The Craft of Programming Interaction

Rikard Lindell
Mälardalen University
Box 883
72123 Västerås Sweden
rikard.lindell@mdh.se

ABSTRACT
The creation of useful artefacts with rich experiential qualities required quality driven interaction designers and programmers with the ability to simultaneous problem setting and problem solving. Interaction design is a design practice that defines the appearance and function of digital artefacts. Bridging interaction design and engineering is problematic because design and engineering have different epistemology. Designers are trained to see a plethora of future designs for a situation and explains the phenomena of a context. Engineering focus on problem solving and depends on agreement about ends. In this paper I suggest that the poor state of designers and programmers who are not standing together can be avoided if we give up the claim that software development should be engineering or science, and instead see it as a quality-driven craftsmanship.

Author Keywords
design; interaction design; experience design; highly interactive prototypes; programming; material; craft

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI):

General Terms
Human Factors; Design

INTRODUCTION
This paper presents a normative view, grounded in literature, and empirical findings, on how to bridge the practices of interaction design and software engineering.

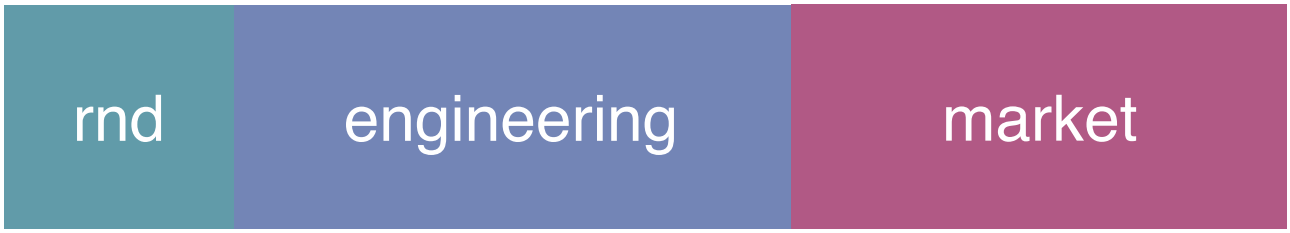
The creation of useful artefacts with rich experiential qualities required quality driven interaction designers and programmers with the ability to simultaneous problem setting and problem solving. People use interactive software, websites and mobile applications in different contexts for different purposes. Boehm shows a focus shift in software engineering to usability and that requirements of interactive artefacts cannot be defined a priori [1].

Interaction design has indulged itself in being a design practice that tries to define the appearance and function of digital artefacts [2]. Sketches, storyboards, videomatics, and interactive prototypes depict the appearance and functionality, and at best convey requirements to software engineers [3,4,5]. The result from a design process is rich in clues to the finished product. But, the material in the design process is different from the code that implements the design into a working artifact [6].

There is a big problem in how a development project runs between the phases of interaction design and engineering [7]. These two activities have different epistemology; interaction design is a design practice [2], while software engineering is struggling to describe itself as engineering and science [1]. Designers are trained to see a plethora of future designs for a situation. Design explains the phenomena of the context. It's about framing the problem space of the context, cut into a search tree of plentiful design proposition to reach the right user experience design of a future artefact [4,8]. Design is the exploratory use of malleable tactile materials and provides suggestions for possible future solutions [2,4,8]. The goal of the design process is that as much as possible frame the problem for an engineering process to take over to solve.

Sketches, storyboards, and paper prototypes works 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. To get talk-back from the interaction design it is necessary to create interactive prototype programs. The design process does not stop when the programming start, on the contrary, programming is a vital part of the design process.

