IEEE pervasive COMPUTING
MOBILE AND UBIQUITOUS SYSTEMS
www.computer.org/pervasive

Hacking, Mashing, Gluing: Understanding Opportunistic Design

Björn Hartmann, Scott Doorley, and Scott R. Klemmer

Vol. 7, No. 3
July–September 2008

Hacking, Mashing, Gluing: Understanding Opportunistic Design

Learn about principles of opportunistic design through an interview study of 14 professional and hobbyist "mashers" from three design disciplines: Web 2.0, hardware, and ubiquitous computing.

Björn Hartmann, Scott Doorley, and Scott R. Klemmer
Stanford University

Opportunistic practices in interactive system design include copying and pasting source code from public online forums into your own scripts, taking apart consumer electronics and appropriating their components for design prototypes, and "Frankensteining" hardware and software artifacts by joining them with duct tape and glue code. We consider these opportunistic practices part of *mashup design*. Although many ubiquitous computing practitioners have engaged in these practices, design tools and software engineering research don't traditionally address them.

Mashup design's ad hoc nature might be antithetical to classical software engineering methods, but it can have a significant impact. For example, Eric von Hippel chronicles the importance of end-user innovation in fueling commercial product development in this issue (p. 66) and elsewhere.[1] Because hobbyists and amateurs often undertake opportunistic design, it relates to end-user programming.[2,3] Even professionals engage in opportunistic practice when speed and ease of development are valued over robustness and maintainability.[4] We aimed to understand how mashup design of software and hardware takes place today to derive goals for better design tools in the future.

In this article, we introduce a framework that situates opportunistic design for ubiquitous computing at the intersection of three existing hacking traditions and distinguishes between deep and surface-level approaches for integrating components. We interviewed 14 professional and amateur "mashers" from three design disciplines: Web 2.0, hardware, and interactive ubiquitous computing. This interview study revealed how designers choose between integration levels; how mashups provide epistemic, pragmatic, and intrinsic values for their creators; and how shopping becomes a central activity.

Ubicomp mashups

In our view, *mashups* consist of recombination and ad hoc design across boundaries of bits and atoms. This broad perspective builds on previous concepts of mashups in computer science and music. Mashups originated in music, where the term denotes the practice of taking elements of two or more existing songs and creating a new piece by rearranging, interspersing, and superimposing parts of these sources. Computer science later adopted the term to refer to applications created by programming against one or more public Web APIs, also known as infrastructure services.[5] We're most interested in the nascent area of *ubiquitous computing mashups*. Ubicomp mashups attempt to move computation off the desktop and integrate it with the artifacts of everyday life.[6] They extend beyond the Web and combine the functionality of both software and hardware components.

Published by the IEEE CS ■ 1536-1268/08/$25.00 © 2008 IEEE

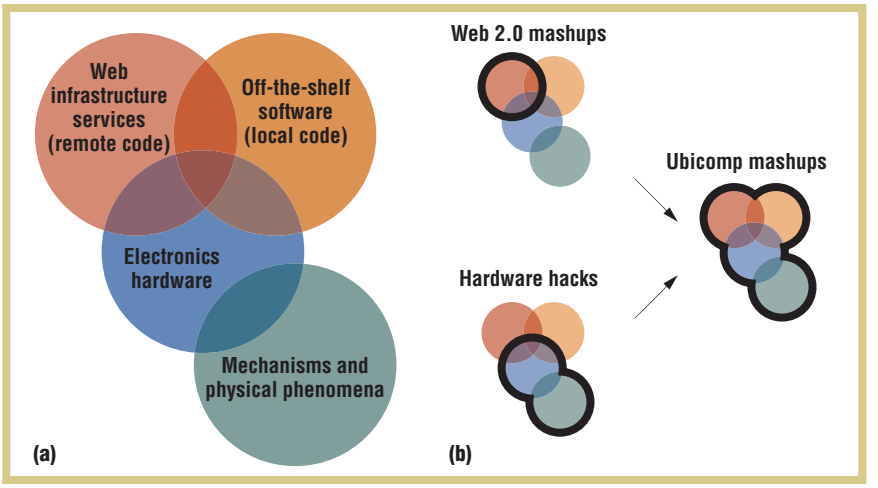


Figure 1. Ubicomp systems ingredients. (a) Four components of a ubicomp mashup. (b) Ubicomp mashups unite hardware and Web practices.

