“Let them use emacs”: the interaction of simplicity and appropriation

M.B. Twidale, M.C. Jones
Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign
twidale@uiuc.edu , mjones2@gmail.com

Abstract. We look at appropriations in a number of contexts, and find that often they involve very simple ways of coordinating functionality that seems to be at odds with much of the approach of computer systems design that emphasizes power and abstraction. We speculate about how we can extend this simplicity in other ways.

Introduction

The infamous quotation attributed to Marie Antoinette has been used to illustrate how completely out of touch the *ancien regime* was from the circumstances of the starving peasantry, and consequently how inappropriate to the point of insulting was the proposed solution of eating cake. There are times when the aristocracy (or should that be priesthood?) of computer scientists can, by their proposed design solutions, appear to be equally out of touch from the needs of end users. Designing an incredibly powerful application with thousands of reconfiguration options and its own built in programming language to enable extensibility does not mean that it will be adopted, appropriated or tailored by end users for their own unique and evolving needs. Emacs is not the solution nor is it the ideal design paradigm. This issue persists despite the great progress made in advocating for user centered design and the development of techniques to integrate better understanding of real workplace needs and practices into systems development. Although at times this recurrent problem can be attributed to persistent developer

arrogance or ignorance of real user needs, we suspect that it is also in part due to a fundamental part of the design ethos that most computer scientists absorb both in formal education at universities and in ongoing professional practice. This ethos involves a concentration on functional power, flexibility and abstraction in the development of effective and elegant design solutions. It has enormous merit in nearly all contexts and in no way are we advocating for its erosion. But we do at least fear that it can sometimes get in the way of end user technological appropriation. In this concept paper we attempt to outline some of our thinking about simplicity and appropriation in the light of experiences with two different research projects, and some initial work in trying to characterize the aspects of the appropriation process.

Given that different people use the word 'appropriation' slightly differently, we should clarify that we are not so much interested in the adoption of a technology by types of people who were not expected or even allowed to use it (e.g. Eglash 2004). Rather we want to focus on cases of innovative use of that technology in ways that its developers had not envisaged, planned or explicitly designed the tool to support. That is, we concentrate on appropriation-as-innovation rather than the alternate (equally legitimate) view of appropriation-as-empowerment. In this way the users of technologies can be involved in some way in the co-development of the technology (Fischer 2002), without having to turn themselves into computer scientists in order to be technology developers.

Paper Prototyping from Forms into Databases

A study (Twidale & Marty 1999, 2000, Marty 2005a, b) of workflow practices in a museum that was simultaneously undertaking a digitization and a packing process prior to a move to a new building revealed numerous cases of end users appropriating aspects of the available technology to meet their needs. With 40,000 artifacts to pack and move, and relying on significant numbers of keen but relatively unskilled undergraduate workers, the museum staff had to design a system so that no artifact could get lost, even though they knew that many mistakes would inevitably be made. Worse, they had never packed and moved their entire collection before, so the workflow process itself would have to change as they learned by doing. External shocks would inevitably force yet more changes to the process – due to changing budget constraints, building schedule changes, etc. The packing process itself involved various paper forms that were used to record who had done which step of the workflow on which day for any given artifact. This allowed for the tracking of all actions and the rollback and recovery from errors. What is notable is how these forms, especially the packing sheet, evolved over time. People would use the form in ways it had not been intended for, writing additional information and notes about exceptional issues

with a particular artifact in the margin. Some of these ad hoc solutions to a particular problem with a particular artifact gradually evolved into practices that were applied to other related exceptions or to whole categories of artifacts. The forms themselves were revised so that later printouts for subsequent artifacts now had special fields and checklists derived from those handwritten practices. The database itself (created in-house using Filemaker Pro) evolved in the light of those changes to provide additional fields and automated checks. Effectively this was a kind of participatory design using paper prototyping. However, unlike conventional participatory design, the paper prototypes were part of the real work process and were actually used, rather than being developed purely to inform the design process.

