Code as Design Material

Rikard Lindell
Mälardalen University
Box 883 S72123 Västerås Sweden
rikard.lindell@mdh.se

Abstract
The new ubiquitous assistive devices have increased design space for innovative highly interactive design. Designers can no longer rely on a design process based on the known interaction idioms. This impedes the design process because the non-interactive material - sketches, scenarios, storyboards - does not provide designers the essential talk-backs needed to be able to make reliable assessments of the design characteristics. Whereas, interactive prototypes provide these talk-backs. What if we think of code as a design material, programming as a design craft, and what if the designer's repertoire include material consciousness with code?

Introduction

The new landscape of ubiquitous device with multitouch screens, accelerometers, gyros, compass, microphone, and camera make it more difficult for interaction designers to rely on a design repertoire based on the known interaction idioms. It requires quality-driven interaction designers and programmers with the ability to simultaneously establish and solve problems to create innovative, useful, and thought-provoking digital artefacts with rich experience qualities.

Interaction design describes itself as a design practice that form appearance and function of digital artefacts (Fällman 2008). Interaction design contributions are often based on *research through design* (Zimmerman et al. 2007). The appearance and functionality are portrayed by sketches, storyboards, videomatics, and interactive prototypes to communicate requirements to the software and product developers (Löwgren Stolterman 2004, Buxton 2007, Lindell 2009). The result of such design work is rich in clues to the finished product's appearance, behaviour, and function. However, the material in the design process is radically different from the code need to be written to implement the design as a working artefact (Lindell 2009, Vallgårda Sokoler 2010).

Keywords: design, interaction design, experience design, experiential qualities, highly interactive prototypes, explorative programming, material, materiality, craft

There is a big problem in how a development project runs between the phases of interaction design and engineering (Memmel et al. 2007). These two activities have different epistemology; interaction design is a design practice (Fällman 2008), while software engineering is struggling to describe itself as engineering and science (Boehm 2006). People who are active in these fields have different ways of thinking about how they work (Buxton 2009). Designers are trained to see a plethora of future designs for a situation. Whereas, engineers are trained to solve well-defined specific problem (Buxton 2007).

Sketches, moodboards, storyboards, and paper prototypes (figure 1) work in design situations where the designer experiments with known interaction idioms. Users, design colleagues, and programmers fill the gaps and imagine the user experience for the finished artefact based on their experience with these idioms. However, this approach does not work for innovative forms of interaction and user experience. To get talk-back from the interaction design it is necessary to create interactive prototype programs. Memmel et al. (2007) shows that the gap between designing digital artefacts and implementing them is not easy to bridge. The designer depicts the function and appearance in a different material than what the programmer uses to construct a program. A material has inherent characteristics that affect and provides the preconditions for what can be created with it - compared to wood, fabric, iron. Code provides other types of talk-backs than scenarios, sketches, storyboards and paper prototypes provide. The design process does not stop when the programming start, on the contrary, programming is a vital part of the design process.

Conversation with the material

Schön (1983: 78) describes how design is a "conversation with the materials of a situation." He portrays how experienced designers have a habitual ability to handle situations that are known to them. Designers create controlled situations by constructing virtual worlds for thought experiments and reflection-in-action in which time can be slowed down so that there is more space for reflection. Habitual skills are necessary for reflection-in-action.

“But the virtual world of the drawing can function reliably as a context for experiment only insofar as the results of experiment can be transferred to the built world. The validity of the transfer depends on with which quality the drawn world represents the built one. ... He learns, for example, how drawings fail to capture qualities of materials, surfaces, and technologies.” (Schön 1983: 159).

Schön describes the architect's material consciousness with both plans, drawings, and the finished building and his/hers ability to move between these materials. This ability can be transferred to the interaction design. In many design situations, designers can experiment with known interaction idioms. Users, design colleagues, and programmers can fill in the gaps based their experience and imagine the interaction experience of the finished artefact. However, this approach does not work for innovative forms of interaction and user experience (Löwgren 2011).

Interactive prototypes are needed to provide talk-backs from design’s features. The moulding of code is thus, a part of the design process for innovative highly interactive digital artefacts. Writing code to explore the design is similar to the ability of craftsmen who simultaneously are setting and solving the problem (Sennett 2008:26). According to Sennett problem, setting and problem solving has a rhythm. This rhythm relates subconscious and conscious reflection-in-action.

