SCOTT R KLEMMER

This statement highlights my three research threads that investigate how software tools can increase the quality of people's creative work—especially interface design and programming. The first explores leveraging online examples of creative work to empower more users to design new user interfaces and software programs, learners to acquire new skills, experts to be more creative, and programmers to engage in more design thinking [1, 2, 13, 24]. The second introduces techniques for designers to rapidly create novel user interfaces, explore more alternatives, and revise prototypes based on feedback [8, 9, 11, 12, 14-17, 19-23, 26, 29]. Third, my collaborators and I study the psychological and social ingredients of design excellence—focusing on the role of alternatives and prototyping [3-7, 10, 18, 25, 27, 28].

EXAMPLE SEARCH AND ADAPTATION

On the Web, people are increasingly sharing examples that show how to implement programming and design concepts. Furthermore, the *content of the Web itself* provides a giant repository of examples. We hypothesized that both types of examples contain significant latent value for creative work [2]. To understand this, our research strategy is to create software tools that embody hypotheses, and perform experiments to assess whether the principles manifest in the tools benefit their users. We have explored opportunities for programmers in the Blueprint project led by my PhD student Joel Brandt, and for designers in the Bricolage project led by my PhD student Ranjitha Kumar.

WITH INTEGRATED SEARCH AND DEVELOPMENT, HIGHER-QUALITY CODE MORE QUICKLY

When a person seeks to program new functionality (such as how to render a list of elements from a database), they face the "build or borrow" question: implement it from scratch, or find and adapt existing code? The increased prevalence of online source code—shared in code repositories, documentation, blogs and forums—enables people to opportunistically program applications by iteratively searching for, modifying, and combining examples [3]. For a simple reminder, a quick Web search and a glance at the search result is often adequate. To learn a new technique requires more time: programmers often perform several searches, and aggregate and compare information across multiple sites.

Currently, Web search has no information about searchers' development context, and code editors are oblivious to the provenance of code users' enter. We hypothesized that integrating search into programming environments can help people find and adapt examples more quickly. To test this, we created *Blueprint*, a Web search interface integrated into the Adobe Flex Builder development environment [1]. Two insights drove Blueprint's design. First, *embedding search into the development environment* allows the search engine to leverage the users' context (*e.g.* programming languages and framework versions in use). This lowers the cost of constructing a good query, which improves result quality. Second, *extracting code examples from Web pages and composing them in a consistent, code-centric search results view* reduces the need to click through to Web pages to find example code. This allows users to evaluate results much more rapidly than with traditional Web search interfaces, reducing the cost of selecting a good result.

In a controlled experiment, professional programmers using Blueprint completed tasks 28% faster—and also wrote significantly higher-rated code—than those using traditional Web search. Thousands have used Blueprint since it was made available for download in May 2009. We have compared Blueprint' logs to those of Adobe Community Help; both systems use the same corpus. This comparison found that Blueprint's example-centric results pro-

vided users' with sufficient information much more often: with traditional results, users needed to click through to the underlying page three times more often. It also found that users re-found information 57% more often with Blueprint: when search is cheap and effective, people more often delegate remembering details to the Web.

ENABLING ANY PAGE ON THE WEB TO BE A DESIGN TEMPLATE

The "view source" option in Web browsers provides an invaluable learning resource: being able to see how a Web page is constructed enabled legions of people to learn by viewing and modifying the source code from other pages on the Web. The Web today provides a vast corpus of diverse examples. Unfortunately, this powerful resource is underutilized. Adapting Web source to a new design remains a time-intensive, manual process: adapting a design requires understanding complex source code; furthermore, content and code are intertwined. Consequently, most design reuse today is accomplished via templates. However, templates homogenize page structure, limit customization and creativity, and yield cookie-cutter designs. Ideally, tools should offer both the ease of templates and the diversity of the entire Web. What if any Web page could be a design template?

