Greenberg, S. (2009) Promoting Creative Design Through Toolkits. In Proceedings of the Latin-American Conference on Human-Computer Interaction (CLIHC'09) CPS, 2 pages, November 9-11.

Promoting Creative Design through Toolkits

Saul Greenberg
Department of Computer Science, University of Calgary
Calgary, Alberta T2N 1N4 CANADA
saul.greenberg@ucalgary.ca
http://www.cpsc.ucalgary.ca/~saul

***Abstract*—Computer science academics and professionals typically consider their contributions in terms of the algorithms, applications, and techniques that they develop. Yet equally important are the tools computer scientists provide to others, including toolkits, libraries, APIs, SDKs and frameworks. Such tools radically shape how most developers think about possible solutions within an unfamiliar problem space. In this keynote, I describe how interface toolkits for novel application areas enhance the creativity of 'average' developers. By removing low-level implementation burdens and supplying appropriate building blocks, toolkits give people a language to think about new interface genres, which in turn allows them to concentrate on creative designs. As a consequence, programmers can rapidly generate and test new ideas, replicate and refine ideas presented by others, and create demonstrations for others to try. To illustrate, I describe example toolkits we have built and how people have leveraged them to create innovative interfaces.**

Keywords-toolkits, creativity, interface design.

I. The Computer Programmer as Designer

Computer programmers are often charged with designing – at least in part – the user interface of a product. While some programmers have solid design experience and are able to produce quite creative interfaces, most average programmers do not. Consequently, many programmers take the path of least resistance. They rely on their existing knowledge of how things are done in a particular genre, and use standard tools to mimic current styles. For example, most programmers depend upon the standard user interface widgets and controls available in a development environment to create an interface that (more or less) looks like other graphical user interfaces.[1]

Yet today's world is full of new technologies that can go far beyond standard windows/icons/mouse interface. The problem is that the average programmer can rarely use these technologies as part of one's design. Even when the programmer wants to do something different, he or she often has to work around system limitations and program at a very low level. Because this type of programming is costly, it often goes undone or it becomes a poorly functioning hack. Because programmers lack appropriate design training and only have conservative tools, most of the designs they produce are uninteresting variations of interfaces that have been around for decades. The consequence is that innovative interface design is stifled.

II. Interface Toolkits As a Creative Medium

Interface toolkits – software libraries, APIs, SDKs, frameworks, and supporting architectures – are one way out of this dilemma, for they make creative idea exploration simple to do in a particular domain. That is, we as a community must develop toolkits appropriate for everyday programmers, where such programmers can use them to develop their own creative ideas in a new domain. By appropriate, I mean that a good toolkit should (see also [2]):

- be embedded within a familiar platform and language in common use so that people can leverage their existing knowledge and skills;
- remove low level implementation burdens;
- minimize housekeeping and other non-essential tasks
- encapsulate successful design concepts known by the research community into programmable objects included with little implementation effort;
- present itself through a concise API that encourages how people think about that domain;
- make simple things achievable in a few lines of code, and complex things possible.

If we remove low-level implementation burdens and supply appropriate building blocks, we provide people a language to think about design [1], which in turn allows them to concentrate on replicating and varying designs in creative ways.

Perhaps surprisingly, the benefit of good toolkits is much more apparent in non-programming communities. Consider hypertext tools. In 1987, Apple produced HyperCard. Its building blocks were simple: a fixed-sized card that could be linked with other cards, bitmap images that could be drawn onto cards, a few UI controls such as buttons and form fields, a direct manipulation development environment, and (for 'advanced' use) a rudimentary scripting language. Yet that was enough to cause an explosion of creativity, where non-programmers (especially teachers) created a rich and varied repository of hypertext applications. Later, HTML combined with a browser's ability to transparently handle networking, spread hypertext page and site authoring to the masses. Today, designers using powerful authoring tools such as Adobe Flash to create a plethora of interactive, multimedia and highly animated web pages that have considerably elevated expectations of web browsing experience. Wikis, blogs and social networking sites have trivialized easy

[1] A full version of this paper, replete with examples, can be found in [1].

construction of template-based but still quite personal web sites. All these tools embody the above-mentioned criteria, and wide-spread creativity happened as a consequence.

III. OUR EXPERIENCES

In our group's investigation of the human and technical factors of 'unconventional' interface genres over the years, we recognized a recurring pattern, illustrated in Figure 1. that led to our appreciating the critical importance of good tools to the creative process.