"Every good craftsman conducts a dialogue between hand and head. Every good craftsman conducts a dialogue between concrete practices and thinking; this dialogue evolves into sustaining habits, and these habits establish a rhythm between problem solving and problem finding. The relationship between hand and head appears in domains seemingly as different as bricklayering, cooking, designing a playground, or playing the cello. ." (Sennett 2008:9)

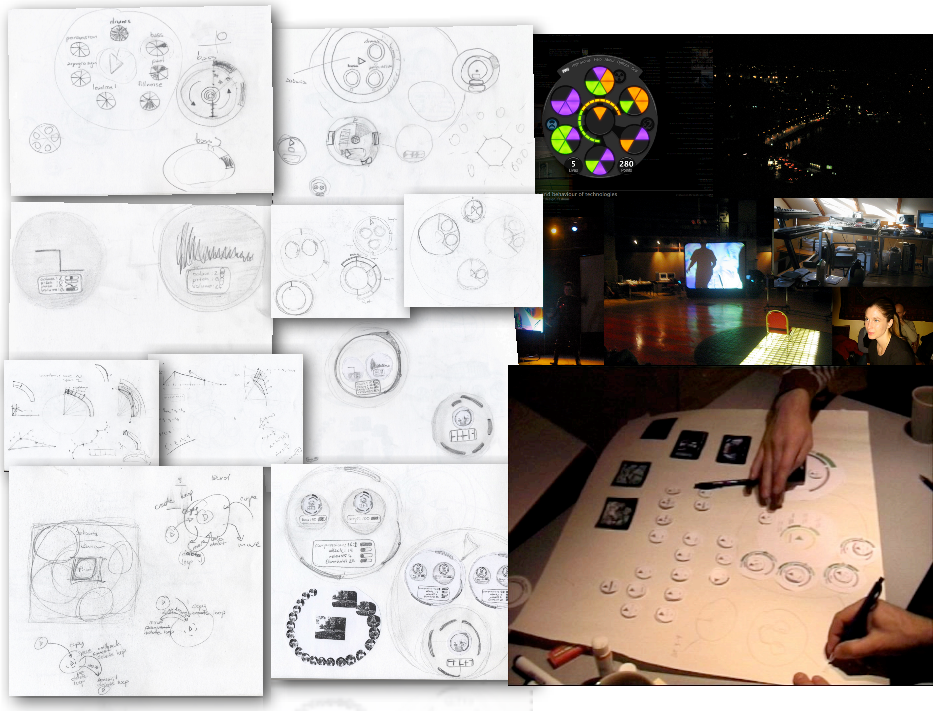


Figure 1. Interaction design materials; sketches, moodboards, and mockups.

Craftsmanship is thus characterised by the ability to see and solve problems through dialogue between hand and mind. Other characteristics are material consciousness and quality-driven approach on the edge of the manic (Sennett 2009: 234).

The profession, the practice, and ability to design and create interactive artefacts is a creative craft. McCullough (1998) discusses the craft related to interactive technology use and how an artisan approach can enrich interaction design. According to McCullough, there is a wide gap between the design of digital artefacts, and computer science and software engineering. Within software engineering craftsmanship is sometimes used derogatory to describe careless programming. Boehm (2006) for instance, uses the notion of craftsmanship as analogy for the 1960s, lack of professional discipline and careless "cowboy programming." However, negligence has nothing to do with craft making. On the contrary, describes Sennett (2008) the craftsman as a quality-driven bordering on manically busy perfecting his work. The craftsman must be patient and not tempt to do quick fixes. External factors - social and economic conditions, poor tools, or bad work environment, can be obstacles to the craftsman's good work. But, the craftsman's commitment is to do a good craftsmanship for its own sake. Sennett also describes the small-scale approach is still relevant.

Empirical study

In an empirical study on programming with 33 participants, including some interaction designers. An open and informal question was sent: "I wish that you write a sentence or two describing how you think and feel about your favourite programming language?"

Out of the collected data we created descriptive categories and concepts using grounded theory (Glaser Strauss 1967). A grounded theory grows in three or four phases, according to Hartman (2001: 40) or Guvå Hylander (2003: 70) respectively. The machinery of grounded theory in each of these phases; theoretic selection, theoretic coding, comparison, and conceptualising (Guvå Hylander 2003: 34). Here, the theoretical selection is the community of users of programming languages. During coding, sentences or words are marked or labeled as indicators that contribute to the growing theory. Preconceptualised ideas to theories are written as memos. Then, the indicators are compared, sorted and commented to be weaved into a theory during conceptualising. The machinery is used analogous in the following phases, but for each phase the theory gets more general and saturated.

Material

Material was the core category in the collected data. The material sets the conditions for the context and use of programming languages. It is also material that provides talk-backs for the creation of models, sketching, or exploration of a design. Utterances on flexibility and simplicity occurred repeatedly in the data. The material; the language and data have an inner pliability that enables designs to be moulded and reshaped, typically dynamic scripting languages (figure 2).