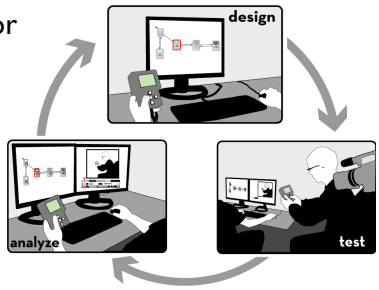
To enable novices and experts alike to more creatively use examples, *Bricolage* introduces a structured prediction technique for transferring layout and style between Web pages [24]. The Bricolage algorithm learns to identify visually and semantically similar regions between pages by training on a set of human-generated mappings. Each page is segmented into a tree of contiguous regions, and mappings are predicted between pages by identifying elements in these trees. Bricolage introduces a novel tree-matching algorithm that allows local and global constraints to be optimized simultaneously. The algorithm matches similar elements between pages while preserving important structural relationships across the trees. Bricolage provides a general, efficient, and automatic technique for designers to retarget their content to the layout and style of any HTML page. This new platform provides designers with the capability to rapidly explore many alternative design styles to find one that best suits their goals.

Furthermore, a fundamental challenge facing Web design today is the increasing heterogeneity of devices. The New York Times, for example, produces print, Web, Reader, iPhone, and iPad versions of its content. Five separate design teams is a major resource commitment for *any* organization – and impossible for most. We are researching how Bricolage can enable designers to rapidly and effectively retarget content to alternate form factors such as mobile devices. Initial results are promising. An interesting outcome of Bricolage's optimization approach is that the mapping cost can be repurposed to measure design similarity. This might valuably be used to create a (semi-)structured browsing interface for design alternatives. Such structure could make the exploratory stages of design much more efficient and effective, enabling designers to quickly find a relevant strategy.

INSIGHTFUL PROTOTYPING TOOLS

Prototyping is fundamental to design innovation, collaboration, and creativity [10]. While prototyping tools are now common for graphical user interfaces on personal computers, prototyping interactions for ubiquitous computing systems remains out of reach for designers [18]. How can tools enable more people to create interactive prototypes of ubiquitous

computing interfaces (like a mobile device with new kinds of sensor input)? And how can design tools support the larger process of learning from these prototypes?



AUTHORING SENSOR-BASED INTERACTIONS BY DEMONSTRATION

Prototyping tools enable designers to create graphical user interfaces quickly and visually: no code required to get started. How might tools offer similar benefits when interfaces break out of the desktop screen to use richer sensing? Our d.tools research, led by my former PhD student Björn Hartmann, introduced techniques that lower the expertise threshold required to author novel physical interfaces [12]. The d.tools hardware platform gives designers plug-and-play sensors and actuators by providing smart components that communicate on a shared bus. d.tools extends designers' existing storyboarding practices through a visual, statechart-based design interface. As designers move from early-stage prototypes to higher fidelity prototypes, d.tools augments visual authoring with scripting.

The complementary piece to rapid interface creation is to provide a more natural means of specifying the relationship between raw sensor signals and application behavior. Exemplar extends d.tools, introducing a demonstration-based approach to bridge the conceptual gap between conceiving and implementing sensor-based interaction [9]. With Exemplar, a designer first demonstrates a sensor-based interaction to the system (*e.g.*, shaking an accelerometer). The system graphically displays the resulting sensor signals. The designer then marks up the part of the visualization that corresponds to the action. Instead of requiring designers to explicitly author a recognition algorithm or filter, Exemplar learns appropriate recognizers from these markups. The designer can review the learned actions through real-time visual feedback and modify recognition parameters through direct manipulation of the visualization. The goal of both systems is to enable users to focus on design thinking (how an interaction should work) rather than implementation tinkering (how hardware and sensor signal processing work).

DESIGN AS EXPLORATION: PARAMETRICALLY CREATING AND TUNING INTERFACES

Designers prototype to learn and elicit feedback. Creating multiple prototypes facilitates comparative reasoning, grounds team discussion, and enables situated exploration. However, current interface design tools focus on creating single artifacts. How might interaction design tools explicitly support creation and management of multiple user interface alternatives? Our Juxtapose source code editor and runtime environment introduced the ability to design multiple alternatives of interaction designs in parallel [15]. In the code editor, designers can define multiple program alternatives through linked editing, a technique to selectively modify source files simultaneously. The set of source alternatives are then compiled into a set of programs that are executed in parallel. Juxtapose generates a control interface for application parameters through source code analysis and language reflection. To show that general strategy of parametric alternative creation applies in other domains, we developed Juxtapose runtime environments for mobile phones and microcontrollers.