1) ***Human factors perspective.*** Our initial goals were typically oriented towards human factors. Essentially, we wanted to understand how people interact when using a particular style of yet-to-be developed application. We would then generalize this understanding to inform other designs (Figure 1a).

2) ***Initial prototype.*** Next, we would set about building the first version of the application. This typically involved huge effort as measured by lines of code, time, learning, failed attempts, debugging, and so on. In spite of this effort, the result was often a fragile and rudimentary system (Figure 1b).

3) ***Prototype testing.*** We would then have people try out this prototype. Because it was an early design, we often saw major usability problems that required fixing (Figure 1c).

4) ***Design blocked for iterative prototyping***. To fix these usability problems, we would iteratively redesign the prototype. Yet this often proved impractical. The prototype code was frequently too complex to change, or the system itself was too fragile. Redesigning from scratch, while possible, was onerous due to the time involved (Figure 1d).

5) ***Retrenchment: building a toolkit.*** We would then realize that—in the long run—iterative prototyping would be far easier if we took the time to build a robust toolkit. Thus we would set ourselves a new technically-oriented goal, where we would delve into the challenges of understanding and building this toolkit and its accompanying run-time architecture. This often meant that defering work on our main human factors goal (Figure 1e).

6) ***The payoff: rapid prototyping.*** After building the toolkit, we would release it to our internally community. There would then be an explosion of activity. Those with core interests in the human factors challenges would rapidly develop and test a variety of interaction techniques and applications. Those with interests lying elsewhere would often create innovative applications as a side project just to satisfy their own curiousity (Figure 1f).

7) ***Testing, improvement and dissemination.*** Because we would develop the toolkit and applications side by side, we would bring well-tested good application ideas back into the toolkit as building blocks that could be trivially included in other applications. Of course, both prototype testing and our experiences in rapid prototyping fed back into our understanding of the basic human factors behind design (Figure 1g), thus acheiving our original project goal.

The keynote talk will briefly summarize the experiences my research group has had with several toolkits that we developed over two decades. Samples will be selected from various domains including: real time distributed groupware, single display groupware, physical user interfaces, and/or proximity-based interfaces. These experiences serve as case studies that show how toolkits helped promote creativity in rapid prototyping and in idea replication and variation.

As researchers, we can promote creativity in innovate areas through several means.

- Package the good lessons learnt from 'one-off' system design as reusable components (or as a clean API) within a toolkit.
- Make them easy to learn by supplying documentation, examples and tutorials.
- Disseminate these tools within our community.
- Recognize toolkit creation as an academic contribution.
- Encourage the inclusion of these tools within mainstream development tools.

REFERENCES

[1] Greenberg, S. (2007) Toolkits and Interface Creativity. Journal Multimedia Tools and Applications (JMTA), 32(2):139-159. (Special Issue on Groupware) Springer, February.

[2] Olson, D. Evaluating User Interface Systems Research. Proc. ACM UIST (2007).

[3] Whorf, B. L. Language, Thought and Reality (ed. J. B. Carroll). Cambridge, MA: MIT Press. (1956)

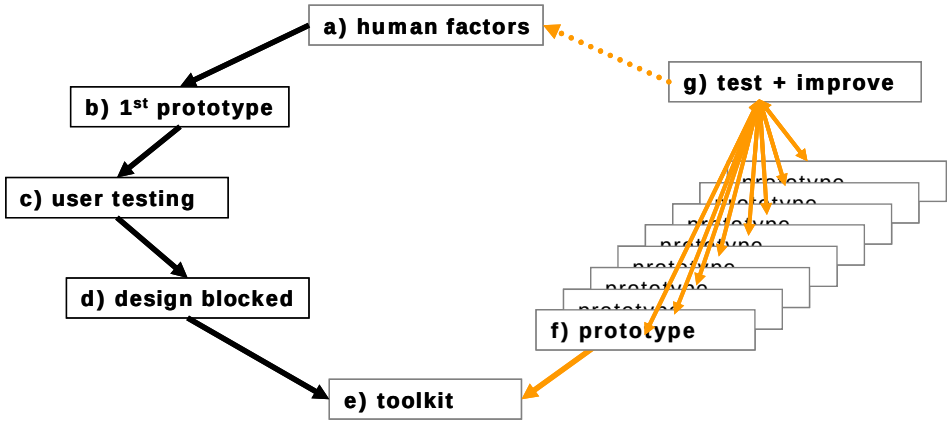

Figure 1. A recurring pattern: How toolkits promote rapid prototype designs.