Software Authoring as Design Conversation

Andrée Woodcock and Richard Bartlett

Visual and Information Design Research Centre,
The Design Institute
Coventry School of Art and Design
Coventry University,
Coventry, UK
A.Woodcock@coventry.ac.uk

Abstract. The paper outlines a pilot study to investigate whether software authorship[1] could be regarded as a design activity, with special reference to 'reflective practice.' Verbal protocols collected from software authors undertaking a 'real task' indicated that there was merit in studying programming as a design activity – that different forms of reflection did take place, and that the collection of reflections and moments of surprise may lead to a greater understanding of both design and the nature of software authorship. This could be used to inform education and the development of software authoring support tools.

1 Introduction

The inspiration for this pilot investigation was on the one hand, my own experience as an ergonomist (or user champion) employed in the early stages of computer projects and on the other, as a design researcher interested in understanding the design process with a view to developing new tools to make such a process more effective. I believed at the start of the research that looking at programming as a design activity could lead to new insights into professional practice that could have ramifications for both software authoring and design research.

Whereas it is customary to start projects with a review of previous research in the field, in order to bring a fresh perspective to the study of programming we have deliberately distanced ourselves from previous research regarding nomenclature, novice vs. skilled programmers, and cognitive psychology. The precedents for this research were set by Schön's seminal works on reflective practice [8 and 9] and later in Winograd [7 and 14], and Ericcson and Simon's [4] use of protocol analysis. These were applied by Dorst [3] in the study of design activities.

[1] Software design, software authorship and to some extent programming are used interchangeably. The focus of interest is on the practitioner who writes and tests the code such as the developer of shareware.

Although programming is sometimes referred to as an ‘art’ or a ‘craft’, this usually leads to a discussion of programming skills and techniques rather than looking at problem solving, creativity and professional competencies. From my own art and design community, there is little interest in software design per se, unless it is part of creative production. Opportunities for cross over between the two disciplines might exist, especially in terms of recognition of software design or authorship as a design profession, but this is not occurring at present.

The focus of this research is on how software authors move around the design solution space as they undertake a realistic, complete design task (see section 2.2 for details), with the aim of addressing four basic questions:

1. Can software authorship be regarded as a design activity?
2. Can design research methods, employed in the study of design activities in other domains be used to gain new insights into the process?
3. Can programmers or software authors be regarded as reflective practitioners?
4. Would studying programming in this way be of any practical value?

1.1 Similarities between software design and other forms of design

All design progresses from an (un)stated need. It is essentially practical in nature. The need may be specified in a variety of ways, as formal, informal or evolving briefs or requirements that may have a loose or tight constraint on design activity. The artifact designed to meet those needs may take many forms. The designer, no matter in what domain, applies his skill, knowledge, creativity and experience to bring together a series of approximate solutions (or concepts). These may be manifest in the form of sketches, sketch models, or initial code. From these an artifact will be created that more or less fulfils the initial need.

The designer in most cases is not the client, and may not even be a representative of the group that originated the need. For this reason, designers sometimes progress without a full knowledge of the necessary facts thereby producing designs that are inappropriate.

Schön [7] cited Vitruvius, the Roman architecture critic as stating that well designed buildings should exhibit firmness, commodity and delight. The same principles can be applied to all cases of design, including software design:

Firmness: A program should not have any bugs that inhibit function.
Commodity: It should be suitable for the purposes for which it was intended.
Delight: The experience of using the program should be a pleasurable one.

As a further example of design, Industrial Design Engineering has been defined as the “development of durables (mass produced products for people, based on the integration of the interests of users, industry, society and the environment),” Buijs [1]. Such artifacts may be large or small, simple or complex. They should be seen as coherent by both the producers and the users, and should integrate the demands of a variety of stakeholders, up to the level of society. For industrial, as well as software design, a number of issues have to be taken into account such as business, technical, ergonomic and aesthetic issues. The final product is usually a compromise between all of these issues, and is the synthesis of input from diverse and often-conflicting disciplines and stakeholders that requires in its shaping and production skills beyond the scope of any one individual. We are only just beginning to understand how such teams should be managed, and the effects of different forms of knowledge on the final design. Group design or authorship is outside the remit of this research, however should the pilot meet it's initial aims this may be an avenue for later study. Instead, this research will focus on the conversation a software author has with his design problem.

