Web Mash-ups and Patchwork Prototyping: User-driven technological innovation with Web 2.0 and Open Source Software

Ingbert R. Floyd, M. Cameron Jones, Dinesh Rathi, Michael B. Twidale
University of Illinois at Urbana-Champaign
Graduate School of Library and Information Science
{ifloyd2,mjones2,drathi,twidale}@uiuc.edu

Abstract

The recent emergence of web mash-ups and open source software is driving the development of new practices in software and systems development. In this paper we explore novel practices of user-driven innovation through an examination of several case studies which illustrate how users and developers are exploiting the proliferation of open APIs and open source systems. Developers can rapidly create proofs of concept that are robust enough for actual use by combining preexisting software components. The underlying programming processes involved make use of tried-and-true software development techniques, and may not appear innovative at first. However, the application of these practices and techniques to problem solving by non-programmers shows a high degree of creative innovation, giving rise to new ways of thinking about technology design and production.

1. Introduction

Web 2.0 and open-source software are but two of the recent trends in software development characterized both by new technologies and by new mindsets on how to do application development. These trends are capturing the imagination of practitioners and academics alike, stimulating creativity, innovation, and a flurry of attempts to anticipate how the landscape will stabilize e.g., [21], [24]. The focus, however, is often on technology or on the new business models that are emerging.

It is easy to forget that most people are interested in technology primarily for how it can help them in their everyday life activities. One of the authors recently taught an Introduction to Web Technologies class geared for non-programmers, where the final project involved creating a prototype of a web mash-up. Invariably, the students described their projects in terms of an immediate, pressing problem in their everyday life which they were creating a web mash-up to solve: for example, they are new to campus and want to know where to eat, so they built a web mash-up to map restaurants using Google Maps. These everyday life activities can involve aspects or combinations of work, personal life, school, etc. It is interesting is how similar patterns of technological innovation, appropriation, and use are emerging in practice by people involved in seemingly very different types of activities: design environments, community building, and classroom learning.

In this paper we take a step back and look at two kinds of user-driven, emergent practices: web mash-ups and a design technique we call patchwork prototyping. Our purpose is to understand how the affordances of recent trends are enabling these two practices, why they suddenly are so prominent, and how they capture the creativity, needs and desires of the users who are driving the approaches. We intend our analysis to provide insights which can be integrated and merged with other rapid, collaborative and participatory mechanisms to support innovative explorations of design spaces, requirements capture, and methods for rapid prototyping and evaluation.

2. Web mash-ups

The original vision of the web was of a system for academics to share information and data in the form of documents [1]. The parallel development of concepts like the semantic web [2], web services, Web 2.0 [19], and the architecture of participation, has resulted in a multitude of new services, web sites, technologies, and protocols. Similar to earlier practices of software reuse, these approaches involve sharing and distribution. However, sharing need not be just of documents but also of services, knowledge, resources, and objects. Distribution has also broadened, not just providing access to humans, but also to applications.

Web mash-ups, websites which combine data and services from across the web, are an emerging trend. The concept of mash-ups originated in the DJ music culture, where the recent development of inexpensive,

professional-grade, music composition and mixing software allowed musicians to create high-quality remixes and to easily sample and recombine digital music [10]. A music mash-up is a remix of music from multiple sources. Similarly, a web mash-up combines data and services from more than one source.

Weiss identifies an intriguing characteristic of web 2.0 applications that is shared by web mash-ups: that they are "… at the same time incredibly innovative and yet—not" [25]. That is, from a computer science perspective, the underlying technology and practices are not really innovative; software developers have been sharing, reusing, and combining applications and code for decades, using code libraries, components and APIs to speed up development, e.g. [11]. What is innovative is how mash-ups are being widely used for the rapid realization of creative ideas which would be too time consuming, or expensive.

Through the use of publicly available APIs (Application Programming Interfaces), mash-up developers are able to access data, services, resources, and interface components, which they incorporate into their new application. There are three aspects of web 2.0 APIs which facilitate innovation with mash-ups:

1. They provide access to highly developed, robust technologies which only a large organization of expert programmers could create;
2. They provide access to massive amounts of content which no individual could gather on their own or afford to keep and maintain;
3. They lower the barriers to developing creative novel applications with powerful technologies.

Amazon.com was one of the first commercial sites to release a free, public API for accessing their content. Coupled with the API documentation were code libraries and examples written in several programming languages. Many applications were written to interface with the Amazon database (e.g., Delicious Library) but not explicitly called "web mash-ups". It wasn't until after the release of the Google Maps system and the development of housingmaps.com site in summer 2005, that the term mash-ups was used to characterize websites.

The website programmableweb.com lists 221 different APIs which can be mashed-up. The available APIs span a wide range of applications, including: search engines, mapping applications, instant messaging, weather data, blogs, RSS aggregators, image and video sharing, social networking, personal and/or team information management systems, social bookmarking, wikis, and auction sites. Over 900 mash-ups have been registered at programmableweb.com at an average rate of three new mash-ups registered every day. Not all mash-ups which have been created are registered at programmableweb.com. Some estimate that as many as 1,000 new applications are developed every six months based on the Google Maps API alone [8].