In a similar way, certain aspects of the database and how it was used were appropriated by people who were not themselves database designers or any kind of programmer to enable them to do their job more effectively. In entering data about an artifact into the database, sometimes a user (particularly an undergraduate) wanted to record their doubts about a particular value. Rather than just guessing and moving on they might enter a value followed by question marks or even type in a comment such as “I don’t know”. These were messages through the database to other users of that database in the future – maybe the same person as entered the database, but several thousand artifacts later. Such commentary or annotation is very common in paper records (as marginal written comments or addition pieces of paper clipped or attached to a form), but is rarely seen in databases, where the prevailing assumption is that all uses of the database can and should be planned in advance of actual data entry. This might be desirable for optimal design, but we suspect it is rarely achievable in reality. Similarly, the database was used for various checking processes in later stages of packing and unpacking. Sometimes people would deliberately enter a value into the database that they knew would trigger an error for that particular artifact at one of these later checking stages. This allowed them to pass a message forwards in time to do an extra step with that artifact at that time that they knew would otherwise get forgotten about. Again, in the physical world this is entirely unremarkable – scrawled marginal notes, sticky notes and supplemental pages paperclipped to the main paper form are the norm in paper workflows, but exceptional or at least rarely remarked upon when it comes to databases.

This particular workplace study was admittedly an extreme case where the organization really did not know much about what precisely they wanted to do, had no experience with the moving process and yet had to get started, but we believe that the problem of designing for change is one that should pervade CSCW systems development.

Copy and Paste as Computational Duct Tape

A study of workplace help-giving in a variety of different settings (Twidale & Ruhleder 2004, Twidale 2005) revealed that in almost all circumstances people were using more than one application at a time. To achieve the work related goal, it was often the case that an element was copied from one application and pasted into another. Documents were frequently attached to emails and passed on to others for comments and further copy and pasting. The use of copy and paste was particularly evident when problems arose and a colleague was asked for help. If a direct solution could not be found quickly, a workaround was developed and again this frequently involved more than one application and copying and pasting to enable the work to be done. Hopefully this kind of interaction is so familiar as to be entirely unsurprising, and yet we believe it to be worthy of more detailed consideration of what it achieves, how it achieves it, and what might be done to improve or extend the approach. Effectively the people studied were creating very complex and sophisticated workflows, but without the use of or benefits from a workflow system. The workflow was mostly mental as they copied and pasted between applications and handed over work elements to others either via email or a shared database. A conventional workflow system might well have helped them optimize their standard actions, but the power of copy-paste was in its ability to deal with exceptions, and especially those whose resolution meant yet another workaround with yet another application, and possibly another person to help out. For example, a report may need some data in a particular form and style. The data might be collected by a number of searches in an internal database, and on the web and pasted into Excel so that all the results could be composed. A graph might be drawn in Excel, but require some tweaking that was done in Paint before pasting it into a Word document and passing it on to a colleague for help. Such a process might be neither elegant nor optimal (the user might be unable to use the report generation facilities in the database and hence be forced to manually compose their data in Excel, and likewise not want to bother with the full power of Excel graphing and so do some tweaking in Paint just because she happens to be more comfortable with that tool), but it does get the job done and can cope with a slightly different job next time.

Non-Computational Appropriations

We are accumulating a set of real-life appropriations to help us in trying to understand the aspects of design that seem to afford appropriation. Non-technological appropriations are particularly easy to collect as stories from non computer scientists. The physical nature of these appropriations can also help in thinking about what helped afford them, and also help in thinking about design for

appropriability both in conventional applications and those exploiting aspects of tangibility found in many ubicomp settings. Here are two examples:

Ad hoc conference calling

The first happened to one of us, amongst a group of very technologically savvy people. It shows two cell phones clustered around an office phone. The problem was to coordinate a synchronous distributed telephonic meeting – hardly a complex activity, and particularly ironic as it was for a CSCW design project. The solution illustrated of putting together two cell phones each with a connection to a remote participant alongside a desktop phone with a speaker option connected to a third remote participant is in many ways inelegant. The sound quality was not perfect, but enough to get the work done. Given various constraints of time and technology (including time to learn about various possible other more elegant solutions) the appropriation of physical affordances illustrated in the photograph was good enough to get the job done, and in many ways faster and better than other experiences we have had with trying to set up synchronous distributed meetings with far more sophisticated technologies. How did it come about? The fact that the nature of the solution is comprehensible to the reader solely from the photograph and without complex explanation of what was done with a CSCW infrastructure is, we believe, an interesting clue. The idea that putting the phones near one another will enable some sort of rough and ready conference call does not seem to require much of a flash of design inspiration, far less in fact than is often needed to figure out how to set up conference calling on most purpose-built phones. In this example the tangibility of the different artifacts and the ease in which different functions are perceived as being associated with different places (microphone, speaker), seem to help in affording appropriation. By contrast, the conventional program with many features all treated as a single complex unit seems to get in the way. We wonder if these aspects of tangibility and separability mean that it will be easier to design for appropriability in a ubiquitous computing context.