Taking the similarities between industrial and software design further. Both are relatively young professions, with industrial product design only emerging as a field in its own right within the last 50 years. It is only within the last two decades that industrial designers have started to reflect on or about what they are doing. Much of this early reflection was based on rational problem solving [10] and methods adopted from mechanical engineering eg [12] with a view to managing process. Software design has borrowed methods from for example, psychology and again rational problem solving, Both industrial and software design have evolved their processes moving from the traditional waterfall, through cyclical development to rapid prototyping whereby design and complexity (as mentioned previously) have to be managed earlier.

More recently the trend in industrial design has been towards looking at the creative, social and psychological aspects of designing [2] and looking at design as reflective practice, i.e. focusing on professional practice rather than theory. This pilot study applies a similar approach to software design.

The many parallels to be drawn between the two disciplines, make software design an appealing area of investigation for design research. Both software and industrial design evolve very quickly – whereas the industrial or product designer may generate hundreds of rapid sketches, the software author will often generate hundreds of lines of code – when searching for the optimum solution to one or more problems. Each solution will have ramifications for the final solution, and impact on different sub areas. Software designers spend a lot of time validating and refining their designs (debugging). "Testing is not just concerned with getting the current design correct. It is part of the process of refining the design....Even the smallest bit of code is likely to be revised or completely rewritten during testing and debugging. We accept this sort of refinement during a creative process like design," Reeves [11]. Likewise a designer, may draw, rubout and redraw a line on a concept sketch of a car time and time again, until the curve is perfect. Both activities indicate that the designer is entering into a conversation with the design. Trapping the verbalizations of software designers engaged in such activities may lead us to a new understanding of their professional practice in a similar way to studies undertaken of product designers. Additionally, the difference in media (written code, as opposed to hand drawn sketches) may make it easier to understand the underlying processes by the removal of certain ambiguities.

In answer to the first two questions posed by the research I think that there are sufficient recognized parallels between software authoring and design in general for it to be amenable to study in a similar way. So, the task is now to determine whether there is evidence that programmers are reflective practitioners, and if they are, if there is any benefit in studying them in this way, and in attempting to view software design as a profession.

1.2 Reflective practice and design conversation

Verbal protocols provide rich sources of information for researchers, for example they can be analysed in terms of recurring patterns, parts of speech, design activity (such as incubation, design, verification and testing) and goal directed working. If software authors are reflective practitioners, engaged in a conversation with the design problem, then periods of reflection should be noticeable – but what will they look like?

For Schön, all professions are design like in so far as "they all consist in conceptualising, planning, patterning or otherwise establishing cognitive order", Walks [13] p43. During the course of design, problems may be experienced, either as a consequence of previous actions or because of a lack of understanding of the requirements of the solution. At such times "practitioners apply tacit knowledge-in-action, and when their methods do not work out, they do not take time out, to reflect or disengage, but reflect in action using the knowledge of their practice rather than the knowledge of science." Walks (op cit), p 44. Design studios are places where such experience and reflection can be acquired and shared. A reappraisal of software authorship as professional practice could eventually lead to computer studios as opposed to computer laboratories.

Protocols of design sessions can be studied to show movement through the design process in terms of the setting and meeting of goals and subgoals. Although I am interested in tracing this movement, which can be seen in the transcripts gathered during this work, for this paper I would like to focus primarily on identifying periods of reflection. What I would hope to do in subsequent work is to identify the relationship between reflections and the setting of goals and subgoals, to mediate movement through the solution space [5]