The rapid explosion of mash-up development activities must have some cause. We have noticed that web mash-ups are often created by individuals or small groups motivated by a particular problem who are inspired to use the new Web 2.0 technologies and mindsets [19] to create a solution. The principle that "every good work of software starts by scratching a developer's personal itch" [21] originally used to describe the success of the open-source software (OSS) process also seems appropriate in characterizing mash-up development, except that the technologies of mash-ups are accessible to both skilled and non-skilled programmers, and the process is faster than in typical OSS development.

One of the earliest web mash-ups was housingmaps.com (Figure 1), created when its developer, Paul Rademacher, was looking for a new house. In examining the daily updated real-estate listings on Craigslist he was confused by which houses he had already seen. One day, he found himself looking at a house he had just visited the previous day [20], and decided he needed to do something about it. Organizing the listings geographically integrated the data around a common interface, which helped him remember where he had already looked.

Given that such an application is only useful to a person while they are actively searching for a house, developing it as a single user without utilizing web-based APIs would have taken too long and been too complicated to be of any practical value. He probably would have found a house before a working system could be finished. However, the mash-up approach drastically reduced the development costs, making the task of developing such an application feasible.

This development is analogous to the changes

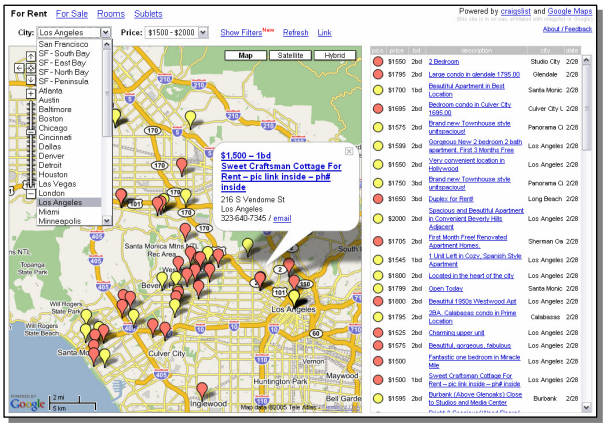

Figure 1. Housingmaps.com shows real-estate listings from Craigslist in Google Maps

which arose out of the introduction of spreadsheets in early PCs in the 1980s. Before the advent of the spreadsheet, numerical computing required the expertise of both programmers and mathematicians. Applications were custom built to address particular problems and took months to implement, and often did not satisfy all of the requirements [8]. Spreadsheets revolutionized numeric computing in organizations by providing a reusable framework for rapidly testing and developing numeric applications. Users were able to create and share a wide range of (but not all) mathematical applications such as payrolls, budgets, and numerical models, quickly and easily [17].

It is true that creating mash-ups does require detailed knowledge of how particular APIs are structured and a solid foundation in web technologies and protocols. Currently, this may restrict development to experienced programmers. However, the key to lowering the barrier to mash-up development probably lies in the development of toolkits, wizards, and other systems which can black-box much of the esoteric details of the implementation, or provide an end-user interface to facilitate creating mash-up code. This can already be seen in sites like mapbuilder.net and wayfaring.com which provide simple to use web interfaces for creating Google Maps mash-ups. While the development of such programming aids will broaden the accessibility of mash-up programming, they will necessarily be unable to provide users with full-access to the complete flexibility of a programming language, much as spreadsheets can only support a subset of all mathematical applications.

3. Patchwork prototyping

We use the term patchwork prototypes to describe applications developed using a different design process than web mash-ups. Patchwork prototypes use combinations of web services, mash-ups, locally developed code and open source software. Both web mash-ups and patchwork prototyping emphasize the central importance of direct user involvement, mitigating lengthy development periods between idea conception and realization.

The concept of patchwork prototyping originated from the observations of Jones et al. on how developers in a series of projects were using OSS and other software to which they had source-code access [12]. It is optimized for ill-defined situations where neither the developers nor the users have a clear idea of what they need the software to do, but rather have an idealized vision of the kinds of things computing technology might enable users to accomplish. Patchwork prototyping is also compatible with community-based initiatives where developers create an environment which is flexible enough for community members to continue to contribute to the development process without the developers' aid [13].

The key to the method is that it is user-driven. The development proceeds and design decisions are made based on the users' collaborative experience of integrating the software into their every-day activities, not based on abstract design principles or predictions of what the users might need.

3.1. Description of patchwork prototyping

Patchwork prototyping has three key components:

- Rapid iteration of high-fidelity prototypes;
- Incorporation of the prototypes by the end users into their daily work activities;
- Extensive collection of feedback facilitated by an insider to the user community.

