

Surveying the Ubicomp Design Space: hill-climbing, fields of dreams, and elephants' graveyards

Michael B Twidale
Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
twidale@uiuc.edu

Introduction

The famous quotation from the movie *The Field of Dreams* seems to guide so much techno-optimism from dot-com bubble business plans to ubicomp research proposals: “if we build it, they will come”. Sometimes they will. But not always. Is there a better way to explore a design space to help decide what exactly to build, rather than just picking the first idea that occurs to us and then focusing exclusively on how to make it work?

I believe that there is a way, by applying a range of high speed low cost techniques to more actively explore and analyze various design spaces. Even spending a few hours doing this can be worthwhile. My contention is that even this is rarely done. The challenge is to explore and refine those methods and to demonstrate their power and their relatively low cost. It is quite understandable that developers are somewhat skeptical of analytic techniques that appear to mostly slow down development. This is even more the case in the context of research activities where the development activity is itself part of the learning process and the requirements elicitation process.

The problem

In teaching undergraduates and graduate students with strong technical development skills I have noticed that unsurprisingly they want to build applications and start building as soon as possible. That is perfectly understandable and something I want to encourage. One learns a lot about a topic by trying to build an application. However the first idea one has is unlikely to be optimal. Once one has spent a lot of time on working on that design and emotionally committing to it, it can be very difficult to abandon it and start afresh. Available resources just may not permit it. I see such a scenario playing out time and again – an obsessive focusing on one point in the design space that was chosen arbitrarily and too early on. This creates a pedagogical problem and one that can be addressed in planning future classes. However, once noting it, I realized that it applies to many research projects too. The decision of what to build is made hurriedly and in ignorance, often as part of the rush of putting together a grant proposal and is not considered thoroughly. Very experienced researchers can draw on their previous work to make snap decisions in such circumstances and stand a good chance of locating a productive point in the design space. Less experienced researchers are unlikely to be so lucky.

In the case of novel ubiquitous computing applications, our intuitions can be wrong and even misleading. There is ample evidence from the field of CSCW research of how considerable experience in developing single user applications does not guarantee success in developing useful, usable and acceptable collaborative applications. It is most likely that the same occurs when we get up from the desktop and move around and think about workspaces, home spaces, social

spaces and moving between these and others. One approach is to acknowledge the problem, but apply a Darwinian, market based solution: just create a whole host of ideas embodied in applications as fast and as productively as possible and let the marketplace of ideas decide which are successful or not. Such a method certainly works but it is very wasteful. Is a more considered approach possible without excessively slow analysis getting in the way of design creativity and innovation?

A solution

The classic, powerful solution to this issue in CSCW is to first do a detailed ethnographic study. I am all in favour of these and have participated in some as both a systems developer and as an ethnographer. However they are very slow and often are better at telling you what not to build rather than what to build. Can such approaches be supplemented by other rapid analysis techniques? Methods that can be applied in minutes or a couple of hours and integrated into rapid prototyping iterations. Below I list some that I've been looking at. Ideally I think the real answer is ways to select from and combine these at different points in a design cycle, tightly integrated with exploratory development with an idea of build in the morning, test, analyse, plan redesign in the afternoon.

Affordance Analysis

Building on the HCI concept of affordances this looks at an individual component of a use case, either how people currently do something or our new proposed device and consider what are the generic kinds of activities that that component affords, supports or enables. Anti-affordances are also considered. In this way it becomes possible to consider a slice-and-dice solution where the new design does not attempt to do everything that the old design does and more and better in all cases, but rather focuses on supporting certain activities for which it is dramatically better and integrates with the old approach for the other activities. As a very simplistic example, an exploration of the technological and social affordances and anti-affordances of cellphone use in public settings can lead to ideas such as sound output but button-press input as a way to enable conversations in public settings where listening is acceptable but speaking is not. A very simple interaction might involve pressing a button that transmits a recorded message such as “I'm in a public place. I can listen to any message you want to give me, but I can't really talk right now. If you want to ask me questions I can press buttons to say yes or no. Otherwise, I'll get back to you in X [typed in] minutes”. Not exactly programming rocket science, but a very simple case of application innovation inspired by affordance analysis.

Goals, Constraints, Opportunities, Issues,

Inspired by SWOT analysis, this is a quick way of considering aspects a design space:

- Goals lists all the desirable features and uses of the envisaged application, acknowledging that some can be contradictory or lead to design tradeoffs.
- Constraints are limitations either of requirements and use or of the available technology or development environment.
- Opportunities lists advantages, most usually caused by new technologies having disruptive effects on traditional cost benefit calculations.
- Issues are things that arise in discussing potential applications and can include features of privacy, ownership, expectations, acceptability, trust etc.

