Mashup Patterns: Designs and Examples for the Modern Enterprise
by Michael Ogrinz; ISBN 032157947x, Copyright 2009 Pearson Education, Inc.

Chapter 1: Understanding Mashup Patterns

Collaborators welcome![1]

Introduction

When the World Wide Web was first unveiled, “collaborators” referred to one small segment of the population: nerds.[2] The first browser ran on a computer that almost no one outside of a university or research lab used.[3] The Web itself consisted of a lone site[4] (WWW Growth, Figure 1.1). Yet from this singularity, a new universe would soon emerge.

Figure 1.1

The growth of the World Wide Web: number of Web sites, 1990–2000

The amount of content didn’t grow much until two years later. That was when the first of several “Big Bangs” would occur. In 1993, the first PC-based program capable of browsing the Web was released.[5] Its introduction instantly put the Web within the reach of a far larger audience. Even so,

[1] From Tim Berners-Lee’s first public Usenet post announcing the public availability of the first Web server and browser in 1991.

[2] A contingent of which I am proud to proclaim myself a member.

[3] The NeXT workstation, conceived by computer luminary Steve Jobs.

[4] Tim Berners-Lee invented the World Wide Web in 1989 while working at the CERN Particle Physics Laboratory.

[5] NCSA Mosaic, released in 1993.

Internet connectivity remained largely restricted to universities, research institutes, and corporations. Consumers enjoyed online communities, but generally did so via prepackaged, fenced-in services such as Compuserve, Prodigy, and America Online (AOL). Connectivity was achieved through slow “dial-up” connections over telephone lines. Access to content was typically billed at an hourly rate.

By 1994, the first independent Internet service providers (ISPs) had begun to pop up. By installing special software on their computers, consumers could access the entire content of the Web (almost 1,000 sites!). AOL began to open up Web access for its millions of subscribers. Prices universally moved to flat-rate monthly charges. WYSIWYG ("What you see is what you get") HTML editors appeared and made creating Web pages just a bit easier. In response, the second explosion in Web growth occurred. By 1996, corporations didn't see a Web presence as a luxury, but rather as a necessity. What better way to instantly push content to the consumer? The Web was viewed as a new media channel that offered endless opportunities for commercial success.

If the waning years of the past century had a motto, it certainly wasn't "Collaborators welcome"; "Venture capital welcome" is probably more accurate. Fueled by ill-conceived business plans and wild speculation, a worldwide expansion of the Web's underlying infrastructure took place. Meanwhile, the browser jumped from home computers to cell phones and mobile devices for the first time. High-speed cable and DSL "broadband" connectivity options became ubiquitous. The third explosion was the popping of the Web bubble, which saw these ventures implode en masse when they failed to turn a profit. This event marked the end of the first wave of the Web's evolution, which in hindsight we label Web 1.0.

Web 2.0

In the aftermath of the Web 1.0 crash, the glut of infrastructure kept the costs of going online low. That simple fact helped attract even more users to come online. A few companies began to figure out how to leverage the Web without going bankrupt. Collectively, their embrace of the Internet represented the slow expansion of the Web from that last primordial blast. New marketplaces evolved as sites like eBay linked buyers and sellers from around the globe. These online flea markets, in turn, spawned communities that helped pioneer the concepts behind new social networking sites like MySpace and Facebook.

By 2006, the firms that had simultaneously feared and tried to control Web 1.0 looked up from licking their wounds and saw the dawn of a new paradigm. In a symbolic changing of the guard, “old media” giant *Time* magazine announced the Person of the Year was “You.”[6] There was no great single occurrence that made this milestone possible. Rather, the driving force was the confluence of many events: the spread of cheap broadband access, the Web-enabling of multiple devices, the arrival of new communication environments, and the emergence of cooperative environments for organizing information. Collaborators were finally running the show.

Industry figurehead Tim O’Reilly is credited with popularizing the term “Web 2.0” to define this new age:

> Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as platform, and an attempt to understand the rules for success on that new platform.[7]

A simpler working definition is that Web 2.0 is a shift from *transaction-based* Web pages to *interaction-based* ones. This is how the power of “You” is mashed, mixed, and multiplied to create value. Social-networking sites, folksonomies (collaborative tagging, social bookmarking), wikis, blogs, and mashups are just some of the components that make this possible. The success of sites such as Facebook, wikipedia, flikr, and digg has demonstrated that democratization of content creation and manipulation is powering the latest wave of Internet growth.