A framework of mashup components

Moving from the physical to the digital domain, a ubicomp mashup can use four types of components (see Figure 1). First, a mashup can contain built or repurposed *mechanisms*, such as a toy doll's movement mechanism. Second, sensors and actuators can interface with these mechanisms and other physical phenomena; *electronics* such as embedded programmable microcontrollers provide the logic for sensors and actuators. Third, designers can write their own programs or leverage *off-the-shelf software* on their personal computers (be it a desktop, PDA, or smart phone). Local applications might offer hooks for programmatic automation through APIs or built-in scripting languages. Fourth, mashups can use *Web infrastructure services* such as search and mapping APIs.

Each of these four components has a history of opportunistic design practice (see Figure 2). Shell scripts and application macros have long functioned as glue between desktop applications. John Ousterhout provides a good overview of scripting languages' advantages for connecting preexisting software components.[7] Bonnie Nardi's account of end-user programming describes tool-independent practices such as *programming by example modification*.[8] In the tangible world of mechanisms and electronics, amateurs as well as professional product designers cannibalize or repurpose off-the-shelf products to fit new needs. Hardware hacking has seen a recent resurgence in popularity with hobbyists, evidenced by the success of publications such as *Make* magazine (www.makezine.com). The advent of open APIs for Web services has spurred development of numerous services and sites that aggregate disparate data sets. The Web API catalog programmableweb.com lists 3,109 Web mashups leveraging 775 distinct APIs as of June 2008.

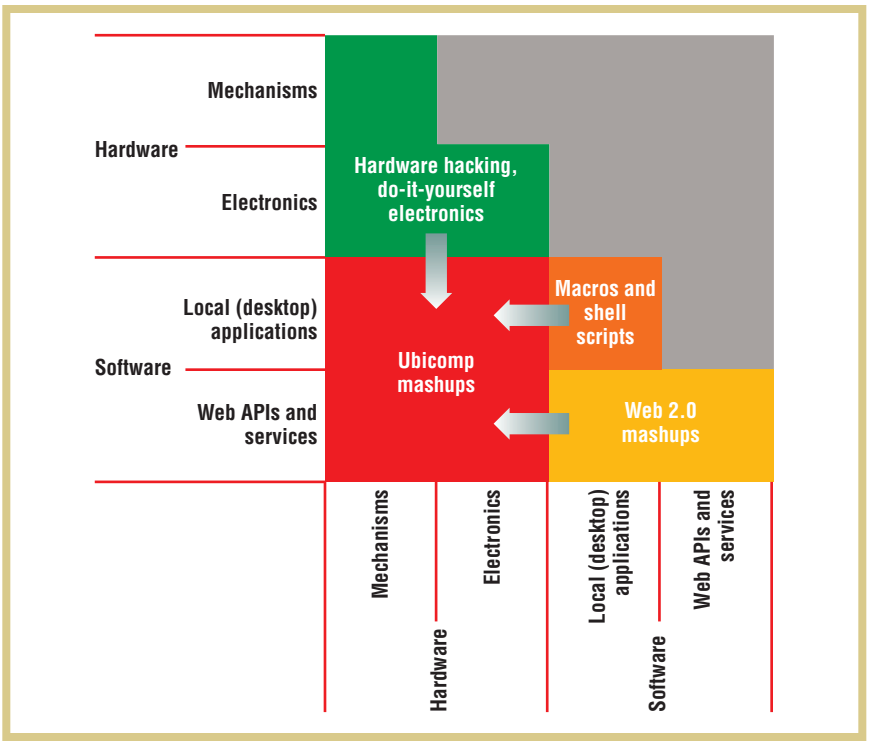


Figure 2. A classification of mashups based on their components. The arrows indicate how existing communities and practices inform ubicomp mashups.

Integration strategies: Dovetail joints versus hot glue

A broad shift that the mashup paradigm introduced is the reallocation of the designer's effort and creativity. More time and ingenuity go to selecting components and shaping the "glueware" that interfaces them.

We distinguish between two approaches to glue. In the first, two components explicitly support combination through a shared interface. They're aware of each other, allowing for tight integration. We use the carpenter's *dovetail joint* metaphor to label these deep combinations. Dovetail joints are

Figure 3. Participants' mashups. Samples include applications for (a) planning an evening out, (b) plotting weather forecasts on a map, and (c) finding train schedules. Participants (d) created a combination toy and flashlight and (e) a flying toy car, (f) listened to audio in noisy environments, (g) developed an application for annotating printed documents with video, (h) developed an indoor positioning prototype for smart shopping carts, and (i) built audio art installations.

documented extension and integration points in the system architecture—APIs in software, breakout headers and connectors in electronics, and mounting holes in hardware.