When a designer makes a move (ie towards generating a solution), that move will produce a series of results, not just the ones that were specified or hoped for. For programmers, a piece of code may or may not work in the way it was intended (ie meet its subgoal) but it may have an effect on other aspects of the program (for example by changing the instantiation of variables). Looking at what has happened in debugging and testing may lead the designer to realize that they had made both simple coding errors, but also might not have fully understood the requirements of the solution to begin with. So, during debugging the programmer may exhibit reflection –in-action and enter into a conversation with the design problem. This in turn will lead to future actions. "This unpredictability is a central attribute of design-it is not necessarily the defining one, but it is important. It means that there is no direct path between the designer's intention and the outcome" Schön [14]. By entering into a conversation with design, the complexity becomes clearer. A study of the software design process, especially debugging may reveal that "sometimes, the designer's judgments have the intimacy of a conversational relationship, where (s)he is getting some response back from the medium, (s)he is seeing what is happening-what it is that (s)he has created-and (s)he is making judgments about it at that level" Schön op cit). This understanding then serves as a "springboard to a new round of problem-solving inventions" or to put it another way, "as you work a problem, you are continually in the process of developing a path into it, forming new appreciations and understandings as you make new moves." [7]

Although this may strike resonance with software design practitioners, it is important to determine whether such occurrences can be found under 'experimental' conditions. How these can be captured. Whether there is any variation in their manifestation, the extent to which these bear similarities to other domains, or may provide new insights into the study of reflective practice.

The vagueness of Schön and researchers in reflective practice in defining reflection has been previously noted (eg [3]). Three types of reflection are recognized:

1. **Reflection-in-action.** This is associated to the experience of surprise. Sometimes, we think about what we are doing in the midst of performing an act. When performance leads to surprise-pleasant or unpleasant-the designer may respond by reflection in action: by thinking about what you are doing while doing it, in such a way as to influence further doing. The designer is reflecting in action, both on the phenomena that is being represented (through his drawing) and on his previous way of thinking about the design problem. Schön described this as 'backtalk'. Where you discover something totally unexpected-"Wow, what was that?" or "I don't understand this," or "This is different from what I thought it would be-but how interesting!" Backtalk can happen when the designer is interacting with the design medium. In this kind of conversation, judgments are made such as, "This is clunky; that is not," or "That does not look right to me," or just "This doesn't work." The designer's response may be "This is really puzzling," or "This outcome isn't what I expected-maybe there is something interesting going on here."
2. Stop and think. Here, the designer exhibits a **reflection-on-action**, pausing to think back over the activities in the project, and exploring the understanding that has been brought to bear on the task. This may include a new theory of the case, reframing the problematic design situation in such a way as to redefine, interactively, both means and ends.
3. **Reflection on practice**, the designer may surface and may criticize tacit understandings that have grown up around repetitive experiences of designing.

A design session may yield instances of all three types of reflection. The pilot study described below was designed to determine whether any of these types of reflection would occur**.**

2 Pilot Investigation

A pilot investigation was conducted in order to determine whether reflective practice occurs during software design. This necessitated constructing a short task, achievable in the space of three hours that would engage experienced software authors in design, development and testing.

2.1 Aims

The overall aims of the investigation were to:

1. determine whether periods of reflection in action, reflection on action and reflection on practice could be found in the verbalizations
2. consider the role of such reflections in mediating movement through the solution space
3. consider the usefulness of this approach in terms of design research and computing.

2.2 Task: The Sheepdog Game

Sheepdog is a real time game that can trace its ancestry to the early days of home computers when the blocky nature of the graphics available did not detract too much from the enjoyment of the game. The fact that blocky graphics are acceptable make this game suitable for the experiment, where we do not want authors to spend a lot of time on graphic design.

It also a fairly obscure game, compared to say Snake, or Space Invaders, and therefore is more likely to be a fresh task to the author, and one where many insights into the possibilities are likely to become apparent during the design process itself.

The principle of the game is that several "sheep" need to be herded into the sheep pen by the "sheepdog". The sheep move around on their own, but are influenced by the behaviour of the sheepdog, which is controlled by the player. There are different targets that can be set - e.g. time limits, no sheep to escape from the field, a point score ... and so forth.

Critical to the design is the way the sheep are programmed to move both in relation to their environment, and according to their intrinsic properties. At the most sophisticated level, environment that they are aware of could consist of the field edge, the pen, other sheep (position and velocity) , the dog (position and velocity),

The authors were encouraged to avoid programming peripheral elements of the game - e.g. options menus, high score tables etc, and to concentrate on the basic mechanics of the game.

