Navigating Constraints: The Design Work of Professional Software Developers

David Socha
Computing & Software Systems
Univ. of Washington, Bothell
Bothell, WA, USA
dsocha@uwb.edu

Josh Tenenberg
Institute of Technology
Univ. of Washington, Tacoma
Tacoma, WA, USA
jtenenbg@uw.edu

CHI 2013 Extended Abstracts, April 27–May 2, 2013, Paris, France.
ACM 978-1-4503-1952-2/13/04.

Abstract
This paper reports on initial results from a study of software developers doing their authentic work in their place of work. We apply the ethnographic and interaction-analytic methods that the CHI community has used to study people carrying out their work in non-software domains. Our preliminary results show professional software developers spending the majority of their time navigating a myriad of largely invisible constraints arising from multiple, concrete, real-world sources. They use frequent *hypothesis-probe-interpret* to cycles navigate the contextual, complex systems that they inhabit and construct. These constraints are qualitatively different from those reported in the literature based on early conceptual design.

Author Keywords
distributed cognition; ethnography; interaction analysis; navigating constraints; software design; video analysis; workplace analysis

ACM Classification Keywords
H.1.2 User/Machine Systems: Human factors

Introduction

The CHI community has a history of studying people carrying out their work in its natural context in order to better design the tools and environments in which lived social practice occurs [4,8,15]. Going to the site of authentic work is critical to this endeavor because "work activities in every case take place at particular times, in particular places, and in relation to specific social and technological circumstances" [14]. Yet this analytic focus has rarely been turned inward, toward the situated social practices of those who develop computational systems, especially software developers.

In this paper, we describe the rationale, design, and preliminary results of an empirical study to answer questions about the processes, tools, representations, and patterns of interactions that software developers employ in carrying out their work. Our approach is to borrow the theoretical and methodological grounding that has been so successful in workplace studies in non-software settings, particularly distributed cognition [13] and interaction analysis [9]. We apply these to the software development setting that has often been studied from a cognitivist perspective [11] using interview and survey methods [5].

Background

A series of recent studies have begun to probe some of the complex work of software designers. In one study, Cherubini et al. [5] report on their investigation of "*how* and *why* developers draw their code," using surveys and interviews of Microsoft employees who reference completed diagrams to which they have access. The authors conclude that such diagrams are transient documents that use ad-hoc representations.

In order to obtain data with higher ecological validity and to look more directly at *joint* activity, two recent studies used video recordings of sketching and discursive activity during design sessions by software developers. Petre et al [12] video recorded the initial design activity of pairs of software developers who had previously designed together from three different commercial software companies. They then invited design researchers to separately analyze these recordings. The analyses give insight into such things as the way software developers vary their solution strategy at each point in time based on their "epistemic uncertainty" [3], the incremental and cyclical nature of solution development [2], and the positive effect of design planning on minimizing context-switching [16].

In another study, Dekel and Herbsleb [6] took videos of groups of software developers carrying out design exercises while participating in the OOPSLA *DesignFest*. Their results indicate that software designers use ad-hoc design representations rather than standard ones such as UML, and that these representations are meaningful only by those who were involved in their creation.

As important as this recent work is, the extent to which these studies provide insight for the design of tools, artifacts, spaces, and pedagogies for authentic joint software development work is limited by the contrived nature of the design problems, because the activity is captured outside the setting in which everyday work is undertaken, or the retrospective and linguistic nature of their data. As Petre et al. comment "[t]he ideal case involves studying real software designers, on a real software project, through the entirety of the product's lifecycle, and possibly beyond." [12:540].

We are in the midst of data analysis from a study of software developers doing their authentic work in their place of work. We describe the study and our preliminary results below.

Method

The organization that we are studying is an 8 year-old software development company in the Seattle area of the United States. This organization is owned by a non-US parent company, and employs about 50 people who all work in a single large room with no dividers (see Figure 1). The organization's product is a software system that helps friends and family share information. It has a significant backend Software-as-a-Service (SaaS) component, and includes client versions for Macintosh, Windows, and iPhone.

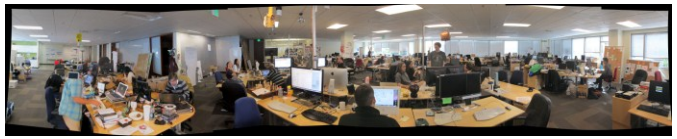

Figure 1. The open floor plan in our study organization was designed to promote pairing and assist collaboration.