In contrast, *hot glue* combinations adjoin components that are either incompatible, don't know about each other, or don't support each other. You can apply hot glue to almost anything, but it has limited adhesive power—all it can offer is shallow, surface-level integration. *Screen scraping*—parsing rendered user interfaces such as Web pages to gather data—and *screen poking*—generating synthetic mouse and keyboard events computationally—are examples of digital hot glue joints. Importantly, a designer's intent is often hidden in such glue code: what is recorded is only a trace of the taken actions (for example, a sequence of mouse clicks), but not their semantics (such as opening a particular file).

In practice, most systems, whether software or hardware, are constructed from preexisting components—code libraries, integrated circuits, and mechanical subassemblies. This raises the question of whether there's a dividing line between component-based engineering and mashup practice.

One distinguishing characteristic might be the degree to which systems rely on dovetail and hot glue joints to operate. Where engineering methods strive to cleanly integrate dovetails, mashups often use both dovetail and hot glue connections simultaneously. In mashup design, component selection is informed, but not dictated, by the availability of a suitable interface. If a clean integration interface is available, the practitioner will use it; if not, the practitioner will resort to more brittle workarounds.

Furthermore, because component vendors don't sanction hot glue joints and appropriations, the source of authoritative information and support shifts away from vendors and manufacturers and toward the community of mashup designers.

We were curious to what extent integration practices are shared by mashup

designers across hardware and software domains. We also wanted to know to what extent current domain-specific tools are appropriate to support ubicomp mashups. We approached these questions through an exploratory interview study.

Interview methodology

We interviewed 14 practitioners from three areas of mashup design. Four participants were involved in Web 2.0 development. Four others focused on hardware hacking—working with mechanisms and embedded electronics. Six participants worked as ubicomp designers—creators of interactive computing systems spanning hardware and software components. In our interviews, we asked participants to describe their work philosophy and general approach to problem solving, and then to focus on one particular recent project. To ground and structure the discussion, we asked participants to produce artifacts or visual representations (photographs or sketches) of their project. Specifically, we asked participants to describe third-party components they integrated, how they decided to include particular parts, and the trade-offs and challenges they experienced.

Sampling mashups: Who, what, why

Here we review the material collected: who our participants are, what kinds of systems they build, and how and why they build them. For brevity, we only mention a subset of the interviewees and focus on commonalities within groups.

Web 2.0 programmers

Our participants were professional programmers or Web developers who didn't feel that mashup programming's technical aspects were a hurdle.

Our first participant, W1, owns a cellphone software company. In his spare time, he developed a mashup Web site that overlays restaurant and bar information on an interactive map (see Figure 3a). Users build a graphical path from one venue to the next to plan an evening out with friends. They can also send these paths to a compatible mobile phone. This mashup combines three online services: CitySearch for entertainment reviews, Google Maps for mapping and navigation on the desktop, and Yahoo! Maps for mapping on mobile devices.

A second mashup, written by participant W2, also builds on Google maps. His Web site features georeferenced weather forecasts and temperature readings, integrated displays of user-contributed webcam feeds, and weather histories. His application aggregates forecasts from more than a dozen national and regional weather data providers and locates these forecasts on a map (see Figure 3b). The site is generating enough traffic—and ad revenue—that he is contemplating making this side project his full-time job.

Aiming at the emerging mobile application market, participants W3 and W4 built a mashup that delivers relevant train schedules for three US commuter rail systems to mobile phones through SMS or email (see Figure 3c). Users send a short message with a station name abbreviation to their system, which replies with upcoming train times. The system links an SMS email gateway to a schedule database gathered from the individual rail companies.

Screen scraping vs. Web APIs. One major concern for our Web 2.0 participants is access to and strategies for getting data: "Getting the data is the absolute hardest part" (W3). The surveyed mashups derived their value from integrating disparate data sets in ways not previously possible. Although two of the three projects used Google Maps' open, documented infrastructure service, all three projects resorted to screen scraping (parsing) to gather at least part of their data. Participants gave two primary reasons for scraping:

- APIs simply weren't available for obtaining the desired data, and
- Web APIs are generally designed for smaller data requests, so it's still easier to obtain large data sets by scraping.

W2 reported building his own scraping toolkit so it now takes him as much time to develop a scraper as it would to integrate an available API.