Many prototypes go through team discussions and reviews before being tested. For text documents, revision management effectively enables asynchronous collaboration. No equivalent functionality existed for interfaces. d.note introduced a revision notation for insertion, deletion, modification and commenting on appearance and behavior of interface prototypes [11]. d.note realizes three benefits: it visually distinguishes tentative changes to retain design

history, allows for Wizard of Oz simulation of proposed functionality, and manages display of alternative design choices to facilitate comparison.

When designers test prototypes with users, they often record these sessions on video. The d.tools video suite provides integrated support for testing prototypes with users and rapidly analyzing test videos to inform subsequent iteration. d.tools logs all user interactions with a prototype and records an event-synchronized video stream of the user's interactions. The video is automatically structured through state transitions and input events. After a test, d.tools enables designers to view video and storyboard in parallel as a multiple view interface into the test data. Two novel query techniques shorten the time to find relevant segments in the video recordings: query by state selection, where users access video segments by selecting states in the storyboard; and query by input demonstration, where designers demonstrate the kind of input that should occur in the video.

RAPIDLY DESIGNING ALTERNATIVES

Iteration is central to learning and motivation in design. Yet its primary virtue—incremental, situated feedback—can also blind designers to other alternatives, steering them to local, rather than global, optima [7]. To combat this, creating multiple alternatives in parallel may encourage people to more effectively discover unseen constraints and opportunities, enumerate more diverse solutions, and obtain more authentic and diverse feedback from potential users. To date, design process decisions are largely faith-based rather than research-based. To address this, my post-doc Steven Dow and I have investigated the relative merits of parallel and serial prototyping. In a between-subjects experiment, participants designed Web advertisements [6]. Participants worked independently and were given equal time to create each prototype and read critique. Parallel participants outperformed Serial participants by all performance measures: click-through rates, time spent on the target client website, and ratings by the clients and ad professionals. Further, independent raters found that the diversity of each participant's prototypes was greater in the Parallel condition. Parallel participants reported a significant gain in self-efficacy, a measure of task-oriented confidence. Serial participants did not. In short, parallel prototyping yields higher-quality results, more divergent ideas, and its practitioners respond more positively to critique.

Rapidly creating alternatives benefits individuals; how does it affect groups? Prototypes help summarize ideas, demonstrate progress and expertise, and ground group communication and decision-making. However, concreteness may focus discussion on refinement rather than exploration—even when inappropriate. Furthermore, presenters often believe their status is on the line, and consequently overinvest in a single concept. Compounding this, collaborative work is susceptible to groupthink, where members reinforce each other's beliefs at the expense of other options. Distributing investment across multiple prototypes can reduce fixation and sunk-cost reasoning. We investigated the hypothesis that sharing multiple prototypes increases design performance, improves group interaction, and increases idea sharing [5]. A between-subjects experiment where pairs of participants designed advertisements found that ads in the Share Multiple condition generated more clicks per impression than the other conditions. Independent (and blind-to-condition) judges rated Share Multiple ads significantly higher and significantly more divergent. Participants in the Share Multiple condition shared significantly more ideas, moved more towards consensus, and reported a greater increase in rapport. In short, sharing multiple designs improves outcome, exploration, sharing, and group rapport. These results have significant implications for how designers and educators structure creative group work.

People around the world use my group's tools, and several HCI books incorporate our research. I have graduated three PhD students: Björn Hartmann, Brian Lee, and Ron Yeh. Björn is an Assistant Professor at UC Berkeley; Brian and Ron work in Silicon Valley. A fourth, Joel Brandt, is imminent; Joel is now a Research Scientist at Adobe.