The underlying driver of Web 2.0 is flexibility. The one trait technologies slapped with the Web 2.0 moniker share is that they are extremely (and perhaps sometimes unintentionally) malleable. The successful products don’t break when a user tries to extend them beyond their original design; they bend to accept new uses. Two success stories of the new Web illustrate this principle:

> flickr was started by Caterina Fake and Stewart Butterfield as an add-on feature for a video game they were developing. The idea was to allow players to save and share photos during gameplay. When they realized that bloggers needed a convenient way to store and share photos, Fake and Butterfield started adding

[6] *Time* magazine, December 13, 2006.

[7] http://radar.oreilly.com/archives/2006/12/web-20-compact-definition-tryi.html

blog-friendly features. Opening up their architecture to allow users of the site to create custom enhancements fueled their viral spread. The original game was ultimately shelved and flickr was sold to Yahoo! a year later for an undisclosed sum.

Deli.cio.us grew from a simple text file that its founder, Joshua Schachter, used to keep track of his personal collection of tens of thousands of Web site links. When the site went public in 2003, it spawned a host of add-ons. The concept of associating data with simple keywords to aid in organization wasn’t new, but the cooperative “social tagging” aspect of deli.cio.us resonated with the frustrations of other Internet users.

Enterprise 2.0

Inevitably, when people discover a useful tool outside the workplace, they want to use it at the office as well. This happened years earlier when employees began sneaking personal computers into their offices to make it easier to manage spreadsheets and documents. More recently, end users have imported instant messaging and unlimited email[8] services from external sources.

User demand for Web 2.0 technologies within existing corporate infrastructure is the catalyst for Enterprise 2.0.[9] The challenge for firms is to integrate these new peer-based collaboration models with legacy technologies and mindsets. Figure 1.2 illustrates three areas that established organizations have typically established to control how solutions are delivered.

Figure 1.2

Typical organizational hierarchy

Enterprise 2.0 breaks down traditional divisional barriers and encourages building bridges. The managerial structure does not change, but the ability to conceive solutions and access the technology to deliver them is available to everyone (as shown in Figure 1.3).

Figure 1.3

Traditional barriers to solution delivery are removed in Enterprise 2.0. Each segment of an organization now has equal access to technology. To leverage this new environment, powerful (yet user-friendly) tools are introduced. These tools enable associates outside traditional IT to create their own solutions.

Changing the social structure of a firm is termed “soft reorganization.” Its

[8] When Gmail (Google Mail) was announced in April 2004, it offered 1 gigabyte of message storage. This was well beyond the storage limit most corporate mail systems impose on their employees.

[9] McAfee, Andrew. “Enterprise 2.0: The Damn of Emergent Collaboration.” *Sloan Management Review,* Vol. 47, Spring 2006.

consequence is movement away from fixed roles and responsibilities and toward a more open and unrestricted workplace. The phrase “economies of scale” refers to the cost advantages associated with large-scale production. We term the benefits of Enterprise 2.0 the “economies of collaboration.” How are they established?

- Nontechnical users are empowered to create application solutions without engaging management or IT personnel in the process. This agility leads to shorter time-to-market cycles.
- Folksonomies replace strict taxonomies (see the “Folksonomies versus Taxonomies” sidebar). Newly discovered connections between data and processes can be exploited to add business value.

- New communication tools mine "the wisdom of the crowd" to encourage collaboration and innovation, a technique known as crowdsourcing (see the "Crowdsourcing" sidebar).

Open interaction can help teams discover how the other lines of business operate. This knowledge, in turn, leads to changes that strengthen relationships across departments.

- IT must learn more about the business associates' goals, and create an environment that facilitates the rapid construction of products that they require.
- Members of the business team must participate more directly in the engineering process (either on their own or in partnership with IT), which requires some knowledge about development best practices.
- Management needs to cede some control to other teams and should work with all associates to encourage collaboration. This may entail:

π Funding the necessary infrastructure.

π Allowing cross-pollination between business teams.

π Being open to ideas from nontraditional sources.

Security becomes a universal concern as the lines between teams vanish. The former "checks and balances" approach doesn't work when small teams are creating end-to-end solutions. In this collaborative milieu, firms have to strike a balance between technical controls[10] and education to mitigate risk.