Scenario Based Design, Personas, Body Storming

Building on a rich tradition in this area of trying to envisage the proposed application in actual use and then acting out these scenarios and critiquing them. The main new contribution here is to consider various forms of failure analysis, inspired by HCI cognitive walkthroughs. That is, one begins with the optimistic scenario of how the ubicomp app ought to work and so be a better way of doing things than the traditional approach. Then at each stage of the scenario we stop and consider “what could go wrong here?” Having identified problem, maybe of learning, use, interpretation, or systems failure, we then consider how people might cope and how the design might cope, either via a redesign to prevent the error, or to mitigate its consequences or to support recovery, or to inspire a rapid exploration of a different design solution altogether.

Creativity and Bad Ideas

The methods above can be characterized as explorations around a particular point on a design space, an attempt to do local hill climbing in that space rather than sticking purely with the point in that space of the first application idea. Working with Alan Dix, we have been looking at how to inspire design creativity to look at entirely different places on the design space. One method is the consideration of ‘bad ideas’ – design solutions and applications that are clearly bad. By then analyzing exactly why they are bad, one does a local design space exploration very similar to the effects of the techniques above. This can then lead to new ideas for applications that are slight variants of the original bad idea, or inversion of parts of that idea. We have written a couple of papers on that work and wish to explore it further.

Rapid prototyping

This approach is mostly about analysis, but the development of prototypes is a very important part of analysis. Very rapid prototyping is helpful. It has a lot of precedent with the classic story of the developer of the palm pilot carrying a block of wood in his shirt pocket and pulling it out periodically and imagining how it could help his life. In traditional desktop application development, paper prototyping has a long tradition in both interface design, but also forms of requirements capture, especially as part of participatory design. The mapping between sheets of paper and windows on a PC is pretty obvious. What is the equivalent of paper prototyping for ubicomp? Working proofs of concept can also be highly valuable as ways to inspire analysis of what really should be built, so long as they can be put together fast enough that they do not cause a lock-in forcing the project to stick with that particular design. We have been exploring the use of mashups, wikis and other pieces of open source software as

ways to produce very crude but operational elements of functionality to illustrate a design idea before committing to a particular implementation

Analysis as a process of exploring spaces

I’ve talked about ‘the design space’ as something to be explored semi-systematically as part of low cost analysis. In fact, it is impossible to fully explore this space. It is so huge it is not surprising that students cling to their first design idea. The immensity of the space can inspire a form of ‘cognitive agoraphobia’ making people reluctant to consider too many or any alternate design ideas for fear of getting swamped by choice and producing nothing. The techniques outlined above are ways to explore some small parts of this space without being overwhelmed. However there are several spaces:

- Design space: what we can build, combining features, functionalities, interfaces
- Adoption space: what people want, might want, like, don’t like
- Research space: build to learn, to think, to understand, to articulate
- Funding space: fashion, strategies, rhetoric, re-articulating, advocating

Related Work

- Dix, A. Ormerod, T., Twidale, M.B., Sas, C., Gomes da Silva, P.A., McKnight, L. (2006). Why bad ideas are a good idea. To appear in Proceedings of the First Joint HCI Educators’ Workshop.
- Jones, M.C., Floyd, I.R. (forthcoming). Patchworks of Open-Source Software: High-Fidelity Low-cost Prototypes. In “Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives”, K. St. Amant, B. Still (Eds). Idea Group, Inc.
- Jones, M.C., Floyd, I.R., Twidale, M.B. (2006). Patching Together Prototypes on the Web. Submitted as a Notes paper to CSCW 2006.
- Jones, M.C., Rathi, D., & Twidale, M.B. (2006). Wikifying your Interface: Facilitating Community-Based Interface Translation. Proceedings of DIS 2006.
- Jones, M.C., Twidale, M.B. (2006). Snippets of Awareness: Syndicating Copy Histories. Submitted as a Notes paper to CSCW 2006
- Twidale, M.B. & Jones M.C. (2005). “Let them use emacs”: the interaction of simplicity and appropriation. International reports on socio-informatics 2(2) 66-71.
- Twidale, M.B. & Ruhleder, K. (2004). Where am I and Who am I? Issues in collaborative technical help. Proceedings, CSCW04. 378-387.
- Twidale, M.B. (2005). Over the shoulder learning: supporting brief informal learning. Computer Supported Cooperative Work 14(6) 505-547.
- Twidale, M.B., Wang, X. C., & Hinn, D. M. (2005). CSC*: Computer Supported Collaborative Work, Learning, and Play. Proceedings, Computer Supported Collaborative Learning (CSCL), 687-696.
- Twidale, M.B. (2006). Worrying About Infrastructures. CHI 2006 Workshop: Usability Research Challenges for Cyberinfrastructure and Tools.