When integrated, these components create a successful design because developers gain access to and respond to the needs of users while those needs are co-evolving, both due to the effects of the introduction of the software, and due to the ever-changing work or community environment.

Patchwork prototyping is a participatory design technique, as it is a type of cooperative prototyping [4], [15]; however, it blends the design and implementation phases of the development process, because the prototype is incorporated almost immediately into users' everyday activities, and because production-scale modules can gradually be introduced as they have been created to replace the OSS applications that were used as prototypes to develop the requirements.

Patchwork prototyping requires a design team consisting of both developers and representatives of every kind of user. The method entails the following five stages, and an entire iteration normally takes no longer than a week:

1. Make an educated guess about what the target system might look like;
2. Select tools which support some aspect of the desired functionality;
3. Integrate the tools into a rough composite;
4. Deploy the prototype, solicit feedback from users;
5. Reflect on the experience of prototype building and on the user feedback, and repeat - quickly.

For the most part, these steps are relatively straight-forward. We provide a summary of the method below, but for a more in depth discussion see [12].

Making the first educated guess about what the target system might look like is the hardest step, because it requires the design team to synthesize their collective knowledge and understanding of the

problem into a coherent design. In early iterations of the process it is often helpful to use paper prototypes and scenarios but their function is primarily to serve as communication devices and brainstorming aids. The high equivocality of the situation almost guarantees that whatever design is produced will be insufficient. This is not a failure. It is an expected part of the process, and the design will be improved on subsequent iterations. The important thing is to have a starting point which can be made concrete, and not to spend more than a couple of weeks hashing out ideas, unless the problem space is still being explored. The key is not to become bogged down in controversies about how the software 'ought' to look, but rather to put together a prototype and test it out with users in their everyday environments and let the users figure out what works, what does not, and what is missing.

The rapid iteration and high-fidelity nature of the prototypes is vital to patchwork prototyping. High-fidelity is necessary because many users have difficulty imagining what software described by other methods such as paper prototypes, scenarios, or feature descriptions will actually do, and how they might incorporate it into their daily activities [15]. In such discussions users might get excited and mention several possibilities, but those possibilities often turn out not to be feasible for a number of reasons unforeseen by either the users or the developers (sometimes for reasons that are impossible to foresee).

Rapid iteration is vital for both social reasons and design improvement. Socially, rapid iteration is important because users are embedded in their own, hectic environment. In a work environment users' focus is on getting their job done, meeting deadlines, dealing with office politics, etc., not on designing software to support these activities. Thus, users will quickly become frustrated with long turn-around times, and become dependent on and adapted to particular implementations which are less than ideal. When a particular prototype has been in use for an extended period of time, users no longer feel that they are trying out a prototype and start thinking about the system as a final product. Additionally, fast response times make users feel like an integral part of the process, where what they contribute is immediately used to improve the software. Maintaining such feelings is vital in order to keep obtaining high-quality feedback from users, and to prevent indifference from setting in about the design process.

Rapid iteration also improves the quality of the design. It allows for the exploration of more features and alternatives. This can uncover overlooked aspects of the system which might be of use. This can also reinforce the importance or necessity of particular features or requirements. Furthermore, iteration provides users with a constant flow of new design possibilities, which gives them the capability to criticize particular instances of the prototype. In addition, the design team can improve their understanding of the broader sociotechnical system [14], [23], because they have seen many design ideas fail, and come to an understanding of why each of them failed from the users' feedback. Ultimately, it is impossible to reach complete understanding of the system given its evolving nature. However, by iterating the prototyping process, the design space may narrow, identifying a set of key requirements. At this point the design is not complete, but work on a flexible production-scale system can begin, and further exploration of the design space can be continued within that system.

The rapid iteration of high-fidelity prototypes has long been the holy grail in prototyping research. Concepts like horizontal vs. vertical prototypes, and high-fidelity vs. low-fidelity prototypes [9], [18] were developed specifically to understand and take advantage of the trade-offs involved in picking one prototyping technique over another. It is only with the development of Web 2.0 APIs, techniques and mindsets, and with the rapid proliferation of high quality OSS software that we are truly close to realizing this vision.

Patchwork prototyping takes full advantage of these new technologies. The basic form for such a prototype is a modular patchwork of various OSS applications and Web APIs. These can easily be switched in and out, turned on or off, or reconfigured in how they are wrapped into the interface. The minimal effort required to add features allows programmers to treat them as disposable, because little effort was needed to implement them, so little effort is wasted when they are switched off or discarded. This facilitates the requirements gathering process, because iterations of the prototype can be rapidly created, with high functionality, at low cost. Deciding between shallow and deep integration, however, can be a matter of considering the tradeoffs between having data flow between modules vs. increasing the facility of exchanging one application for another. The key is to have a prototype where there are many features and options which can be easily turned on, off, and back on again as users require or wish to explore, thus allowing users to explore via action, trial, and error, rather than by trying to conceptualize precisely how the system will work ahead of time.