BACKGROUND
Schön discuss how faith in rational, scientific, and technological solutions are spread because of how they were successfully applied during World War II, where the solution to a problem was to supply more resources [9]. The point he makes is that engineering is close to science. “They began to see laws of nature not as facts inherent in nature but as constructs created to explain observed phenomena, and science became for them a hypothetico-deductive system. In order to account for his observations, the scientist constructed hypotheses, abstract models of an unseen world which could be tested only indirectly through deductions susceptible to confirmation or disconfirmation by experiments. The hart of scientific inquiry consisted in the use of crucial experiments to choose among competing theories of explanation.” This quotation describes how belief in deductive reasoning disconnecting the explanation of the world from the material to be explained. A scientific approach allows the engineer to deduce, analyse and define problems in a rational way; the positivist epistemology of science [9].



Figur 1. Buxton's image of the organisation for the engineering driven product development [4].

Boehm describes in his expose of just over a half-century of software engineering how the field evolved, mainly that we increasingly focusing on usability and value [1]. Software engineering has realised the problems with the top-down waterfall development model and introduced iterative models [10]. These models deal with changes in the problem space by development iterations. Each iteration in the spiral model or in the rational unified process (RUP) is basically a waterfall model. The foundation is still the technical rationality epistemology.

Buxton describes how an engineering-driven organisation is organised in a simple diagram [4], see Figure 1. First you do research and development, then you do engineering, and finally hand the product to the sales organisation. This type of organisation requires an agreement on ends. At changes and difficulty in clearly defining the problem dissonance arises in the organisation: “Technical Rationality depends on agreement about ends.” In the citation Schön delineates how technical rationality does not address situations where the result is uncertain and where there is no ready-defined problem to solve.

Reflection-in-action and interaction design

Technical rationality and focus on ends has a different epistemological dimension than Reflection-in-action - Schön's term for the reflective practitioner way of thinking and acting. The reflective practitioners have practical knowledge (knowledge-in-practice), they can be aware or unaware of this knowledge regardless of guild. Reflective practitioners deal with problem setting and unique and complex situations, mainly through reflection-in-action (reflection-in-action). Reflection-in-action can be summarized in three phases that are repeated: (1) Frame the problem, assess the situation, and understand the working material. (2) Perform moves over the situation. These moves are parts of the practitioner’s repertoire. They are small experiments with the intentional result, but often with unintended effects (both positive and negative). (3) Reflect and evaluate the consequences of action in conversation with the situation. Practitioners take in and reflect on how the situation responds (talk-backs). The conversation happens in what Schön calls the medium's language. After this phase the process starts over again.

Design problems are often vague, complex, and contradictory [11]. In the problem setting phase interaction designers name the phenomena that they will pay attention to and work with. They create design concepts through various design techniques: sketches, mood boards, storyboards, or paper prototypes to better understand and frame the problem. Concept design are evaluated and refined through introspection, criticism, and user studies, such as Wizard of Oz method [4]. The design work will also increase the understanding of the situation and context. Sketching interfaces and designing paper prototypes will also learn interaction designers more about the context [12]. Figure 2 shows a design driven organisation and how the design team follows the design through the entire process. Such an organisation also understand that fellows close to the market can provide feedback from users.

Interaction Designers have a repertoire of interaction styles that they can apply for different problems [3]. To be able to design great interfaces interaction designers should master programming. It is part of being conscious of the design material [5, 13]. While interaction designers can implement a design by composing software, they must not be seduced by technologies for technology's sake.

Interaction Designers create architecture for interactive artefacts and their spatial and temporal properties. They design the artefact topology, the artefact appearance on the screen or in the space and how artefact change over time because of interaction. Interaction Designers understand the consequences of different designs and have a feel for how a design can be realised. Similarly, interaction designers build interactive prototypes for technical substantiate and in full understand what they designed. Its about material consciousness [14]. The difference between the architect and interaction designer is that the latter build their model in full scale, albeit quickly, and at times chaotic, but it is a model and not a product.

Craft

The profession, the knowledge, and ability to design interactive artifacts is a creative craft. McCullough discusses the craft related to interactive technology use and how an artisan approach can enrich interaction design [15]. According to McCullough, there is a wide gap between the design of digital artifacts, and computer science and software engineering.