Folksonomies versus Taxonomies

Taxonomies describe the organization of data within a strict hierarchy. In the business world, they are typically artifacts of established corporate structures. The managerial chain of command establishes processes for the composition, categorization, and flow of information. The structure of a rigid taxonomy may be nonintuitive to outsiders and consequently may restrict the sharing of useful information across the firm.

In a folksonomy, the community takes responsibility for collectively classifying and organizing information through a process known as

[10] For example, putting a formal development process with relevant checkpoints and milestones in place.

“tagging.” Tagging simply entails labeling content with a few relevant keywords that describe the information or the ways in which it can be used. As more reviewers add and refine tags, it becomes easier to locate and navigate large amounts of information. The process of tagging creates a dynamic knowledge base of material that is not constrained by conventional organizational techniques.

Crowdsourcing

With crowdsourcing, a problem is framed so that it can be tackled by multiple teams or individuals, working either competitively or as a group effort. User-driven mashups can facilitate this type of mass collaboration in the enterprise, thereby resulting in far more resources contributing to solutions besides traditional IT.

A danger of this approach is that a "herd mentality" might develop that stifles creativity. Some degree of oversight can offset this risk, but care must be taken not to discourage participation.

Crowdsourcing success stories include the Ansari X-Prize, which was designed to encourage low-cost space travel, and Wikipedia, which benefits from the combined contributions of thousands of users.

The Birth of Mashups

> You can have it "good," "fast," or "cheap." Pick any two of the three.
>
> —Classic programmer's adage

Quick, easy, and affordable application development has always been a goal of software engineering. Reusing something that's already been built, tested, and paid for is one of the quickest ways to achieve this objective. From subroutines, to external libraries, to object orientation, to templates, to Web Services, each great advance in programming has been born from the desire to reuse material instead of starting from scratch. The limitation inherent in each of these milestones is that they were created by developers for the sole use by others in their profession.

It seemed inevitable that with the vast amount of new material being placed on the Web 2.0 every second, it could somehow evolve into raw material for software development. Tim Berners-Lee envisioned this leap in Web reusability in what he termed "the semantic Web," which describes a platform for the universal exchange of data, knowledge, and meaning.[11] And while work continues to define new languages and protocols to realize Sir Tim's dream, mashups are making this vision a reality now.

[11] Berners-Lee, Tim, James Hendler, and Ora Lassila. "The Semantic Web." *Scientific American,* May 17, 2001.

Mashup Patterns: Designs and Examples for the Modern Enterprise by Michael Ogrinz; ISBN 032157947x, Copyright 2009 Pearson Education, Inc.

Mashups are an empowering technology. In the past, resources had to be designed for reuse. Application program interfaces (APIs) had to be created, packages compiled, documentation written. The application developers and solution architects who recycled resources were subject to the whims of the original designers. With mashups, you aren’t limited to reusing an existing API; you can *impose* your own if none exists. So if an application or site offers no API, or if you don’t like the access methods that are already in place, you can design and implement your own (see the API Enabler pattern in Chapter 4 for several examples). The promise of achieving programmatic access to almost unlimited data is intoxicating. Even more exciting is the notion that the tools for constructing mashups have begun to reach a level of usability where even nontechnical users can build their own solutions.

Many popular definitions of a mashup would have you believe the term is limited to a combination of Web-based artifacts: published APIs, RSS/Atom feeds (see the “RSS and Atom” sidebar), and HTML “screen scraping.” Although there are certainly valuable solutions in that space, a broader world of data can be mashed up, including databases, binary formats (such as Excel and PDF), XML, delimited text files, and more. The rush of vendors attempting to capitalize on the burgeoning market for enterprise solutions hasn’t helped bring clarity to the field. To turn a classic phrase on its head, we have a ton of nails out there, and everyone is trying to tell us that they have the best hammer.

RSS and Atom

RSS (also known as Rich Site Syndication or Real Simple Syndication) and Atom are formats for publishing Web-based content in a manner consumable by special applications termed “feed readers.” Feed readers aggregate multiple feeds (or “subscriptions”) so that a user can view updates to numerous Web pages from a single environment.

Before RSS and ATOM existed, users had to manually visit each site and check for any new updates. Feeds also serve as a popular method for allowing Web sites to dynamically incorporate content from external information providers. Regardless of their originally intended purpose, because feeds are created using a well-structured format (XML), mashups can easily consume them as a data source.