Business models and obstacles. All participants reported that their mashups started as side projects to their full-time jobs as consultants, business owners, and developers. However, two of the three projects expressed interest in turning the mashup into a profitable business. With Web mashups, shifting from the personal sphere to the commercial sphere can be challenging for both legal and technical reasons. W1 reported that making money by using scraped content is problematic because of licensing restrictions. W2 reported that he had to add redundant data sources because individual weather providers could alter the format or withdraw their data streams at any time.

Hardware hackers

In the physical and electronic design realms, we interviewed three toy inventors at two design companies and a hobbyist who refashions consumer goods into personalized tools and publishes instructions for creating these tools online. The toy inventors build prototypes that illustrate new interaction design concepts. They don't create finished products. Project schedules are very short, ranging from two days to less than a month.

When we visited participant H1, she was working on a toy that functioned as a flashlight with sound effects. To make the concept tangible, she bought a pair of plastic monkeys from a local toy store because the monkeys had a similar opening mechanism to the one she envisioned (see Figure 3d). She then embedded a tactile switch into the mechanism's lever to trigger light and sound effects using external electronics. A previous

project prototype combined a toy car body with plastic rocket engines from a model plane kit to create a new flying car (see Figure 3e). To her, the aesthetics (the "toyness") of the repurposed packaging mattered, even though the final product would have a radically different look.

At the second toy company, participants H2 and H3 described how they prototyped a handheld wireless controller for a TV game. They took the controller's barrel from a soda bottle, and they built the grip from a Gyration wireless mouse that uses a gyroscope to sense tilt, transforming that tilt data into cursor movement. A custom-made plastic mold joined the two pieces into one unit using custom-made plastic molds. They then used the wireless mouse's cursor and click events to animate graphics on a laptop (used as a stand-in for a television set) running Adobe Flash.

In contrast to the toy designers' rough-and-ready prototypes, participant H4 builds his hardware-based mashups for long-term private use. Many of the artifacts he uses daily were created by modifying consumer goods. One project he created was a pair of jackhammer hearing-protection earmuffs that he retrofitted with a pair of airline headphones to listen to audio books in noisy environments (see Figure 3f). According to H4, this design offers better noise reduction than commercial noise-canceling headphones and is significantly cheaper.

For all three toy inventors, visiting large retail stores to purchase interesting new toys was an integral part of their core practice. They would later disassemble these toys in their shop. We identified three strategies of appropriating store-bought toys:

- Designers extract mechanisms and reuse them in different skins (for example, H2 and H3 transferred a purchased toy's animated movement into a new prototype).
- Designers keep a toy's shell but embed new electronics into it (H1 did this "because it immediately looks like a toy").
- Designers fuse different shells (such as H1's metal toy car with air plane rocket engines) to produce a composite object.

While many Web mashups build on a few high-value components, such as Google Maps, our hardware hackers' choices didn't cluster around high-value products. To the contrary, within a given genre the toy designers collected a wide variety of products in their storage bins for later reuse.

In contrast to the toy designers, H4 saw the tailoring of existing artifacts as a partial rejection of consumer culture. The self-sufficiency of "do it yourself" offers a degree of intrinsic satisfaction along with a level of personalization and novelty unavailable in mass-produced artifacts. For H4, the economies of scale that mass-produced consumer goods leverage are incentives. Picking existing parts is cheap: "It's never cheaper to start from scratch to make your own."

Ubicomp designers

Our six ubicomp developers used mashups as prototypes and proof-of-concept deliverables, but also as a way to design and implement site-specific tools for a single user or a small community.

Participant U1, a design researcher, worked on a system for design teams to annotate printed documents with short video messages. In his functional prototype (see Figure 3g), users push a button to initiate video message recording on a laptop. After recording, the system prints a small label displaying a snapshot of the video and a bar code. The user attaches this bar code to the document described in the video. If another user wants to access the video, she waves the bar code in front of the same camera, upon which the system retrieves and plays back the desired video. U1 relied heavily on commercial off-the-shelf software, combining five different applications through AppleScript. For example, he scripted QuickTime to record and play back video, and he used the Excel spreadsheet software as a database. To convey this project's complexity, Figure 4 shows our redrawn version of his system architecture sketch.