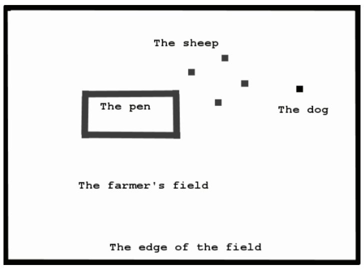

Fig 1. Representation of the game elements for Sheepdog

2.3 Method and Participants

Participants were required to be proficient in a third generation computer language, ideally with an interest in games programming. All were recruited locally, interviewed about their level of competence, assessed for their ability to talk out loud and told the purpose of the study and mature of the trial. Payment was offered.

Sessions took place in the Usability Laboratory at the Design Institute, Coventry University, which allowed sessions to be videoed and participants unobtrusively watched from the observation room. The researcher intervened to provide refreshments and talk to participants and remind them to keep verbalizing.

Prior to the session necessary applications were uploaded so that the participant would use a version of the software they were familiar with and provided with a written design brief (see section 2.2). They were asked to start thinking abut the task and develop the game whilst providing a verbal stream of consciousness. A period of three hours was allocated to the task. Although this would not give sufficient time to fo the programming of advanced features, it was considered long enough for the mechanics of a basic game to be developed in one session. All participants managed to complete and test the basic functionality of the game in this time.

The video capture allowed recording of the screen, the face of the participant and the verbalizations (using the Observer system). Screen events and verbalizations were transcribed so that the comments could be related to events on the screen – this association will be the subject of subsequent research. The transcriptions were chunked into meaningful sections on the basis of the topic being addressed (eg sheepdog behaviour). Each chunk was then further coded:

1. using an adaptation of the encoding scheme developed by Dorst [3] for description of design as a rational problem solving activity.
2. identifying periods of 'reflection–in' and 'on-action' and 'reflection-on-practice'.

A visual representation of the topics addressed by each participant (including the reflections) was produced using winCmap vsn3.6 (available from the Institute for Human and Machine Cognition) at http://cmap.ihmc.us).

2.4 Preliminary Results

2.4.1 Reflections

All participants reflected-on-action during the course of their programming. Charting these using winCmap showed that these occurred throughout the session and were integral to the activity, ie they did not occur when the participant stepped out of the activity. One participant did not reflect as much, but this might have been due to the experimental setting – some participants find it easier to produce a verbal stream of conciousness than others. These reflections were classified as relating to

- Their own ability, eg mathematical reasoning, programming and memory such as:

  "*hmm, I kept the wrong thing….I do that a lot, I'm very forgetful…you can find that takes so long*" subject 3 while changing his code

- Elements of the specific task eg game play
- Programming in relation to the task eg errors and coding strategy

  "*classic copy and paste mistake.*" Subject 3 while copying code for 'sheep behaviour' to the 'dog' and again

  "*there's probably nothing as much , that causes so many bugs as cut and paste*"

Given the nature of the investigation it was not anticipated that there would be many instances of reflection-on-practice that lead to changes in professional practice. This belief was confirmed. Where such reflections occurred these seemed to be a result of the experimental setting – more as an aside to the experiment. For example, Subject 1 in his last session made the following reflections regarding his practice. These were occurring outside of the main activity and were perhaps made solely for the edification of the researcher.

Time	**Transcript**
1.15	*"again, y'know I…something at the beginning of the whole project told me what the data structure should do, my general knowledge of this type of environment if you like, the sort of data structure would be useful and they've….I've been able to get the up and running and they…..I think they've been sort of*
1.30	*been useful in establishing um, where the problems are, the different ideas have given me a feel for the lie of the land…that acts as a sort of like an impetus for me then to go forward into pen and paper stage*
1.45	*Strangely enough I don't think it would have been worthwhile for me to have gone into pen and paper stage first because you are committing yourself to ideas there which might fall apart, um very early on in the process*

Reflections-in-action were the ones most closely associated with movement through the design solution space. Indeed many of the phrases used were those already cited such as, "wow", "oh no", "why's it doing that", "I didn't expect that…….", "that means that," and laughter. These occurred mainly during debugging or visualization of the code (in the graphics screen), occasionally when pen and paper were used to work out mathematics problems. Such surprises could be attributed to production errors (eg syntax, spelling, cut and paste problems), coding imprecision or simply not understanding the requirements of the solution. The latter two cases show that such reflections shape the movement through the solution space.