Access to the source code of component applications and the freedom to modify it is not an essential prerequisite to development by integration. Over many years public APIs to closed proprietary source code have facilitated the development of

thousands of innovative applications in various software platforms. Nevertheless, source code access can be very useful. Without it, developers are limited in how well they can patch different modules together, in which features they can enable or disable, in how quickly they can enable or disable them, in how they create a visual integration with the rest of the system, and in their ability to understand the underlying complexity of the code which they are integrating – and will likely have to rewrite themselves for the production scale version. By using and delving into the open-source code, developers can get a feel for how complicated it will be to implement a particular feature robustly, and can make better estimates for the costs to implement a particular feature.

During deployment of the prototype, users integrate the software into their work practices for an extended period of time and collaboratively explore what they can do with it. The feedback of user experiences allows requirements gathering which is not purely need-based, but also opportunity- and creativity-based. By seeing a high-fidelity prototype of the entire system, users can develop new ideas of how to utilize features, and conceptualize new ways of accomplishing their work. In addition, users will become aware of gaps in functionality which need to be filled, and can explain them in a manner that is more concrete and accessible to the developers.

When reflecting on the collected feedback, however, the design team (including representatives of all stakeholders) must realize that the prototype does not simply elicit technical requirements; it elicits requirements for the collaborative sociotechnical system as a whole. The existence of the prototype creates a technological infrastructure which influences the negotiation of the social practices being developed by the users via the activities the infrastructure affords and constrains [16]. The design team must be aware of how features of the prototype are affecting the development of social practice, and must consider how to redesign the system so that desired social practices are supported and encouraged by the structure of the system (in addition to any social means of encouraging or requiring the practices). The design team must also be sensitive to the needs of users not on the design team, in order to avoid creating deleterious power imbalances which will doom the effort to create an acceptable collaborative system (the disempowered will not be interested in collaborating). By allowing users to interact with the prototypes for extended periods, collecting feedback on their experiences, and paying attention to the social consequences of the cyberinfrastructure, a richer understanding of the sociotechnical system as a whole can emerge. Reflection is a process of attending to the consequences of the design on the broader sociotechnical system, and integrating these into a holistic understanding of how the system is evolving.

4. Case studies

In this section we present four case studies which illustrate various aspects of creating mash-ups or patchwork prototypes. The examples are meant to give a flavor of the two methods, and to illustrate some of their relative advantages.

4.1. Wasabe: an example mash-up

The authors developed a web mash-up called Wasabe[1] (an acronym for the Wikipedia-Amazon Search And Browse Environment) as a prototype hybrid library catalog system that allows users to search within a single interface both the detailed bibliographic information typically found in library catalogs as well as more general information about the topic of interest, typically found in encyclopedias (Figure 2). Wasabe is a mash-up that demonstrates all three key features of most mash-ups: the use of the computational power of web services, access to large amounts of real content, and the speed with which mash-ups can be created with a minimum of effort.

The first version of this system used the Amazon E-Commerce API and the Google SOAP Search API to execute a user-initiated search of both Amazon's book database and Wikipedia's articles (this functionality is now present in the A9 search engine which allows for side-by-side searching of multiple sources; the first Wasabe prototype was created before the A9 release). The authors were able to build the first Wasabe prototype in less than 10 minutes, writing only 100 lines of PHP code.

Two subsequent revisions have been made to Wasabe to connect the search results to our university's library catalog system. The second

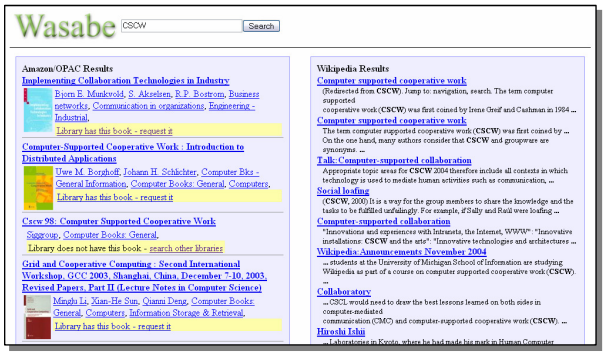

Figure 2. Wasabe mash-up prototyping a hybrid library catalog search.

version added 30 more lines of PHP code. These extra lines expanded the functionality in two ways: first, they recursively searched the Amazon database, using Amazon's recommendations to find related items and their ISBN's; second, the ISBN's were appended to a library web catalog search URL, used to query the catalog and determine whether the book was available.

Searching the library's catalog on the server side proved to be too slow, so a third version was written using AJAX (Asynchronous JavaScript And XML) techniques to perform the same operation and load the data on the client side. This version has a combined total of 125 lines of JavaScript and PHP code.

While the final version had three times more code than the first, the total amount is still very small considering the functionality it provides. It is also worth noting that very little of the code in any version is significantly more complex than simple looping operations to count things up or print things out. Despite being created by an experienced programmer, the speed at which Wasabe was created and the simplicity of the underlying code were amazing.