Participant U2, an industrial researcher, described a project where he designed an indoor positioning prototype for smart shopping carts. This positioning system employed computer vision. To test the vision data quality, U2 attached a custom-built optical rotation sensor to a shopping cart's wheel and soldered its contacts to the left button of a gutted PC mouse, so that each revolution yielded one click (see Figure 3h). By counting the total number of clicks on the PC, he received ground truth data about the total distance the cart had traveled. (For more information, also see "Hacking in Industrial Research and Development" in this issue.)

U3 has been developing his own musical programming language and graphical environment for producing and performing electronic music. He builds audio installations that he shows at the annual Burning Man festival (see Figure 3i). Although he spent years designing his software from the ground up, the physical controllers he used were off-the-shelf game console input devices such as "Dance Pad" floor mats. According to him, "you can choose what level of effort you want to put in—you can buy the next level of integration." To him, a key component enabling his installation was a small hardware converter that lets him connect controllers built for proprietary game consoles to a PC USB port.

As Web 2.0 programmers employ screen scraping to harvest information from online databases, ubicomp programmers use screen poking to remotely control software. In addition to U2's appropriation of a mouse button for measuring turns of a wheel, U1 initially used the macro software Automate as a means to control desktop applications by computationally injecting synthetic mouse and keyboard events. U3 purchased a hardware converter that transformed the output of pressure-

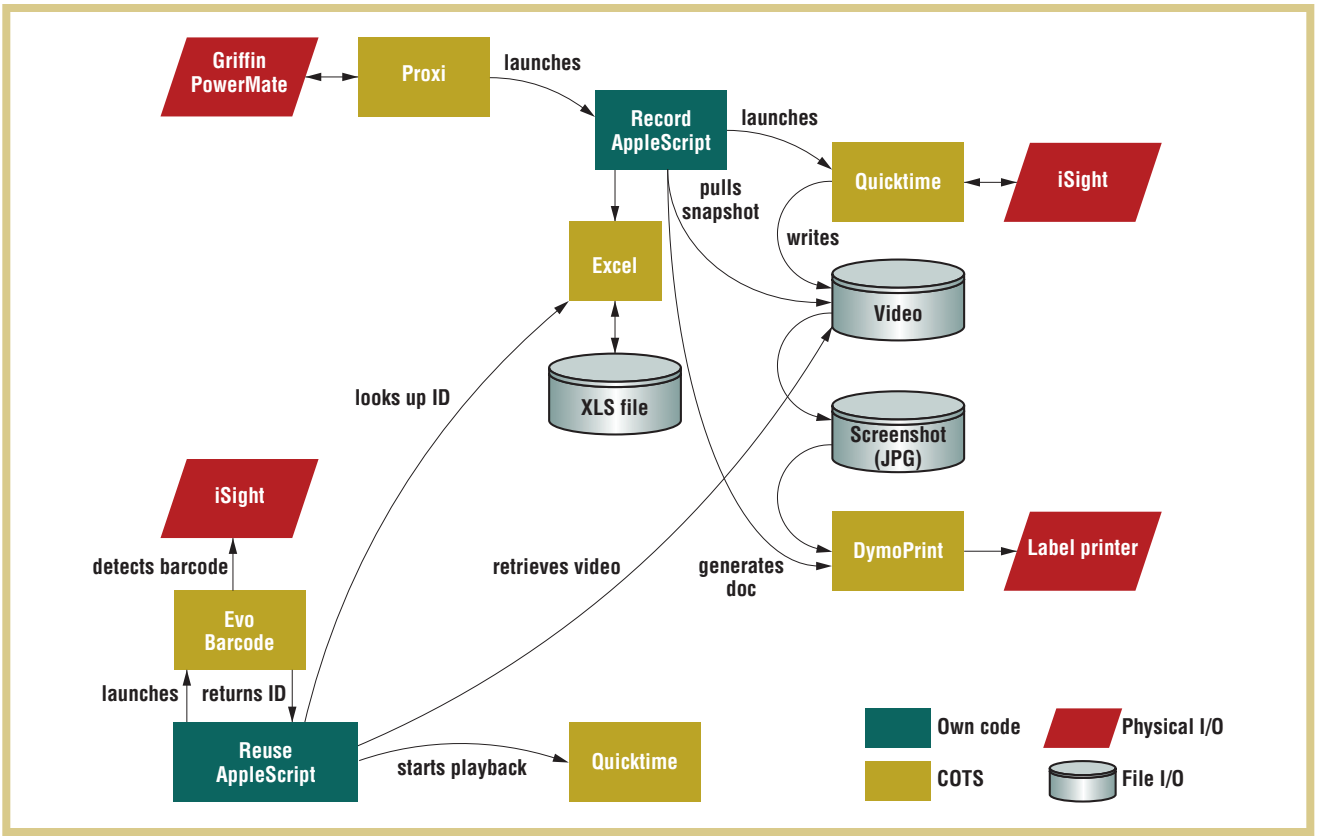


Figure 4. System diagram of U1's project. The project enabled designers to annotate printed documents with video messages.

sensing dance pads into Windows platform game controller events. U3 chose these glueware techniques for similar reasons as screen scraping: APIs are sometime unavailable, don't yield the desired information, or are more time-consuming than surface-level instrumentation.