Explorative Programming

Another category that was identified was *exploration*. With explorative programming designers strive to portray aesthetic expression, function or interaction. They try their way into conversation with the material to find a design, in a continuous, problem setting and problem solving rhythm. Move-making-experiments explore mini-hypothesis via reflection-in-action. Here, programming languages are tools to incrementally explore and understand a problem. "What I cherish the most is that [Processing] is incrementally so that I can test my way. Sometimes it feels like sketching in the truest/best sense, when I can try my way to a new idea into an interactive behaviour. Sometimes."

Here is another quote that shows the exploration and problem-setting approach: "Both in the case of Flash and Processing you get to see directly and graphically the results of your coding, a kind of feedback that really enhances your comprehension of programming concepts, such as: "Oh, that's what happens if I loop it!", and "Hold it right there till someone presses a button!"." This quote indicates that interaction designers explorative programming is about exploring a design, to do both problem setting and problem solving. They obtain a material consciousness of digital artefacts. Explorative programming can be seen as part of a design repertoire and as a craft.

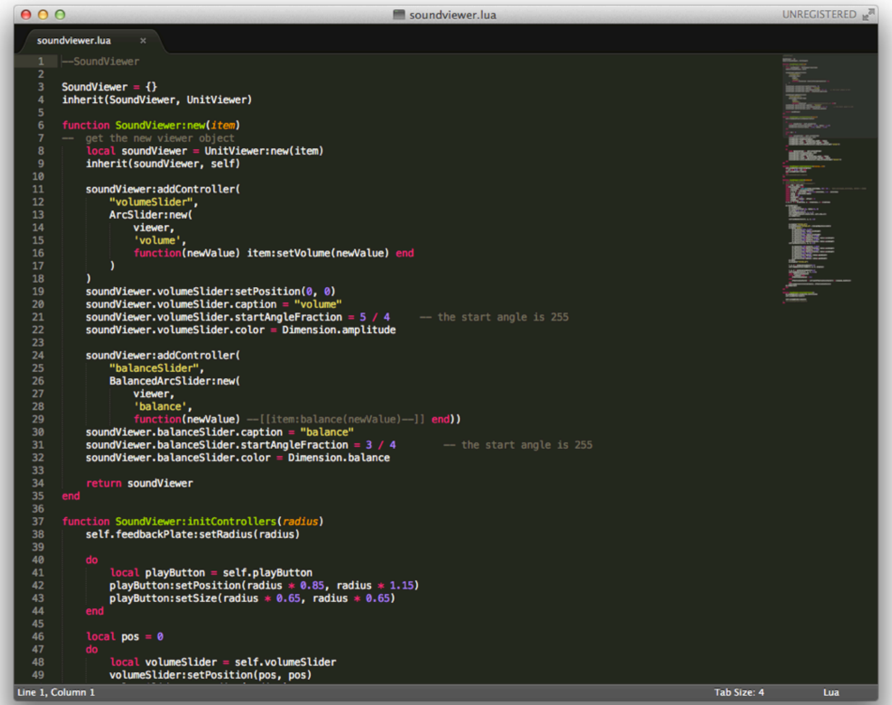
Rational

In the rational category the respondents described language specific theoretical and technical features, such as polymorphism, abstraction levels, and performance. The language paradigm was important in this category. But, there were also those who liked multi-paradigm languages with focus on the languages' technical features. Here's an illustrative quote: "... the language supports multiple levels of abstraction. Depending on the application, you can choose to either write code at a high level of abstraction, with object orientation, encapsulation, inheritance, dynamic binding and so on., Or on a more "hardware related " level, with standard C functions, simple data types and structer so on. "

The utterances in the rational category are not about what a language can be used for, and there are no emotional reasons as to why they prefer a specific language. The rational approach is also closest to a scientific approach to language that is derived from academic studies on this topic and bears witness to a technically rational approach. The respondents in this category do credit to their university education in computer science.