The access to large amounts of real content was also vital to Wasabe's success as a proof of concept. The nature of the research question being asked in the Wasabe development necessitated a large catalog of books, a database of user browsing and purchasing habits, and an extensive encyclopedia of information. Arguably, such a prototype could only exist as a mash-up. Attempts to prototype the system using a small sampling of data or a mocked-up database of records, would be unlikely to have yielded many insights into its utility as it would have constrained the user experience to performing artificial tasks. By harvesting real data, the authors were able to demonstrate the utility of including both bibliographic and contextual information within the same interface.

4.2. Teaching, simplifying and democratizing mash-ups

As mentioned above, mash-up development currently requires a diverse knowledge and skill set. We suspect that most of the confusing details of mash-up creation are not inherent to the concept and can be simplified through a mixture of social (teaching and explaining) and technical (better design environments and toolkits) means. Through such means the barriers to creating web mash-ups can be lowered even farther.

As a preliminary investigation of this, one of the authors recently taught an undergraduate course on Web Technologies as part of a Minor in Information Technology Studies. Students were sophomores, juniors, and seniors from a range of majors including graphic design, psychology, political science, finance,

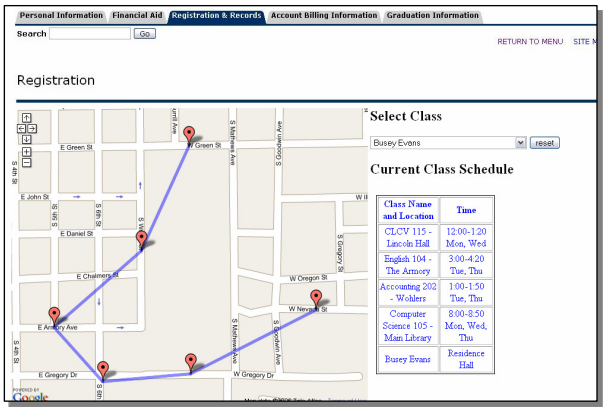

Figure 3. Campus route planner mash-up built by students to help plan a class schedule.

comparative literature, media studies, and rhetoric; most students had no prior programming experience. As part of a 15-week semester covering a range of other topics, the students' final team projects were to build a prototype web mash-up of their choosing using their newly-gained knowledge of HTML, XML, CSS, JavaScript and other related technologies. All of the groups took a very need-oriented approach to the project and developed ideas which satisfied perceived needs in the students' lives.

One team decided to tackle a common problem with course registration [7]. As new undergraduates unfamiliar with all of the buildings on a large campus, they had each experienced the pains of having scheduled consecutive classes at opposite ends of campus, leaving them an impossible distance to cross in the ten minutes between classes. The team decided to create a web mash-up which would combine course time and location information with a map-based interface, so that students could see how far apart the buildings were and plan their schedule accordingly.

The prototype combined Google Maps with a sampling of courses harvested from the university timetables. In the campus route planner mash-up, when students select a course, it is added to their daily route, showing them the distance they would have to travel and giving them an overall picture of how much walking they would have to do each day (Figure 3).

Most of the students in the class had no prior programming experience, and the only experience the class provided them with was a brief introduction to JavaScript. Yet by the end of the course they were able to create functional prototypes. Their primary method for creating prototypes was to copy code from existing mash-ups and modify and incorporate it into their own work. This suggests that an explosion of web mash-ups made by non-technophiles similar to the explosive growth of the web by non-technophiles copying HTML pages is a distinct possibility in the near future.

4.3. Patchwork prototyping in a cyber-collaboratory

We were involved in a project building a cyberinfrastructure for environmental engineers. In this project, the developers built a prototype cybercollaboratory using as the foundation an open-source portal called Liferay. One of the key features of this project was how rapidly the prototypes were created (new iterations were often ready in less than a week), and, as a result, how un-invested the developers were in any particular version of the prototype. The following example illustrates how through user feedback a particular function within the prototype was changed over time.

At an early stage, based on feedback from the stakeholders, a need was identified for users to collaboratively edit documents in the system. To provide this functionality the developers simply enabled a wiki portlet available for Liferay. However, users found the wiki too difficult to use, partly because of confusion with the wiki-markup syntax, and partly because they had no immediate tasks which clearly lent themselves to the use of the tool. Later, some members of the design team wanted to demonstrate the usefulness of scenarios and personas in facilitating requirements gathering. Based on their prior experience of success with this approach they suggested using a wiki. In response to this request and the prior difficulties in using the bundled wiki, the developers installed MediaWiki on the server, and added a link from the CyberCollaboratory's menu next to the existing wiki tool pointing to the MediaWiki installation. No time was spent trying to integrate the Liferay and MediaWiki systems; each application had separate interfaces and user accounts. They were only connected by a simple hyperlink and thus in users' conceptions. A benefit of using MediaWiki was that it allowed people to use the system without logging in, thereby mitigating the need to integrate authentication mechanisms. Users found the MediaWiki system significantly easier to learn and use, and became eager adopters, using it exclusively over the built-in Liferay wiki. The wiki was later embedded in the Liferay system using an HTML IFRAME, and the authentication mechanisms of the two systems were eventually integrated.