TEACHING HUMAN-COMPUTER INTERACTION

The insights from this research have shaped my design teaching, which emphasizes creating and comparing diverse alternatives. My colleagues and I have created a Human-Computer Interaction curriculum that presents an exciting, interdisciplinary field synthesizing computer science, design, and the social sciences. In my six years at Stanford, I have created or significantly overhauled each of the three core HCI courses (cs147, cs247, and cs376), as well as teaching topics classes (such as Mobile Interaction). Sep Kamvar and I created a capstone class (cs294h) where students create web-scale software systems using rapid, data-driven iteration; this class provided students with firsthand experience using Web analytics in design, and revising a software system with real users. Last year, Phil Levis, Chris Manning, and I created a new PhD-level course designed to introduce PhD students to performing experimental computer science research. Many student research projects that began in my courses have subsequently presented their work as posters and (with follow-on work) several have published full papers at the top HCI venues. I currently advise forty undergraduate and masters students in Computer Science and Symbolic Systems. I led the creation of the HCI track for the Computer Science Bachelors degree, and its revision for the Computer Science Masters, Symbolic Systems Bachelors, and Symbolic Systems Masters degree. HCI is one of the most popular options for all four of these degrees. This fall, the 150 students in my introductory HCI class hail from sixteen majors.

Projects play two important pedagogical roles in my courses: creating a culture of prototyping and teaching designing for others. In interaction design, formal knowledge plays a critical but insufficient role. The "enlightened trial and error" of prototyping offers a means for understanding the exigencies of the current situation. When students first test their projects with users, and a design intended to be elegant turns out to be inscrutable, this provides a powerful learning moment. Doubly so when students figure out how to revise their design. The techniques, materials, and reading lists I have developed have been adopted by many other universities including Berkeley, CMU, Harvard, U Mass, MIT, UNC, Olin, Utah, Virginia Tech, Yale, the American University of Beirut, and FAST National University.

I believe strongly in undergraduate research. It provides experience for undergraduates, mentoring skills for graduate students, and important research contributions for the field. In my six years at Stanford, approximately 40 undergraduates have worked in my group. Many of my former research assistants from Stanford and Berkeley went on to pursue a PhD: three at MIT, three at Berkeley, two at UCSD, and also CMU, Maryland, UCLA, USC, and UW. Loren Yu, a CURIS intern, received nationwide Honorable Mention for CRA's 2008 Outstanding Undergraduate Award. Michael Bernstein, another CURIS intern, received a Stanford Firestone Medal for Outstanding Undergraduate Thesis. I joined our department's undergraduate research committee in 2004, participated in the strategic plan that outlined goals for increasing undergraduate research, and created a research-based senior project course. I help Terry Winograd organize the Stanford HCI Seminar, which provides a forum for students to hear leading researchers firsthand. We are working with SCPD to put the Seminar videos online; thousands have watched most of the posted videos.

REFERENCES

1 Brandt, J., M. Dontcheva, M. Weskamp, and S. R. Klemmer. Example-Centric Programming: Integrating Web Search into the Development Environment. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2010

2 Brandt, J., P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Opportunistic Programming: Writing Code to Prototype, Ideate, and Discover, *IEEE Software*, 2009

3 Brandt, J., P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2009

4 Carter, S., J. Mankoff, S. R. Klemmer, and T. Matthews. Exiting the cleanroom: on ecological validity and ubiquitous computing. *Human-Computer Interaction* **23**(1), 2008

5 Dow, S. P., J. Fortuna, D. Schwartz, B. Altringer, D. L. Schwartz, and S. R. Klemmer, Prototyping Dynamics: Sharing Multiple Designs Improves Exploration, Group Rapport, and Results, in *Submission*. 2010

6 Dow, S. P., A. Glassco, J. Kass, M. Schwarz, D. Schwartz, and S. R. Klemmer. Parallel Prototyping Leads to Better Design Results, More Divergence, and Increased Self-Efficacy. *ACM Transactions on Computer-Human Interaction* **17**(4), 2010

7 Dow, S. P., K. Heddleston, and S. R. Klemmer. The Efficacy of Prototyping Under Time Constraints. In Proceedings of *ACM Creativity & Cognition*, 2009

8 Everitt, K. M., S. R. Klemmer, R. Lee, and J. A. Landay. Two Worlds Apart: Bridging the Gap Between Physical and Virtual Media for Distributed Design Collaboration. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2003