Discussion

The future challenges in the interaction design field are how we can meet the need for innovative highly interactive design for the drastically increased design space so that designers and programmers can reason about artefacts' materiality and the material they are built of. There are already programmers who have more of an exploitative approach to programming, where the code is a design material. Whereas, there are programmers who have a rational and scientific approach to software engineering. How can we develop tools and programming languages that provide richer feedback and facilitating the transition between designing and crafting interactions? How can the designer's repertoire be expanded to include material consciousness with the code for explorative programming? I believe the answer is in how the think of programming. The development of software – programming – is an activity that is closer to the craft than to science or engineering. Sennett (2008:24-26) depicts the Linux programmer as the modern craftsman. Valverde et al. (2006) has shown how open source communities are self-organising. An open source community is similar to a guild, where the masters are in control but also share knowledge and teach those who are less skilled. With this view on programming the epistemology of interaction design and programming is similar. We will have craft, but with different material.



```lua
--SoundViewer

SoundViewer = {}
inherit(SoundViewer, UnitViewer)

function SoundViewer:new(item)
--  get the new viewer object
    local soundViewer = UnitViewer:new(item)
    inherit(soundViewer, self)

    soundViewer:addController(
        "volumeSlider",
        ArcSlider:new(
            viewer,
            'volume',
            function(newValue) item:setVolume(newValue) end
        )
    )
    soundViewer.volumeSlider:setPosition(0, 0)
    soundViewer.volumeSlider.caption = "volume"
    soundViewer.volumeSlider.startAngleFraction = 5 / 4     -- the start angle is 255
    soundViewer.volumeSlider.color = Dimension.amplitude

    soundViewer:addController(
        "balanceSlider",
        BalancedArcSlider:new(
            viewer,
            'balance',
            function(newValue) --[[item:balance(newValue)--]] end))
    soundViewer.balanceSlider.caption = "balance"
    soundViewer.balanceSlider.startAngleFraction = 3 / 4        -- the start angle is 255
    soundViewer.balanceSlider.color = Dimension.balance

    return soundViewer
end

function SoundViewer:initControllers(radius)
    self.feedbackPlate:setRadius(radius)

    do
        local playButton = self.playButton
        playButton:setPosition(radius * 0.85, radius * 1.15)
        playButton:setSize(radius * 0.65, radius * 0.65)
    end

    local pos = 0
    do
        local volumeSlider = self.volumeSlider
        volumeSlider:setPosition(pos, pos)
```

Figure 2. A screen dump of Lua code – a dynamic scripting language

Acknowledgements
I thank all the participants of the study. I thank professor Jonas Löwgren of Malmö University for our conversations on the design and the craft of making interactive artefacts.

References
Boehm, B. 2006. *A view of 20th and 21st century software engineering*. In Proceedings of the 28th international conference on Software engineering (ICSE '06). ACM, New York, NY, USA, 12-29.

Buxton, B. 2007, *Sketching User Experiences - getting the design right and the right design*, Morgan Kaufmann, ISBN 978-0-12-374037-3

Buxton, B. 2009. *On Engineering and Design: An Open Letter*. Businessweek. April 29, 2009. http://www.businessweek.com/innovate/content/apr2009/id20090429_083139.htm

Fällman, D. 2008. *The Interaction Design Research Triangle of Design Practice, Design Studies, and Design Exploration*, Design Issues, MIT press 2008 4-18.

Glaser, B., Strauss, A. 1967. *Discovery of Grounded Theory*. Strategies for Qualitative Research. Sociology Press

Guvå, G., Hylander, I. 2003 *Grundad teori ett teorigenererande forskningsperspektiv*. Liber, ISBN 9147050837

Hartman, J. 2001. *Grundad Teori*. Studentlitteratur AB, ISBN 9144006527

Lindell, R. 2009, *"Jag älskar att allt ligger överst" – En designstudie av ytinteraktion för kollaborativa multimedia-framträdanden*. MDH University Press Dissertation: 72, ISBN 978-91-86135-24-9.

Löwgren, J., Stolterman, E. 2004, *Desgin av informationsteknik - materialet utan egenskaper*, Studentlitteratur 2004, ISBN 91-44-04203-5

Löwgren. J, 2011. In personal correspondence on prototyping digital artefacts using Processing. Nov 2011.

McCullough, M, 1998.*Abstracting Craft - the practiced digital hand*, MIT Press, ISBN 0-262-13326-1

Memmel, T., Gundelsweiler, F., and Reiterer, H. 2007. *Agile human-centered software engineering*. In Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it - Volume 1 (BCS-HCI '07), Vol. 1. British Computer Society, Swinton, UK, UK, 167-175.

Schön, D. 1983, *The Reflective Practitioner - how professionals think in action*. Basic Books, ISBN 0-465-06878-2

Sennett, R. 2008. *The Craftsman*. Penguin Books, ISBN 978-0-141-02209-3

Vallgårda, A., Sokoler, T. 2010, *A Material Strategy: Exploring Material Properties of Computers*. International Journal of Design 2010, Vol 4, No 3

Valverde, S., Theraulaz, G., Gautrais, J., Fourcassie, V., Sole, R.V. 2006. *Self-organization patterns in wasp and open source communities*. Intelligent Systems, IEEE , vol.21, no. 2,

Zimmerman, J., Forlizzi, J., and Evenson, S. 2007. *Research through design as a method for interaction design research in HCI*. In *Proc. CHI 2007*, 493-502, ACM Press 2007