Raffle tickets

This example is interesting because we believe it to be the case where the appropriated use (raffle tickets) is far more prevalent than the originally intended designed use (coat checking). Coat check tickets make ideal raffle tickets because they both need unique tally options, are cheap and easy to separate, and both serve a similar job of separating and bringing together in the future. The idea of using the physical device for a quite different purpose that happens to exploit the affordances outlined above has propagated so much as to dwarf the originally intended use.

Conclusion

As our examples have illustrated, many of the successful appropriations that we have observed have involved very simple coordinations of technologies and technological elements that may have relatively limited functionality, and so may be regarded by end users as simple, even if their programmatic execution is complex (but hidden). The simplicity of copy-paste, or writing things into a database field definitely seem to have contributed to the success of the appropriations. But does that mean that simplicity is a necessary attribute of appropriability? Does that mean that we should give up on developing more sophisticated tools, interfaces and functions and just concentrate on these simple, rather mundane-looking ones? We are sure that it is not a good idea to go to the extreme of full power emacs-like generic toolkits, but we hope that it is possible to develop functionalities that offer slightly more power than copy-paste.

One aspect of the issue seems to be the interaction of granularity and connectivity. Too fine a granularity and one ends up with the power but the associated complexity of emacs and programming languages. Too coarse a granularity and one has purpose built applications that may have some pre-planned built-in tailoring and customization options, but that do not easily support innovative re-use. Connectivity via widespread provision of copy and paste facilities allow for combining functionalities between applications but without the power and complexity of semi-automation. Automation, iteration and branching get to the heart of computing, both the power and the complexity of programming. No matter how helpful, congenial, benign or graphical the interface, once these features are available, end users are in some way programming and will need to acquire the basic concepts of trying to plan for unexpected and unwanted outcomes such as dead ends, anomalous inputs and outputs and appropriate termination. The manual control of the flow of control seen in our examples looks computationally weak, but paradoxically may be a strength in terms of supporting imaginative leaps, simple experimentation and robust ad hoc usage.

It can be rather disconcerting for computer scientists to consider this aspect of appropriation, should further evidence accumulate to substantiate our claim. The training of computer scientists frequently emphasizes certain design aesthetics. Power and sophisticated functionality are naturally valued. It would seem very odd to extol one system because of how *little* it did compared to another. Even if such a minimalist design aesthetic is adopted (and there is evidence for the Open Source usability debate that that is not a foregone conclusion), there is also the problem of the abstraction and extensibility aesthetic. This is sometimes typified as: 'don't build a thing – build a thing builder'. It means that designers typically aim for generic solutions to problems to maximize power and re-use. This can mean incorporating many tailorability options, configuration settings, and even a programming language within an application so that it can be modified to address a host of future unanticipated needs. The concept of the abstract data type reveals the nature, and the power of the approach. But can such raw power (apparent in many Unix commands – and of course emacs), which is so useful to a skilled programmer, actually be a handicap for the end user in appropriating for any purpose? Can even the abstract names given to such functionalities deter appropriation by end users? As our work in this area continues, we plan to systematically explore the different features that support appropriation and to develop illustrator applications to test out those features.

References

Eglash, R. (2004). Appropriating Technology: An Introduction. In R. Eglash, J. Crossiant, G. Di Chiro and R. Fouché, (eds.) Appropriating Technology: Vernacular Science and Social Power. University of Minnesota Press.

Fischer, G. (2002). Beyond "Couch Potatoes": From Consumers to Designers and Active Contributors. First Monday 7(12)

Marty, P. (2005). Factors Influencing Error Recovery in Collections Databases: A Museum Case Study. Library Quarterly. In Press.

Marty, P. (2005). Factors Influencing the Co-Evolution of Computer-Mediated Collaborative Practices and Systems: A Museum Case Study. Journal of Computer-Mediated Communication. In Press.

Twidale, M.B. & Marty, P.F. (1999). An Investigation of Data Quality and Collaboration. Technical Report ISRN UIUCLIS--1999/9+CSCW.

Twidale, M.B. & Marty, P.F. (2000). Coping with errors: the importance of process data in robust sociotechnical systems. Proceedings, CSCW'00, Philadelphia, 269-278.

Twidale, M.B. and 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. To appear in Computer Supported Cooperative Work: The Journal of Collaborative Computing.