Education Programme at a University of Technology.* She Ji: The Journal of Design, Economics, and Innovation, 2020. **6**(1): p. 50-66.

[2] Buxton, B., *Sketching user experiences: getting the design right and the right design*. 2010: Morgan kaufmann.

[3] Aprile, W.A. and A. van der Helm. *Interactive technology design at the Delft University of Technology-a course about how to design interactive products*. in *DS 69: Proceedings of E&PDE 2011, the 13th International Conference on Engineering and Product Design Education, London, UK, 08.-09.09. 2011*. 2011.

[4] Van der Helm, A., T. Jaskiewicz, and N. Romero Herrera. *Problem-based teaching of interaction design to fresh minds: Lessons from integrating experiential interactive prototyping in a course on interaction design and user experience research*. in *ICERI2015: Proceedings of the 8th International Conference of Education, Research and Innovation, Seville, Spain, November 18-20, 2015; authors version*. 2015. IATED.

[5] Jaskiewicz, T. and A. van der Helm. *Unlocking the Interactive Office: Concurrent Prototyping Approach*. in *Proceedings of the 2018 Designing Interactive Systems Conference*. 2018.

[6] Van Boeijen, A., et al., *Delft design guide: Design strategies and methods*. 2014.

[7] Harel, D., *Statecharts: A visual formalism for complex systems.* Science of computer programming, 1987. **8**(3): p. 231-274.

[8] Wagner, F., et al., *Modeling software with finite state machines: a practical approach*. 2006: Auerbach Publications.

[9] Harel, D. and A. Pnueli, *On the development of reactive systems*, in *Logics and models of concurrent systems*. 1985, Springer. p. 477-498.

[10] Thimbleby, H., *press on: Principles onf Interaction Programming*. 2007, MIT Press, Cambridge.

[11] Feijs, L., *Multi-tasking and Arduino: why and how?* Design and semantics of form and movement, 2013. **119**.

[12] Sorva, J., V. Karavirta, and L. Malmi, *A review of generic program visualization systems for introductory programming education.* ACM Transactions on Computing Education (TOCE), 2013. **13**(4): p. 1-64.

[13] Andrianoff, S.K. and D.B. Levine. *Role playing in an object-oriented world*. in *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*. 2002.

[14] Sentance, S. and A. Csizmadia, *Computing in the curriculum: Challenges and strategies from a teacher’s perspective.* Education and Information Technologies, 2016. **22**(2): p. 469-495.

[15] Boess, S., D. Saakes, and C. Hummels. *When is role playing really experiential? Case studies*. in *Proceedings of the 1st international conference on Tangible and embedded interaction*. 2007.