That craftsmanship has not been highly regarded is not new. Within software engineering is sometimes used craftsmanship derogatory to describe careless programming. Boehm for instance, uses the notion of craftsmanship as analogy for the 1960s, lack of professional discipline and careless "cowboy programming" [1]. However, negligence has nothing to do with craft. On the contrary, describes Sennett the craftsman as a quality-driven bordering on manically busy perfecting his/her work [14].

Figure 2. Modified figure of Buxtons model of a design-driven organisation [4].

The craftsman must be patient and not tempt to do quick fixes. But, the craftsman's commitment is to do a good craftsmanship for its own sake.

"Craftsmanship may suggest a way of life that waned with the advent of industrial society–but this is misleading. Craftsmanship names an enduring, basic human impulse, the desire to do a job well for its own sake. Craftsmanship cuts a far wider swath than skilled manual labor; it serves the computer programmer, the doctor, and the artist; parenting improves whet it is practiced as a skilled craft, as does citizenship. In all these domain, craftsmanship focuses on objective standards, on the thing in itself. [...] And though craftsmanship can reward an individual with a sense of pride in work."

Sennett describes the craftsman's ability to simultaneously identify problems and solve them. This is consistent with Schön's ideas about refelektion-in-action, discussing problems qualifying in difficult situations. Sennett says that problem setting and problem solving has a rhythm that 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." [14].

Material

Information technology, according to Löwgren and Stolterman, is a material which it has no recognisable features [3]. This view combines interaction with "traditional" design trades and crafts.

The similarity between the industrial designer and architect on the one hand and the interaction designer however, lies in creating technology. But, the industrial designer and architect's material is concrete as opposed to interaction designer material that is intangible. We distinguish between these disciplines from one another by the material they are working in and what they create, but they have similar practices, methods, and approaches to design. IT is on the surface visual, auditory, or haptic, but this is an illusion created by calculations and represented in ones and zeros and described with programming languages. Media and language for interaction designers are sketches of the interface's appearance, creating paper prototypes and to write computer programs that embody digital artefacts' behaviour in working prototypes.

If we do not consider the development of software such as software engineering, which qualities are in the development process for the continuing development work performed as a work of reflective practitioners and quality driven craftsmen?

PROGRAMMING IS A CRAFT

"How can we make sure we wind up behind the right door [good code or bad code] when the going gets touch? The answer is: craftsmanship." [16].

In Martin et. al quote there is a notion of pursuing the mastery of craftsmanship. Gaining experience through a dialogue between tacit knowledge and explicit critique, and relying on their mastery in their practice. [14].

Empirical Findings

The Manifesto for Agile Software Development and later the Manifesto for Software Craftsmanship provide both empirical evidence supporting the idea of programing as a craft. Manifesto for Agile Software Development was written as a critique of a rigid approach to requirements specification, analysis, design and documentation and to put focus on creating useful artifacts with rich user experience. The manifesto reads: Individuals and interactions over processes and tools, Working software over comprehensive documentation, Customer collaboration over contract negotiation, and Responding to change over following a plan [17]. The manifesto reflects the programmer's frustration that spend most of the time to document and manage the project instead of writing code.

As I write this, the Manifesto for Software Craftsmanship[1] – Raising the bar has been signed by over 9,000 people (9,410) with the constant rising number of signatures. The manifesto reads:

"As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value: Not only working software, but also well-crafted software. Not only responding to change, but also steadily adding value. Not only individuals and interactions, but also a community of professionals. Not only customer collaboration, but also productive partnerships"

[1] http://manifesto.softwarecraftsmanship.org/

Figure 3. Design and Programming as a craft facilitates the transition in the design work's change of materials and technology – from paper to pixels from sketches to code.