Another common misconception is that mashups combine at least two disparate sites to form a brand-new “composite” application, complete with a neat new user interface. That’s certainly possible, but mashups need not be an end in themselves. It is more accurate to say that all composite applications are mashups, but not all mashups are composite applications. The enterprise mashup creator can use the technology to transform the Web into his or her own private information source. This data can be used for strategic planning or analysis in systems like Excel or MATLAB. Mashups may also be used to access a single resource at superhuman levels to mine data or migrate content. Creating mashups is all about finding data, functionality, and services and using them to both solve problems and create opportunities.[12]

Types of Mashups

Mashups have several different colloquial interpretations, which has resulted in some confusion regarding the term and its use. The word originated in the music industry, where a mashup was a combination of two or more songs to create a new experience. Typically, the vocal track of one song was combined with the instrumental background of another in this process.

The technology industry extended this definition to encompass a new application genus that described the combination of two or more sources into an integrated site. This technique of development hybridization can be roughly split into two separate categories: consumer mashups and enterprise mashups.

Consumer mashups are generally associated with Web 2.0. They require a lesser amount of programming expertise because they rely on public Web sites that expose well-defined APIs and feeds (see Figure 1.4).

Figure 1.4

A small number of sites with public APIs account for the majority of consumer-created mashups. Source: http://www.programmableweb.com/apis

The output is usually created by one of the sites participating in the mashup. In the classic “show craigslist listings on a Google map,”[13] the API of Google Maps is used to plot and present the feed obtained from craigslist.com. The limitation of this approach was that resources had to be

[12] This naturally presents potential legal complications, as discussed in Chapter 10.

[13] http://housingmaps.com

“mashup ready.”

Enterprise 2.0 mashups (sometimes referred to as data mashups) are more complex. Depending on which solution a firm deploys, enterprise mashups can emerge in several ways:

- Mashups are used solely by IT to rapidly deliver products. Application developers use both internal and external sources to create data mashups and employ traditional coding techniques to create the user interface around them. Users aren't directly involved in the construction process but they benefit from IT's ability to provide solutions more quickly.

- IT creates a set of "mashable" components and gives end users a sand-box environment where they can freely mix and match the pieces together themselves. If users need new components, they have to solicit IT help to create them.

- An organization deploys an environment that lets anyone create and combine his or her own mashups. This approach is the most difficult implementation to manage, but probably has the greatest impact. To understand the challenge of this approach, consider the use of Microsoft Excel in many firms. Users can create spreadsheet-based applications and pass them around without any central oversight of what exists, how it is used, or if it was tested. This friction-free creation and distribution model spreads good solutions as quickly as bad ones.

Whether mashups are used by IT, business associates, or both, their agile nature makes them a key enabler of Enterprise 2.0. Unfortunately, they are not without potential downsides. In an attempt to "deconstruct" the success of Google, the *Harvard Business Review* points out several pitfalls[14] that can hinder success in a culture of open development:

[14] Iyer, Bala, and Thomas H. Davenport. "Reverse Engineering Google's Innovation Machine." *Harvard Business Review,* April 2008.

- As people spend more time experimenting, productivity in other areas can suffer.
- Poor coordination across groups can lead to duplication of efforts and repeated mistakes.
- A constant stream of new products may confuse the organization and its employees.

Despite these potential hazards, the authors indirectly identify the virtuous circle of Enterprise 2.0 (Figure 1.5). As diverse products are combined to create useful new resources, they themselves become fodder for the next generation of useful products. In principle, this process isn't very different from the long-standing goal of reusability that firms have strived for in their applications and architecture. Three important differences arise this time around, however:

In the age of mashups "reuse" is no longer an ivory-tower concept restricted to the purview of application architects. Because end users and developers alike will be creating solutions, *everyone* will engage in the practice of reuse.

The existing approach to reuse front-loads development efforts with additional planning and coding to create open APIs and extra documentation that may never be used. Because mashups impose reusability "after the fact," their creators will build their own APIs and include only the minimum functionality needed.

Traditional reuse practices don't require that a system that leverages existing code or libraries is itself reusable. This leads to implementations that are essentially "dead ends." Mashups are implicitly reusable, which creates a never-ending cycle of potential associations and recombination.

Figure 1.5
The virtuous circle of mashups

Acquiring Data from the Web

> Need input, More Input, MORE INPUT!
>
> —Johnny Five, *Short Circuit,* 1986

As we saw in the last section, the majority of consumer mashups use the public APIs of a handful of Web sites. In the enterprise model, the best potential sources for mashup data may not be as forthcoming. In these situations, it becomes necessary to employ creative techniques to extract information. One of the most common and controversial techniques is often referred to as “screen scraping.” This derogatory phrase carries a long sullied history and is thrown around by detractors seeking to undermine this approach.