Screen scraping can also be interpreted as an act of sensing, while screen poking in turn is analogous to actuation. As sensing the physical world yields ambiguous, noisy data that must be conditioned and filtered, data from screen scraping often has to be cleaned and processed. This suggests that mediation techniques for ambiguous sensor input[9] might transfer to Web scraping, and vice versa. Despite the analogies, there are barriers in crossing the chasm between Web-centric applications and the physical realms of sensing and actuation. One reason is that client-side Web technologies have increasingly moved into secure-execution sandboxes that can't communicate directly with external hardware. We still need design tool support for bridging these two domains to enable experimentation by lead users.

Themes in opportunistic programming

Our interviews uncovered some common concerns across the three design domains. Choosing between levels of integration, shopping, and connecting to larger communities of mashup designers emerged as unifying themes, among others.

Dovetail joints versus hot glue revisited

Across domains, our interviewees freely mixed deep and surface-level integration techniques in their projects. Each choice has important limitations: while shallow hot glue is brittle, deeper integration might have limited reach. These trade-offs are exemplified by U1's experience. He scripted an earlier version of his document annotation system using software that lets users record interaction with GUI widgets and replay those actions programmatically. Although this system succeeded as an experience prototype, it wasn't robust enough for any unsupervised deployment. Seeking to improve on stability, U1 then switched to AppleScript, which let him leverage application-specific APIs. Although the deeper glue that AppleScript provides is significantly cleaner for expressing logic than GUI events, U1 found no programmatic means within AppleScript for uploading the video clips to an online media-sharing site, a task that his previous strategy could accomplish.

Beyond the technical consideration of how to adjoin components is a larger question about the relationship between the designed intent of the constitutive elements and that of

the resulting mashup. Mashups might appropriate technologies, repurposing them as building blocks toward a goal at odds with their original design.

One suitable definition of appropriation is "the extent to which a violation of a technology's intended purpose occurs."[10] This violation is easy to see in toy hacking: toys were intended for children to play with, not for designers to take apart. Similarly, in the digital realm, screen scraping appropriates output intended for human consumption as program input. In contrast, using Web 2.0 APIs such as Google Maps isn't an act of appropriation because the API's providers give explicit permission to use the service in new contexts.

It's notable that in the Web 2.0 space, where the general trend has been to open up infrastructure services to allow reuse without appropriation, all of our participants still resorted to screen scraping techniques. There are valid business reasons not to make all company data available for automatic processing by others through APIs. Simultaneously, those same business reasons make capturing the data valuable for third parties. We conclude that support for both tight and loose coupling (dovetail joints and hot glue) will be inevitable for design tools. Opportunistic design is based on integrating existing artifacts that best fulfill a functional or informational need, regardless of their programming interface or licensing agreement.

Mashing as a design activity

Next, we consider the activity of creating mashups: when, how, and why is mashing preferable to other design and development approaches? What value do practitioners derive from it?

Short timelines, small audiences? Mashup design in the physical world tends to happen on short timelines—the mashups we encountered were built quickly, and many were discarded just as quickly afterwards. By necessity, the artifacts were intended for small audiences; physical mashups are one-offs that can't be duplicated easily. The emphasis on speed is a good match for designers who want to rapidly prototype multiple ideas, consultants operating on compressed project schedules, and hobbyists with limited leisure time. Similarly, for these constituencies, the audience of a user's mashup is small: the design team, a single client, or oneself.

The Web mashups we encountered have different traits: they operate continuously, and their success is measured in the number of users they attract. Thus, engineering for robustness, redundancy, and maintenance becomes important—in this respect, building Web mashups more closely resembles traditional software engineering. This difference could be an artifact of our small survey population, but Web applications offer the unique opportunity to reach larger audiences without reengineering from the ground up: the prototype *is* the product. This opportunity to scale could lead Web developers to contemplate robustness from the outset.