For example, Subject 2 in this section is checking and revising the code, while looking at the movement of the 'dog' on the graphics screen. This shows coding errors, surprises, backtalk, and an increased understanding of the requirements of the solution through engagement with the problem.

Time	**Transcript**
	<laughs>
7.30	*as it gets near the sheep it either vanishes or jumps into the nearest corner which isn't ideal*
	Oh I forgot to put that in
	One word and it does completely the wrong thing
	Christ… that doesn't…oh that's
8.00	*Oh…okay*
	I was doing greater than
	So if the dog was greater than 50 pixels, I'd say more whereas what I wanted to say was closer than 50 pixels
8.20	*And it doesn't seem to work*
	50 pixels is probably a bit big
8.30	*Let's change it to 20*
8.40	*Well its still very big*
	Can't actually move anywhere that doesn't make the sheep run away from me
8.50	*And if you're in the corner you're stuck*
	Uh
	Let's just make it something like 10 even
	'cos 10s as big as the er actual dog isn't it
9.20	*Why is it doing that?*
	Unless I've forgotten to put an 'and' in somewhere
9.30	*Oh so what I need to do is combine these two*
9.40	*Oh that's really stupid*
9.50	*Oh that's annoying*
	What its doing is changing its mind independently
10.15	*So it doesn't care if you are not near….. it depends if you are in line with the sheep*
	So I've got to change that
10.40	*I have a feeling this is going to get confusing*
11.00	*This is where bugs come in to programs*
	So what I'm doing now is making it a bit more exclusive and saying don't do this unless its near on the 'x' and 'y'

2.4.2 Parallels with design activity

Although not the primary focus of attention in this paper, the reflections also revealed evident parallels to other design domains. For example, in using a basic approximation to a solution as a starting point. For designers this may be a very basic sketch, for programmers it is rudimentary code as evidenced below.

> "*okay…here, I'm going to get something, anything at all working as fast as possible. That way I get a good feeling, and , er, instant gratification*," subject 3.

Also, in the necessary toleration of solutions that are not correct, but that allow progress to be made on other subgoals

> "*this is going to flicker a lot and I also know how to fix it, but I'm not going to bother for now…although its probably going to get irritating very quickly*," subject 3.

2.4 Discussion and Conclusions

This preliminary research has shown that it is possible to use design research methods to study programming activity and that periods of reflection-in-action, on- action and on-practice could be found. Not only do the reflections help to solve the immediate problem, they also mediate movement through the solution space.

The third aim of the investigation was to look at the verbalizations and in particular the reflections, to determine whether there is added value in looking at programming activity in this way. This will be addressed in the final section

3 Added value of this approach

The value of looking at design as reflective practice has made clear contributions to that discipline in terms of understanding what is means to work in practice, the process of single and group activity and the education of designers. Given the early stage of this work it is not easy to identify what the added value of such an approach may be. However, the research has identified four areas where contributions could be made.

- Training. Obviously training identified will be task dependent, but verbalizations are associated with times when the designer is experiencing problems due to a short fall in knowledge. In this study there was a clear need for additional training in mathematics (eg calculus) and help with visualization of 'x and y' co-ordinates and '+ and –' . All programmers had difficulty with logic in these areas, as evidenced by recourse to pen and paper, numerous changes to the code, and relying on running their programmes to understand where the errors were and when the code was not doing what was expected. The follow up interviews confirmed a need for top up training,