9 Hartmann, B., L. Abdulla, M. Mittal, and S. R. Klemmer. Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2007

10 Hartmann, B., S. Doorley, and S. R. Klemmer. Hacking, Mashing, Gluing: Understanding Opportunistic Design, *EEE Pervasive Computing*, vol. 7(3), 2008

11 Hartmann, B., S. Follmer, A. Ricciardi, T. Cardenas, and S. R. Klemmer. d.note: Revising User Interfaces Through Change Tracking, Annotations, and Alternatives. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2010

12 Hartmann, B., S. R. Klemmer*, et al.* Reflective Physical Prototyping through Integrated Design, Test, and Analysis. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*, 2006

13 Hartmann, B., D. MacDougall, J. Brandt, and S. R. Klemmer. What Would Other Programmers Do? Suggesting Solutions to Error Messages. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2010

14 Hartmann, B., L. Wu, K. Collins, and S. R. Klemmer. Programming by a Sample: Rapidly Creating Web Applications with d.mix. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*, 2007

15 Hartmann, B., L. Yu, A. Allison, Y. Yang, and S. R. Klemmer. Design As Exploration: Creating Interface Alternatives through Parallel Authoring and Runtime Tuning. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*, 2008

16 Ju, W., B. Lee, and S. R. Klemmer. Range: Exploring Implicit Interaction through Electronic Whiteboard Design. In Proceedings of *CSCW: ACM Conference on Computer-Supported Cooperative Work*, 2008

17 Klemmer, S. R., K. M. Everitt, and J. A. Landay. Integrating Physical and Digital Interactions on Walls. *Human-Computer Interaction* **23**(2), 2008

18 Klemmer, S. R., B. Hartmann, and L. Takayama. How Bodies Matter: Five Themes for Interaction Design. In Proceedings of *DIS: ACM Conference on Designing Interactive Systems*, 2006

19 Klemmer, S. R. and J. A. Landay. Toolkit Support for Integrating Physical and Digital Interactions. *Human-Computer Interaction* **24**(3), 2009

20 Klemmer, S. R., J. Li, J. Lin, and J. A. Landay. Papier-Mache: Toolkit Support for Tangible Input. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2004

21 Klemmer, S. R., M. W. Newman, R. Farrell, M. Bilezikjian, and J. A. Landay. The Designers' Outpost: A Tangible Interface for Collaborative Web Site Design. In Proceedings of *UIST: ACM Symposium on User Interface Software and Technology*, 2001

22 Klemmer, S. R., A. K. Sinha, J. Chen, J. A. Landay, N. Aboobaker, and A. Wang. SUEDE: A Wizard of Oz Prototyping Tool for Speech User Interfaces. 2000

23 Klemmer, S. R., M. Thomsen, E. Phelps-Goodman, R. Lee, and J. A. Landay. Where Do Web Sites Come From? Capturing and Interacting with Design History. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2002

24 Kumar, R., J. O. Talton, S. Ahmad, and S. R. Klemmer. The Bricolage Structured-Prediction Algorithm for Example-Based Web Design. *In Submission*, 2010

25 Lee, B., S. Srivastava, R. Kumar, R. Brafman, and S. R. Klemmer. Designing with Interactive Example Galleries. In Proceedings of *CHI: ACM Conference on Human Factors in Computing Systems*, 2010

26 Li, Y., S. R. Klemmer, and J. A. Landay, Tools for Rapidly Prototyping Mobile Interactions, in *Handbook of Research on User Interface Design and Evaluation for Mobile Technology*, 2007

27 Maldonado, H., S. R. Klemmer, and R. D. Pea. When is Collaborating with Friends a Good Idea? In Proceedings of *CSCL: Computer Supported Collaborative Learning*, 2009

28 Maldonado, H., B. Lee, S. R. Klemmer, and R. D. Pea. Patterns of Collaboration in Design Courses: Team dynamics affect technology appropriation, artifact creation, and course performance. In Proceedings of *CSCL: Conference on Computer Supported Collaborative Learning*, 2007

29 Sinha, A. K., S. R. Klemmer, and J. A. Landay. Embarking on Spoken-Language NL Interface Design. *The International Journal of Speech Technology* **5**(2), 2002