Traditional “screen scraping” owes its origins to the early days of desktop computing, when IT departments developed various techniques to migrate “dumb terminal” mainframe applications to end-user computers. Rather than tackle the costly and time-consuming task of rewriting or replacing existing applications, many IT departments used special PC-based applications that emulated the original terminals.[15] These applications could receive the data from the mainframe and extract the contents of the forms presented on the old green-screen systems. User keystrokes were likewise emulated to send input back to the original application. This technique relied on developer-created templates and was both highly position-sensitive and extremely unforgiving. The smallest alteration in the mainframe display would break the predefined template and break the new application.

[15] Such as an IBM 3270 or VT220.

Because of these drawbacks, screen scraping was generally viewed as a hack and a last resort. The negative experiences associated with this approach continue to haunt any solution that promises to extract raw data from a user interface. Before organizations feel comfortable with mashups, users will need to understand how modern methods differ from the brittle approaches of the past.

Too many of us have forgotten that the “L” in HTML stands for “Language.” In HTML, the *description* of the presentation and the *presentation itself* are inexorably bound in most people’s minds. Many view HTML and what is displayed in their browser as two sides of the same coin.

In fact, it is the underlying Document Object Model (DOM) that makes mashup “screen scraping” something that should more appropriately be referred to as “Web harvesting” or “DOM parsing.” When HTML is read by a browser, it is internally organized into a hierarchal structure. The underlying data structure is tree based and much more organized than what the user sees (see “The Structure of HTML” sidebar). HTML elements may contain additional nonvisual information such as the id and class attributes (see “The class and id Attributes” sidebar).

The Structure of HTML

Consider the following simple Web form:

This is the underlying HTML:

```
<form method="POST">
<table border="0" width="250">
   <tr>
      <td width="85">User Name</td>
      <td><input id="user1" type="text" name="user_field" size="20"></td>
   </tr>
   <tr>
      <td width="85">Password</td>
      <td><input id="pw" type="password" name="password_field" size="20"></td>
   </tr>
</table>
<input type="submit" value="Logon" name="B1">
```

```
</form>
```

When parsed by a browser, this HTML is internally organized into a hierarchical structure known as the Document Object Model (DOM). The DOM is more conducive to automated analysis than the presentation users receive.

The class and id attributes

The ubiquitous use of `id` and `class` in HTML make them ideal markers for Web scrapers to identify document elements.

Uses of `id`:

A style sheet selector

```
<P id=”bigheader”>Important Update</P>
```

A target anchor for hypertext links:

```
<H1 id=”news”>Today’s Top Stories</H1>
```

A means to identify an element in JavaScript:

```
document.getElementById("news");
```

Used to name a declared OBJECT element:

```
<OBJECT declare
      id="newyork.declaration"
      data="city.mpeg"
      type="application/mpeg">
      A tour of Manhattan.
</OBJECT>
```

Uses of `class`:

Assign one or more CSS styles to an element:

```
p.error {font-size: 18px; color: red;}
<p class="error">Incorrect Password</p>
```

Beyond their original intent within HTML, `id` and `class` attributes can also serve as "markers" for general-purpose processing by other applications/agents (e.g., mashups). Unlike the screen scrapers of the past that relied solely on positional information to parse screen content, mashups are able to examine the underlying attributes used to build the presentation. Although not a foolproof approach, this data changes much less frequently than the look and feel of a site, as demonstrated in the sidebar "Presentation Changes Don't Break Object Discovery." While consumer mashup builders queue up and wait for content providers to expose an API, enterprise teams are using Web harvesting to grab whatever data they want.

Presentation Changes Don't Break Object Discovery

This example shows a sample Web page before and after a radical redesign. Although a visitor might be disoriented by the drastic changes, similarities in the underlying HTML (and resulting DOM tree) will not slow down a mashup that examines the site.