The development of software - programming - is an activity with a wide range of intrinsic properties that are closer to craft than science or engineering. Sennett describes the Linux programmer as the modern craftsman [14].

"People who participate in "open source" computer software, particularly in the Linux operating system, are craftsmen who embody some of the elements first celebrated in the hymn to Hephaestus, but not others. [...] The Linux system is public craft. The underlying software kernel in Linux code is available to anyone, it can be employed and adapted by anyone; people donate time to improve it. Linux contrast to the code used in Microsoft, its secrets until recently hoarded as the intellectual property of one company. During these two decades, the software industry has morphed within its brief life into a few dominant firms, buying up or squeezing out smaller competitor. In the process, the monopolies seemed to churn out even more mediocre work." [14].

Martin et al. press the importance of quality-driven and disciplined practice in the programming craft. The programmer must carefully name functions, classes, interfaces, methods [16]. Martin et al. focus on the code, to carefully write clean code based meticulous attention on the principles and guidelines for the scope of a function or method, of responsibility for a class, how test-driven development is pursued, how concurrency is best implemented, etc. Above all, Martin et al. show that the problem cannot be solved at once but a problem can be explored by writing tests and constant iteration of possible improved solutions.

Agile Development with XP and Scrum in particular is a big step for software engineering in the direction of focusing on service qualities and user experience as opposed to non-agile development models, such as RUP, the spiral model, and waterfall model. But, despite the Agile Manifesto have XP and Scrum and other iterative development models still a clear plan-implement-evaluate cycle oriented that extends over a longer period, as at least weeks, but in practice longer. A common feature for these methods is agreement about ends.

In recent years, the Kanban method attracted attention by providing even greater freedom for adaptation [19]. "Scrum is less prescriptive than XP, since it doesn't prescribe any specific engineering practices. Scrum is more prescriptive than Kanban though, since it prescribes things such as iterations and cross-functional teams. ... Kanban leaves almost everything open. The only constraints are Visualize Your Workflow and Limit Your WIP. Just inches from Do Whatever, but still surprisingly powerful." [19]. This quotation shows how Kanban can be a support for an agile development process in constant change. Kanban allows the goal of a work in progress (WIP) change during the process. This means that a WIP can have an open end. Thus, Kanban a radically different approach than all the earlier development models; Scrum, XP, RUP, Spiral model and Waterfall model included.

The open-endedness of Kanban stands out and allows practitioners reflection-in-action. Dealing with messy situations and continuous problem setting and problem solving important becomes pillars of programmers' work. This makes the interaction designer and the programmer standing on common ground; craftsmanship epistemology. The ongoing design process turns into a software craftsmanship process, see Figure 3.

The main difference between interaction designers and programmers is that the material of interaction design has slightly different characteristics than the material for programming. The transition between design and programming is in this situation regards the knowledge exchange between practitioners of different repertoires.

DISCUSSION

According Buxton et al the problem setting should be done without writing code. However, programming is a good tool for a design that is difficult to portray on paper; for example, collaboration, pliable, or highly interactive features. Innovative interaction techniques require interactive prototypes. But, exploratory programming allows various designs to be explored and in retrospect transform the code into clean code [15]. One way to explore a design is to propose solutions by writing tests. By first writing tests explores and sets the problem while the programmer simultaneously solves the problem [15].

Buxton notes that there is still a division between design and engineering and suggests how it can be bridged [13]. But, is still an open question for research in design and software engineering [7, 13].

We need to use methods of agile software development with a different approach. The development model Kanban contains characteristics that allow an artisanal approach. The Kanban development model does not prescribe specific roles, and is designed to accommodate continual change.

Programmers and designers can simultaneously be doing problem setting and problem solving.

One way to bridge the design process and implementation here is to let the designer be part of the development team. Instead of user stories as a description of the work in progress (WIP), the concrete material from the design process is used - mood boards, sketches, storyboards, videomatics etc. Initial WIP uses an explorative programming approach to continue the design process and explore the problem space. As the artefact takes shape, the development process can adopt a more pragmatic approach.