As users began incorporating the prototype into their daily activities, it quickly became clear that different users in different social contexts needed different means of interacting with the tool. Some people in administrative roles needed the collaborative editing functionality integrated with the rest of their real-life administrative activities, because that was their primary use of the tool. Others were mostly using the tool for exploring the nature of what should be built next, for example by generating and refining scenarios. They wanted the tool kept visually distinct from the other functions of the system because they saw it as a separate module of the system, devoted entirely to a particular task. This difference in needs and use raises two issues. Firstly there is the common problem of uncovering the different uses and users that the software needs to accommodate, which leads to various incremental design tradeoffs. But additionally, there is an interesting consequence of being able to develop prototypes that are robust enough for some everyday use. This led to real life use needs interacting with more conventional experimenting with a proof of concept. Without paying attention to how social roles and workflows were evolving, the designers would have been unable to properly incorporate the tool into the system.

The Liferay portal offered developers the opportunity to explore other features as well via tighter integration with the extensible Liferay framework. The developers built prototypes of research tools for monitoring developments on the web using the Heritrix web crawler and Lucene search engine; incorporated a prototype of a GIS system using the open-API Google Maps system; and built an awareness monitor using RSS feeds. They also used numerous existing portlets already written for Liferay. Not all of the imported applications were publicly available OSS; some were in-house applications developed by other projects, for which developers had complete access to the source code. These were used to build a data-mining application and a knowledge management tool.

Common through all of these experiences was the relative ease with which the developers were able to rapidly explore different options and variations. The prototype changed over time reflecting the developers' evolving understanding of users' needs.

4.4. Patchwork prototyping in community inquiry labs

Community Inquiry Labs (iLabs) are part of a project investigating the design and development of web-based tools to support inquiry-based learning and teaching. The iLabs system allows groups of users to create a collaborative space, customized in the number, type, presentation and description of various core tools to support information creation and sharing, communication and collaborative interaction. In this example we focus on the ease of integrating OSS into an existing prototype.

In the earliest version of iLabs, users expressed an interest in having a bulletin board tool. The developers

selected the phpBB system and manually installed copies of phpBB for each community that wanted a bulletin board; the bulletin board was simply hyperlinked from the community's iLab.

In the next iteration of the prototype, the phpBB system was modified to be more integrated with the rest of the prototype. The integration of phpBB took a developer an afternoon and required modification of one file in the phpBB system, adding about 25 lines of new code (much of it copied from other functions in the phpBB code) and modifying two other lines elsewhere in the same file. A function was added to the iLabs source (about 30 lines of code) containing the SQL statements needed to create a phpBB forum and associate it with an iLab, and a hyperlink was added to the interface to execute this function.

The minimal coding effort had a big payoff: it integrated the full functionality of the phpBB system with iLabs. Users could now install a bulletin board themselves, without involving the developers, by clicking a link on the interface. Furthermore, bulletin board authentication and account management was integrated with the rest of the prototype, eliminating the need for users to log in twice.

5. Discussion

There are two perspectives from which one can describe the similarities between web mash-ups and patchwork prototypes. One is technical, involving the common property of drawing upon a disparate variety of computational resources including APIs, OSS, and web services in general. The ease with which all of these components can be combined and the relative power of the resultant combination are significant when compared with the amount of time and effort it would take to code a similar result in more conventional ways, even with extensive use of software libraries and other traditional forms of code-sharing. The sharing involved in using APIs and OSS software goes beyond the sharing of algorithms typical in code libraries. It is a sharing of development experience, as the massive amount of effort put into creating web services and OSS has already discovered and overcome a whole series of bad design ideas the hard way, which is something the mash-up or patchwork prototype developer would need to re-experience using traditional methods. As such, these approaches seem to get to the heart of the promise and potential of software reuse, advocated in software engineering research for many years but rarely attaining the levels of adoption and efficiency predicted for it.

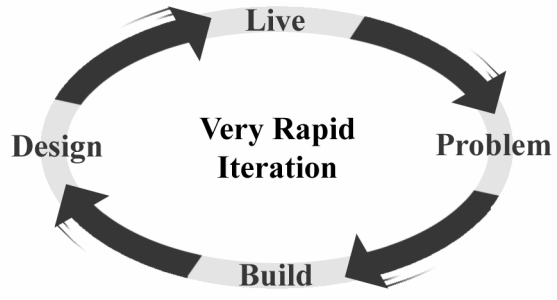

Figure 4. The user-driven model.

The other perspective is social, involving the user-driven nature of both processes. There is a long history of technology users being side-lined in development. The problem became so acute, that entire academic fields such as Human-Computer Interaction (HCI), Social Informatics (SI), and Science and Technology Studies (STS) have developed in order to understand and rectify this problem. However, even the most democratic and inclusive methods for including users in the design process, such as participatory design (PD), have only a mixed history of success.