Although it's certainly fast to get applications up and running by appropriating existing technology, completing the "last mile"—fine-tuning application logic and interaction design—can be difficult as desired functionality and offered features of existing components diverge. On the other hand, building with lower-level blocks, or even from scratch, incurs a large initial cost because developers must write their own tooling. In exchange, they preserve flexibility and can leverage their own tools later in the project cycle. The sweet spot for rapid, disposable mashups that our interviews found is consistent with this analysis. It also suggests an opportunity for design tools that leverage opportunistic development early on while preserving flexibility or offering some level of guaranteed robustness.

Epistemic, pragmatic, and intrinsic values. We found that mashups provided both pragmatic and epistemic value to our participants. An artifact is pragmatic to the extent that it enables actual use, and it's epistemic to the extent that it serves as a locus of communication with other stakeholders—clients, team members, and users—and provides information that can drive future design.[11,12] For some participants, creating mashups also held intrinsic value generated by the activity itself, rather than the utilitarian or educational value of the outcome.

Pragmatic decisions for mashups are made if using mashups is more efficient or effective than other techniques to reach a goal. Participant U3 estimated that by repurposing a mouse button to fire a click event with each revolution of a wheel, he was able to complete the sensing part of his project in a quarter of the expected time. Furthermore, incorporating existing pieces lets designers leverage functionality that they couldn't build themselves. Framed this way, we can think of the set of existing technologies in the world as a vast library that we can use to lower the threshold for development. For example, U4 didn't have sufficient technical knowledge to build his own physical music controller, but, through adapters, he was able to leverage commercially available game controllers.

Other times, practitioners employ mashup design as a means of exploration, learning, or inspiration. This epistemic activity was most prevalent among our toy inventors, who chose mashups as effective means to illustrate new concepts. What their clients paid for was the idea, prototyped through the mashup, not the implementation. Furthermore, rapidly creating prototypes gives designers concrete artifacts they can expand on, react against, modify, and transform. This conversation with materials (as opposed to thinking in the abstract) is an important strategy of reflective practice.[13] Reflective practitioners are concerned with problem setting as much as problem solving, and they let prototypes inform their understanding of the larger design space.

In the intrinsic case, practitioners create mashups because they regarded the activity of mashing as fulfilling

in its own right. They derive intrinsic value from the joy of exercising a craft ("what a great way to spend an afternoon") or from a personal ideology ("recycling is my form of protest against consumer culture"). Our interviews suggest that intrinsic activity is most common among hobbyists.

Shopping for functionality. As Frederick Brooks wrote, "The most radical possible solution for constructing software is not to construct it at all."[14]

How exactly does the activity of designing and developing change when no "new" software is created? Participants reported spending significant time on finding and acquiring their ingredients. In fact, some reported that this was the most challenging or time-consuming part of their process. U1 described the processes of searching for components and determining how to integrate them into his design as "the main part of the whole thing." Or, as U3 put it, "The real challenge is finding the interface between the problem and commercially available stuff."

Our toy inventors also reported frequent trips to the toy store without having a shopping list for a project. U4 did the same at electronics retail stores. We found three reasons for shopping without a project in mind:

- It builds awareness of the state of the art and shows designers what's commercially available.
- It reduces the cost of future searches. Like squirrels gathering nuts before the winter, designers stockpiled mechanisms to have them ready later. H2 said, "We collect [mechanical] movements. … [During a project, one of us will say] 'Remember that freaky belly movement?'"
- It inspires new projects. "I go on shopping trips and think about repurposing objects. ... I'll walk around Walgreens and look at objects and think, 'What could this be?'" (H1).

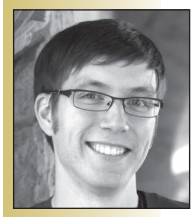
Searching for and acquiring pieces was inspirational and helped steer projects in a particular direction. This suggests that shopping itself can take on an epistemic function.

Searching for bridges. Several times, participants reported finding crucial connecting pieces for their mashups in fields only tangentially related to their own. U4 discovered that a MIDI-to-relay interface used by church-organ builders would trigger lights based on music commands for his Burning Man installations. Adapters and bridges are well-known design patterns for software engineers. We focus on the social side—the bridges that led practitioners to discover these connections in the first place. While Web search was universally used, effective search requires prior knowledge of the space of opportunity. Community sources play an important role: for example, U1 integrated two external button interfaces into his project because he knew that other researchers in his building had used those particular models successfully. Scaling such community awareness to geographically distributed teams of designers is an important goal for the future. In the hobbyist market, Web sites like http://instructables.com that publish instructions and parts lists for do-it-yourself projects have begun to address this need.