Kanban is a relatively new model in software engineering but has since become popular in game development. It is no coincidence, since game development is focused on highly interactive experience and game play. But, to use Kanban artisanal manner, the participants in the project need to have the craftsmanship epistemology.

A practice oriented epistemology and ontology bridge the designing and constructing activities within interaction design and programming. An artisanal approach facilitates the design and development of innovative and highly interactive digital artifacts that have novelty and relevance.

REFERENCES

1. Barry Boehm. *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. (2006)
2. Daniel Fällman. *The Interaction Design Research Triangle of Design Practice, Design Studies, and Design Exploration*, Design Issues, 24.3, p. 4-18, MIT press (2008).
3. Jonas Löwgren, Erik Stolterman. *Desgin av informationsteknik - materialet utan egenskaper*, Studentlitteratur, ISBN 91-44-04203-5, (2004)
4. Bill Buxton, *Sketching User Experiences - getting the design right and the right design*, Morgan Kaufmann, ISBN 978-0-12-374037-3, (2007)
5. Rikard Lindell. *"Jag älskar att allt ligger överst" – En designstudie av ytinteraktion för kollaborativa multimedia-framträdanden*. Mälardalen University Press Dissertations: 72, ISBN 978-91-86135-24-9. (2009)
6. Anna Vallgårda, Tomas Sokoler. *A Material Strategy: Exploring Material Properties of Computers*. International Journal of Design. Vol 4, No 3 Dec 21. (2010)
7. Thomas Memmel, Fredrik Gundelsweiler, and Harald Reiterer. *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. (2007)
8. Klaus Krippendorf. *The Semantic Turn*. CRC Press, Taylor & Francis Group. ISBN10: 9-415-32220-0, (2006)
9. Donald A. Schön, *The Reflective Practitioner - how professionals think in action*, Basic Books, ISBN 0-465-06878-2, (1983)
10. Boehm, B.W. 1985. "*A Spiral Model of Software Development and Enhancement," from Proceedings of an International Workshop on Software Process and Software Environments,* Coto de Caza, Trabuco Canyon, California, March 27-29, (1985)
11. Erik Stolterman. *The Nature of Design Practice and Implications for Interaction Design Research*. i International Journal of Design, 2(1), p 55-65. (2008)
12. Barbara Tversky, *What do sketches say about thinking?,* in Proceedings of AAAI Spring Symposium on Sketch understanding, sid 148-151, (2002)
13. Bill Buxton.-*On Engineering and Design*: An Open Letter. Businessweek. April 29, 2009. http://www.businessweek.com/innovate/content/apr2009/id20090429_083139.htm
14. Richard Sennett. *The Crafsman*. Penguin Books, ISBN-13: 978 0 141 02209 3, 2008
15. Malcom McCullough, *Abstracting Craft - the practiced digital hand*, MIT Press, ISBN 0-262-13326-1, 1998
16. Robert C. Martin, Michael C. Feathers, Timothy R. Ottinger. *Clean Code: A Handbook of Agile Software Craftsmanship*. ISBN 978-01-3235-088-4
17. Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas. *Agile Manifesto*. http://agilemanifesto.org/. (2001)
18. Mikael Lindvall, Vic Basili, Barry Boehm, Patricia Costa, Kathleen Dangle, Forrest Shull, Roseanne Tesoriero, Laurie Williams and Marvin Zelkowitz. *Empirical Findings in Agile Methods*. Ed: Don Wells, Laurie Williams. *Extreme Programming and Agile Methods — XP/Agile Universe* 2002 Lecture Notes in Computer Science, Volume 2418/2002, 81-92 (2002)
19. Henrik Kniberg, Mattias Skarin. Kanban and Scrum - making the most of both. C4Media inc, ISBN: 978-0-557-13832-6. (2010)