The failures of PD are often attributed to an incorrect application of the method [3], [5]. It is interesting to note that many of the projects which have used traditional PD techniques have been initiated by management with the goal of increasing workplace efficiency, software companies trying to develop new software, or anybody else besides the people who will actually be using the technology. While this fits with the overall PD agenda of empowering workers and ensuring that new software they are compelled to use does not adversely affect their working conditions or job security, it is still a far cry from technological innovation driven by user needs and desires [15].

Both web mash-ups and patchwork prototyping are phenomena which have been observed first, and then formally described, rather than invented deliberately and implemented according to a model of how things work, or how things might work better. As a result, they are reflective of the models users have for using technology in their lives: i.e., a problem driven model. This emergent model seems to call for a reformulation of the traditional concept of design.

In traditional models, like the waterfall model [22], the conception is usually linear, and can be captured by the following oversimplification: Design → Build → Use. The emergent user-driven approach calls for a more circular model (Figure 4). In this model, the starting point is people's every-day lives. In the course of living their lives, they encounter problems. These problems may be with existing technologies that they happen to be using, or they could be problems which have little to do with technology, but which people think that technology could be used to solve. Because

of the ease with which mash-ups can be constructed, or patchwork prototypes restructured, a quick fix is built that addresses the immediate problem. This building phase is then followed by a reflection on how the new fix plays into the comprehensive design of the system, and how else the need might be addressed through a reformulation of the current concept of the system design. After the system has been adjusted to take into account the new design, it is reintroduced into people's lives, and they continue with their every-day activities until the next problem occurs. Thus, a key feature of this model is not just its shape but how quickly it is possible to cycle around it.

Of course iterative and spiral models have been advocated for a very long time [6]. The methods described here are in that tradition, but emphasize the impact that very rapid prototyping by assembly of pre-existing components can have on speeding up the iterative cycle. As a result, each step of the process needs to be less thorough as more cycles are possible in the same time, with more opportunities to identify and correct errors. This in turn means that the informality and creativity, indeed playfulness of the design processes can mesh with similar approaches for requirements capture and evaluation, fitting most appropriately with the aims and ethos of PD.

The extent to which the design component is present in web mash-ups may be debatable. Most web mash-ups of any duration, however, end up going through several iterations of progressive refinement of the concept. Thus, while the reflection on the design might not be as explicit a part of the process as it is in the patchwork prototyping model, it is clearly present in the mash-up programmer's reflection on the personal use of his or her mash-up, or the comments received from other users.

5.1 Contribution of recent trends

The recent trends encapsulated by the term Web 2.0 and open participatory movements such as OSS provide components allowing user-driven approaches to be more successful. Web mash-ups are fundamentally dependent on the vast array of APIs currently available, and the relative simplicity of integrating the APIs into working code. They are also dependent on the Web 2.0 service model which companies such as Amazon and Google have epitomized. These companies provide the fruit of considerable development resources vast computing power and vast amounts of organized content to innovators, essentially for free, and the creativity and diversity of ideas of how to combine and recombine these various services is evident from the number of mash-ups currently being created.

Similarly, patchwork prototyping would be impossible without the vast array of high quality OSS applications that exist today. Without this quality code, developers would be unable to customize and glue together applications as quickly and easily, and would have to forsake the speed that is the essential point of the technique. Without having several different high-quality applications to choose from to prototype any part of the system, it would not be so easy to switch out a module and replace it with a more appropriate one as the needs and desires of the users evolve and are refined. The key is to have high-fidelity modules, and if the modules used in the prototyping process are buggy or unreliable, the users will simply be frustrated by the prototype, and not be able or motivated to use it to explore the design space. Thus, while patchwork prototyping may seem like an obvious solution for eliciting design requirements, it was impossible to do before the OSS movement became both strong and prolific.

6. Conclusion

Web mash-ups and patchwork prototyping are two methods enabling user-driven design that are now possible with the technologies and mindsets that accompany recent trends in software development such as Web 2.0 and OSS. The methods are not wholly new. They are firmly rooted in both formal traditions of software reuse and component based programming, and informal techniques of tinkering and experimenting with toy applications and proofs of concept. What is noteworthy is how they manage to combine (even mash-up) these traditions to enable larger numbers of people to produce experimental software that is robust enough to be tested in everyday situations and hence go through very rapid iterations of development and authentic situated evaluation.

With web mash-ups, individuals and small groups are able to create their own technological solutions to the problems they face in their everyday life, without the need to be expert programmers. Patchwork prototyping is a more formal design technique which allows such user-driven technological innovation to occur with the support of developers, and on a larger scale (i.e., to support communities). In both methods technological innovation is initiated by users, and the innovation is driven by user needs and experiences as they incorporate the technologies into their every-day life. From a technological standpoint, the methods are similar in that both take full advantage of the computational power, encoded experience, and diversity of options of various already-built computational tools in an exercise of recombination and bricolage. The end result is better software

because it is specifically geared to meet the needs of the users involved in the development process.