- Education. During the literature review, very few references to reflection or programming as a craft (in the design sense) were found. There may be opportunities for future research to look at the benefits of reflective practice and computer studios (as in design studios) where talk back is used more extensively to help the programmer understand the complexity of the situation. If such work is undertaken this may have major ramifications for the training of programmers.
- Development of programming support tools. Reframing of the problem occurred most when the code did not work as expected and all the simple production errors (eg typo's) had been identified. More complex errors may require the whole problem to be reconceptualised. Tools could be developed which supported this reframing.
- Furthering understanding of reflective practice. Having studied designers reflecting during concept design and programmers working at a similar stage of their design, it seems easier to understand what is happening in this domain, because it is text based media. A lot of design is visual, which causes additional problems for interpretation. The transcripts clearly show that a conversation is being entered into, and the graphics screen and debugging provide a voice for the author to listen to. This can be very rapid. However, it may provide added opportunities to study reflections. Examples of contributions from this study include 1) the scale of the reflections. When talking about reflective practice one can overestimate the importance of the reflections. These were small, they did not contribute to profound changes in practice, but very clearly effected task progression; 2) backtalk. The design research community has focused on pairs or groups of designers working together. This research has shown that designers, in this case software designers, enter a conversation with the design problem, by themselves, and that this can provide meaningful insights into activity; 3) research in the community, because it has focused on two-person conversations has not considered, to a large extent, the experimental effects such a situation may produce – where the participant may alter their behaviour or provide reflections that are not actually part of practice, but may be what the experimenter wishes to hear.

4 Reflection

This initial research project had very modest aims and fulfilled these by showing that it is possible to apply design research methods to software design and study the activity as reflective practice. Several shortcomings of he work can be identified such as the nature of the task ,it's duration, the expertise of the programmers. It has been difficult to engage the software community in the ideas behind the work, perhaps because of its novelty and the fact that few researchers are interested in software as reflective practice. It is hoped that the papers emerging out of this research may stimulate further research in this area.

It might be self evident that software authors are reflective practitioners in much the same way that everyone engages in reflection as part of their everyday lives, but few studies have considered this. However, for me one of the surprises has been that reflections do occur so often, they are small but do affect design progress – that the programme talked back – and given the scope of the project it was possible to capture some of these. Future analysis will look in more depth at the way the reflections mediated this movement.

Acknowledgements

The research was funded by the Arts and Humanities Research Board (AHRB) Small Grants in the Creative and Performing Arts, entitled Programming as Reflective Practice.

References

1. Buijs, J.A.: Design Education at the Faculty of Industrial Design Engineering, in Achten, H.H. (ed), Design Education in the Netherlands, TU Eidhoven (1997)
2. Cross, N.G.: Modelling the Creative Leap, Third International Round Table Conference on Computational Models of Creative Design, Heron Island, Australia, 3-7 December (1995)
3. Dorst, K.: Describing Design: A Comparison of Paradigms, PhD Thesis (1997)
4. Ericsson, K.A. and Simon, H.A.: Protocol Analysis: Verbal Reports as Data. MIT Press, Cambridge, MA, Rev. edition, (1993).
5. Gill, H.: The Nature of Problems, International Conference on Engineering Design, Boston, (1987) 185-190.
6. Kapor, M.: A Software Design Manifesto, (1996) at http://hci.stanford.edu/bds/ accessed 8/4/2005. reproduced from Winograd, T (ed) Bringing Design to Software, Addison Wesley.
7. Schön, D.: Reflective Conversation with Materials (1996), at http://hci.stanford.edu/bds/9-schon.html accessed 8/4/2005
8. Schön, D.A.: The Reflective Practitioner, Basic Books, New York (1983)
9. Schön, D.A.: Educating the Reflective Practitioner, Basic Books, New York (1987)
10. Simon, H.A.: Sciences of the Artificial, MIT Press, Cambridge, MA. (1967)
11. Reeves, J.W.: What is Software Design? C++ Journal, at http://www.bleading-edge.com/publications/C++ Journal/CpJour2.htm accessed 8/4/2005 (1992)
12. Roozenberg, N.F.M. and Eekels, J.: Product Design: Fundamentals and Methods, Wiley, Chichester (1995)
13. Walks, L.J.: Donald Schön's Philosophy of Design and Design Education, International Journal of Technology and Design Education, 11, (2001), 37-51
14. Winograd, T.: Bringing Design to Software,Addison Wesley, (1996)