So far, one of us (David) has visited the organization 5 times over a 6-week period. David spent 14 hours observing the activities and interactions taking place within the organization, taking notes and photographs, videoing software developers working together (9 hours in total to date), , and taking took over 300 photographs. Field notes are augmented and reflective memos are written immediately following each visit. We use interaction analysis [9] to do fine-grained analysis of the speech, gestures, tools, and externalized representations used by the software developers. We use ELAN [10] to replay and annotate the videos, often doing the analysis sitting side-by-side.

Results

The organization is very collaborative, with development practices based upon Extreme Programming and Scrum. Because their software developers usually program in pairs, their speech and gestures become public resources that enable joint work. This has the advantage of allowing us as researchers to obtain much of the information normally exposed by "think aloud" protocols [7], without having to impose the artificiality of such a protocol.

Our preliminary observations and analysis show the software developers' spending the majority of their time navigating constraints. We take a *constraint* to be when the digital, physical or social system in which they are operating or are trying to manipulate hinders their activity. These constraints are highly contextual, and often involve lack of knowledge, ambiguity, or misconceptions concerning such things as locations of data files, who controls access to such files, meanings of variables, and how large user-stored images can be without exceeding the contractual limit for data storage, to name just a few. Constraints are often unknown in advance, and arise when the developers and encounter them while carrying out their tasks.

Compared with early conceptual design sessions (Figure 2) based upon a 1-2 page design brief, such as used in many prior studies (e.g. [12]), the software developers we observed (Figures 1 and 3) are severely limited by an enormous set of concrete and often invisible constraints (see Table 1). Their constraints have accumulated and evolved over 8 years, 750K lines of

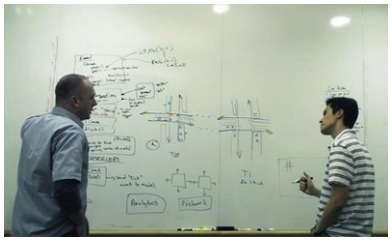

Figure 2. A pair of designers engaged in the early conceptual design task studied by Petre, et al [9].

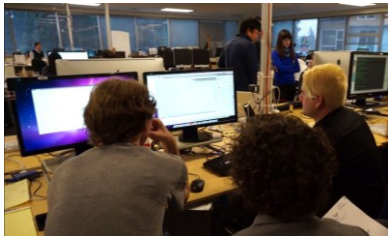

Figure 3. A pair of our subjects engaged in their daily work on their successful product. Note the third employee also engaged in this collaboration.

	Early conceptual design [1,12]	**Daily work on successful product [this paper]**
Macro goals	Inform tool design Inform pedagogy (Via workshop publications)	Increase Customer Lifetime Value (CLV) Increase Recency, Frequency, Monetary Value (RFM) (Via product changes)
Task derivation	Researcher & Educator	Business
Where design happens	Research lab / workshop	Business workplace
Focus / Final product	Model possible system	Deliver value to customers
Design team size	2 (in design session) 54 (in workshop)	2 (when pairing), 4 (when probing requirements), 50 (in organization), millions (customers)
Product domain history	None	Years
Team history	None - Some	Years
Commitment	2.5 hours (design session) 2.5+ days (workshop)	Livelihood; years
Nature of constraints	Highly visible (in prompt) 22 (in prompt) Simplistic context	Hidden / Bumped into Uncountable Highly multidimensional context
Existing code	• None	• 750K LOC
Existing data	• None	• Lots
Organizational	• None	• 8-year-old company with 50 employees
Contractual	• None	• 10+
Users	• None	• 13+ million
	About what COULD BE	Mostly about what IS
Length of feedback cycle (probing)	None (in design session)	Seconds – minutes (when pair programming) Minutes – hours (when probing requirements) Days – months (in organization) Minutes – months (for users)

Table 1. This table illustrates the dramatically different contexts between laboratory studies of early conceptual software design, and in-the-wild studies of professionals doing their authentic work on a successful software product. This table also shows the impact that these differences have on the nature of constraints that software developers deal with, and the length of the feedback cycles between the software developers and the system in which they are working.

code, and millions of users. Instead of conversations about general possibilities or abstract conceptualizations of an imagined system (as we see in studies of early conceptual design), we observed conversations about a fine-grained concrete understanding of what currently exists in their system as they bumped into constraints while trying to understand or change their system.