7. Acknowledgements

The authors would like to thank the other members of the design teams we worked with while making the observations that led to the development of these ideas, particularly Luigi Marini and Yong Liu.

8. References

[1] Berners-Lee, T., Cailliau, R., Groff, J.F. and Pollerman, B., "World-Wide Web: The Information Universe", *Electronic Networking: Research, Applications and Policy*, 1(2), 1992, 52-58.

[2] Berners-Lee, T., Hendler, J. and Lassila, O., "The Semantic Web" *Scientific American,* 284(5) 2001. pp 34-43

[3] Blomberg, J.L. and Henderson, A., "Reflections on Participatory Design: Lessons from the Trillium Experience," *Proceedings of CHI'90*, ACM Press, Seattle, WA, April 1990, pp. 353-360.

[4] Bødker, S. and Grønbæk, K., "Cooperative Prototyping: Users and Designers in Mutual Activity", *International Journal of Man-Machine Studies*, 34(3), 1991, 453-478.

[5] Bødker, S., and Iversen, O.S., "Staging Professional Participatory Design Practice – Moving PD beyond the Initial Fascination of User Involvement", *Proceedings of NordiCHI*, ACM Press, 2002, pp.11-18.

[6] Boehm, B., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, 21(5) 1988, pp. 61-72.

[7] Dombrowski, A., Marselle, J. and Szot, L., "The Banner Schedule Route Planner" Technical Report ISRN UIUCLIS--2006/12+CSCW, 2006.

[8] Feldstein, M. and Masson, P., "Tutorial: Unbolting the Chairs: Making Learning Management Systems More Flexible", *eLearn*, 2006(1), 2006, p. 2.

[9] Floyd, C., "A Systematic Look at Prototyping", In Budde, R., Kuhlenkamp, K., Mathiassen, L., and Zullighoven, H. (Eds.), *Approaches to Prototyping*. Springer-Verlag, Berlin, 1984, pp. 1-18.

[10] Gunderson, P.A., "Danger Mouse's Grey Album, Mash-Ups, and the Age of Composition.", *Postmodern Culture & the Johns Hopkins University Press*, 15(1), September 2004.

[11] Heineman, G.T. and Councill, W.T. "Component Based Software Engineering: Putting the Pieces Together" Addison-Wesley 2001.

[12] Jones, M.C., Floyd, I.R and Twidale, M.B. "Patchwork Prototyping with Open-Source Software" in *The Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives,* St. Amant, K. and Still, B. (Eds), Idea Group, 2007.

[13] Jones, M.C., Rathi, D. and Twidale, M.B., "Wikifying Your Interface: Facilitating Community-Based Interface Translation", Proceedings of *ACM Conference on Designing Interactive Systems,* 2006. pp. 321-330.

[14] Kaghan, W.N. and Bowker, G.C., "Out of Machine Age?: Complexity, Sociotechnical Systems and Actor Network Theory", *Journal of Engineering and Technology Management*, 18(3-4), 2001, pp. 253-269.

[15] Kensing, F., and Blomberg, J., "Participatory Design: Issues and Concerns", *Computer Supported Cooperative Work*, 7(3-4), 1998, pp. 167-185.

[16] Kling, R., "Learning About Information Technologies and Social Change: The Contribution of Social Informatics", *The Information Society*, 16, 2000, pp. 217-232.

[17] Nardi, B.A. and Miller, J.R., "Twinkling lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development", *International Journal of Man-Machine Studies*, 34(2), 1991, pp. 161-184.

[18] Nielsen, J., *Usability Engineering*. Morgan Kaufman, San Francisco, CA, 1993.

[19] O'Reilly, Tim "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software", 2005, *www.oreillynet.com/lpt/a/6228.*

[20] Rademacher, P., "Are You Ready for Web 2.0?", BayCHI, *www.baychi.org/calendar/20050809/*, Aug. 2005.

[21] Raymond, E.S., The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly and Associates, Sebastapol, CA, 2001.

[22] Royce, W.W., "Managing the Development of Large Software Systems," *Proc. of IEEE Wescon*, 1970, pp. 1-9.

[23] Trist, E.L., "The sociotechnical Perspective; the Evolution of Sociotechnical Systems as a Conceptual Framework and as an Action Research Program", in Ven, A.H. van de, and Joyce, W.F., *Perspectives on Organization Design and Behavior,* Wiley, New York, 1981, pp. 19-75.

[24] von Hippel, E., "Innovation by User Communities: Learning from Open Source Software," *Sloan Management Review*, 42(4), July, 2001, pp. 82–86.

[25] Weiss, A., "The Power of Collective Intelligence", *netWorker*, 9(3), ACM Press, 2005, pp. 16-23.