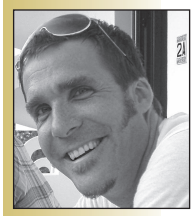
Our analysis raises several suggestions for creating future mashup design tools. First, it's important to recognize mashup programmers and hardware hackers as a unique target audience: they're not professionals, in that their primary job description isn't creating mashups, but neither are they untrained end users. Our participants were all technologically sophisticated and used mashup techniques to achieve some other goal in their domain of expertise. So, design tools must strike a balance between complexity and flexibility.

Second, the use of both dovetail as well as hot-glue combinations in many of the projects suggests that we need tools that better support fluidly transitioning between the two integration styles within the same project.

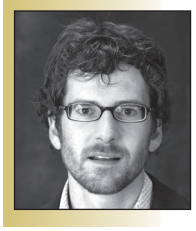
the AUTHORS



Björn Hartmann is a PhD candidate in HCI at Stanford University. His research focuses on prototyping tools for designers and lead users. Hartmann received his MSE in computer and information science from the University of Pennsylvania. Contact him at bjoern@cs.stanford.edu.



Scott Doorley is the director of the environments lab at Stanford University's Hasso Plattner Institute of Design. His research interest is applying design methods to creative domains such as writing, film making, and informal learning. Doorley received his MA in learning, design, and technology from Stanford University. Contact him at sdoorley@stanford.edu.



Scott R. Klemmer is an assistant professor of computer science at Stanford University, where he codirects the Human-Computer Interaction Group. His primary research focuses are interaction techniques and design tools that enable integrated interactions with physical and digital artifacts and environments. Klemmer received his PhD in computer science from the University of California, Berkeley. Contact him at srk@cs.stanford.edu.

Third, we can learn from product designers who keep their studios stocked with cannibalized parts by developing tools that more fully embrace "design by example modification" or "design by example augmentation" as a fundamental strategy.

Finally, design tool research often focuses on the construction of applications. The important epistemic and pragmatic functions of shopping suggest that tools that support search, selection, and sharing of existing components could be equally valuable.

REFERENCES

1. E. von Hippel, *Democratizing Innovation*, MIT Press, 2005.
2. A. Cypher, *Watch What I Do: Programming by Demonstration*, MIT Press, 1993.
3. H. Lieberman, F. Paterno, and V. Wulf, *End-User Development*, Springer, 2005.
4. J. Brandt et al., "Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice," *Workshop End-User Software Eng.*, ACM Press, 2008, pp. 1–5.
5. E.A. Brewer, "Lessons from Giant-Scale Services," *IEEE Internet Computing*, vol. 5, no. 4, 2001, pp. 46–55.
6. M. Weiser and J.S. Brown, "The Coming Age of Calm Technology," *Beyond Calculation: The Next 50 Years of Computing*, P.J. Denning and R.M. Metcalfe, eds., Copernicus Books, 1997, pp. 75–86.
7. J.K. Ousterhout, "Scripting: Higher Level Programming for the 21st Century," *Computer*, Mar. 1998, pp. 23–30.
8. B.A. Nardi, *A Small Matter of Programming: Perspectives on End User Computing*, MIT Press, 1993.
9. J. Mankoff, G.D. Abowd, and S.E. Hudson, "OOPS: A Toolkit Supporting Mediation Techniques for Resolving Ambiguity in Recognition-Based Interfaces," *Computers and Graphics*, vol. 24, no. 6, 2000, pp. 819–834.
10. R. Eglash, "Appropriating Technology: An Introduction," *Appropriating Technology: Vernacular Science and Social Power*, R. Eglash et al., eds., Univ. of Minnesota Press, 2004, pp. vii–xxi.
11. D. Kirsh and P. Maglio, "On Distinguishing Epistemic from Pragmatic Action," *Cognitive Science*, vol. 18, 1994, pp. 513–549.
12. S.R. Klemmer, B. Hartmann, and L. Takayama, "How Bodies Matter: Five Themes for Interaction Design," *Proc. 6th Conf. Designing Interactive Systems*, ACM Press, 2006, pp. 140–149.
13. D.A. Schön, and J. Bennett, "Reflective Conversation with Materials," *Bringing Design to Software*, T. Winograd, ed., ACM Press, 1996.
14. F.P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1995.