When an unanticipated constraint is encountered, a great deal of their conversations and actions are about running quick experiments to probe the system in order to better understand what it currently is and the resulting actions that it affords. The developers continually interact with their partners and the computer performing quick experiments of formulating a hypothesis, probing the system, and interpreting the results. These hypothesis-probe-interpret (HPI) cycles are surprisingly rapid. For instance, in a sample 2-minute portion of video, we identified 18 hypotheses, 6 probes, and 16 utterances that were associated with interpreting data.

Conclusion

The preponderance of constraints that we observed the software developers navigating is qualitatively different from the relatively unconstrained initial conceptual design sessions used in many prior studies (e.g. [12]). While all organizations probably spend time doing both types of design, it behooves us to further understand what it means for software developers to have to navigate such a huge set of concrete and often invisible constraints. Situations with those magnitudes of constraints are difficult to create outside of the complexity of workplace products. And even if we could create those in a non-workplace situation, there is distinct value in better understanding how professional software developers do their authentic work in their situated place of work.

We expect that further analysis will lead to implications for theory, tool development, practice, and pedagogy.

Acknowledgements

We thank the illuminating discussions with Kevin Sutanto, Kyle Patterson, and Skip Walter. We acknowledge the Helena Riaboff Whiteley Center at the Friday Harbor Laboratories of the University of Washington for providing a quiet and conducive space in which some of this work was carried out. Finally, we thank our study participants who have been so kind to extend to us the level of trust that is critical for this type of research.

References

1. Baker, A., Van der Hoek, A., Ossher, H., and Petre, M. Guest Editors' Introduction: Studying Professional Software Design. *IEEE Software 29*, 1 (2012), 28–33.

2. Baker, A. and Van der Hoek, A. Ideas, subjects, and cycles as lenses for understanding the software design process. *Design Studies 31*, 6 (2010), 590–613.

3. Ball, L.J., Onarheim, B., and Christensen, B.T. Design requirements, epistemic uncertainty and solution development strategies in software design. *Design Studies 31*, 6 (2010), 567–589.

4. Bell, G., Blythe, M., and Sengers, P. Making by making strange: Defamiliarization and the design of domestic technologies. *ACM Transactions on Computer-Human Interaction 12*, 2 (2005), 149–173.

5. Cherubini, M., Venolia, G., DeLine, R., and Ko, A.J. Let's Go to the Whiteboard: How and Why Software Developers Use Drawings. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07*, ACM Press (2007), 557–566.

6. Dekel, U. and Herbsleb, J. Notation and Representation in Collaborative Object-Oriented Design: An Observational Study. *OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, (2007), 261–280.

7. Ericsson, K.A. and Simon, H.A. *Protocol Analysis: Verbal Reports as Data*. MIT Press, 1993.

8. Jordan, B., ed. *Advancing Ethnography in Corporate Environments: Challenges and Emerging Opportunities*. 2013.

9. Jordan, B.B. and Henderson, A. Interacton Analysis: Foundations and Practice. *The Journal of the Learning Sciences 4*, 1 (1995), 39–103.

10. Lausberg, H. Coding gestural behavior with the NEUROGES-ELAN system. *Behavior Research Methods 41*, 3 (2009), 841 – 849.

11. Miller, G. The cognitive revolution: a historical perspective. *TRENDS in Cognitive Sciences 7*, 3 (2003), 141–144.

12. Petre, M., Van der Hoek, A., and Baker, A. Editorial. *Design Studies 31*, 6 (2010), 533–544.

13. Salomon, G., ed. *Distributed Cognitions: Psychological and Educational Considerations*. Cambridge University Press, 1993.

14. Suchman, L. and Trigg, R. Understanding practice: video as a medium for reflection and design. In J. Greenbaum and M. Kyng, eds., *Design at Work: Cooperative Design of Computer Systems*. Lawrence Erlbaum Associates, 1991, 65–89.

15. Szymanski, M. and Whalen, J., eds. *Making Work Visible: Ethnographically Grounded Case Studies of Work Practice*. Cambridge University Press, 2011.

16. Tang, A. What makes software design effective? *Design Studies 31*, 6 (2010), 614 – 640.