Before

As part of a larger system, a mashup is created to sign in to a Web site by supplying a “Sign On ID” and a “Password.” The form attributes and DOM information are displayed following the screenshot.

```
...
<td width=”74” height="25"><div class="fixedfont">Sign On ID: </div></td>
<td width=”89” height="25">
<p align=”right”><input maxlength=”20” name=”username” size=”10”
  style="font-family: courier"></p></td></tr>
<p align=”center”>
<tr>
<td width=”74” height="5"><div class="fixedfont">&nbsp;</div></td>
<td width=”89” height="5"><div class="fixedfont">&nbsp;</div></td></tr>
<tr>
<td width=”74” height="25"><div class="fixedfont">Password:</div></td></p>
<td width=”89” height="25">
<p align=”right”><input maxlength=”20” name=”password” size=”10”
  style="font-family: courier" type=”password”></p></td></tr>
<tr>
...
```

After

Even though the site has been radically redesigned, it still contains form elements for “Sign On ID” and “Password.” A peek at the underlying HTML and DOM shows that these fields retain the same attributes. A mashup most likely will not have a problem recognizing the new design, even though a human might take some time to become accustomed to the new interface.

```
...
<tr> <td width="70" class="text_boxsubtitle">Sign-On ID:</td> <td><input type="text" maxLength="20"
name="username" size="10" style='width:122px;FONT-FAMILY: Courier'/></td> </tr> <tr> <td
width="70" class="text_boxsubtitle">Password:</td> <td><input maxLength="20" name="password"
size="10" type="password" style=”width:122px;FONT-FAMILY: Courier”/></td> </tr>
...
```

Enterprise mashups are not restricted to skimming content from HTML: They can leverage more structured formats such as XML (RSS, ATOM), Web Services, or even binary formats such as Excel and PDF (as shown in Figure 1.6). Nevertheless, the great promise of enterprise mashups derives from their ability to treat the entire World Wide Web as a first-class data source.

Figure 1.6
Enterprise mashups can consume a variety of different data sources.

The Long Tail

Although first coined to describe customers who purchase hard-to-find items,[16] the phrase “the Long Tail” has come to have a special meaning in the world of software. Traditionally, application development dollars are directed toward those projects and enhancements demanded by the largest group of users. This practice of catering to the masses doesn’t necessarily lead to an outcome with the greatest positive impact on productivity. Unfortunately, because of the huge effort involved in developing applications, it is often impractical to provide custom solutions to a lone employee or a small team, even if it would greatly increase their efficiency (Figure 1.7). Thus only the “head” of the application demand curve is ever addressed. The exact cutoff point isn’t fixed and will vary by organization, although the Pareto principle[17] or “80-20” rule suggests that 80% of application development efforts will benefit only 20% of your users.

[16] Anderson, Chris. “The Long Tail.” *Wired,* October 2004.

[17] The Pareto principle is based on empirical observation and isn’t a mathematical certainty in all cases.

Figure 1.7

The Long Tail

The cumulative potential of unfulfilled Long Tail opportunities exceeds that of the “head” of the curve. Alas, fulfilling the requirements of the remaining 80% of your staff might seem an impossible goal. Most technology departments do not have enough staff to meet the needs of each individual user. Unless there is a way for developers to become drastically more productive or for end users to solve their own problems, the prospects for meeting unmet demand seem bleak.

Meeting User Demand

> Give me a place to stand on, and I will move the Earth.
>
> —Archimedes

Enter the mashup. Armed with powerful new tools that leverage the resources of the Internet, developers and power users can quickly assemble products to target the Long Tail. We are witness to the dawn of a new era in technology. Mashups are making IT more agile and empowering individuals to create their own solutions.

The Long Tail is useful from an analysis standpoint only if it represents the universe of possible solutions that can be constructed. Consider the mashup example in “A Sample Mashup Use Case.”

A Sample Mashup Use Case

Mashup Patterns: Designs and Examples for the Modern Enterprise by Michael Ogrinz; ISBN 032157947x, Copyright 2009 Pearson Education, Inc.

> There are countless examples where mashups can benefit an enterprise, and they needn't be complex. Consider the following example.
>
> Every day, the employees of a firm have numerous conference calls to discuss project planning, resource management, and corporate strategy. Whenever someone new joins the conference, there is a "beep" that announces that individual's presence. The first ten minutes of every call go something like this:
>
> "Beep."
> "Hi, who's on the line?"
> "It's me, Rob."
> "Beep."
> "Hi, who's on the line?"
> "It's me, Maureen."
>
> On each call, valuable time is wasted while the moderator takes attendance and furiously scribbles down names. Later on, he may try and match those (frequently misspelled) names to an email address or telephone number.
>
> We can save time and expedite the meeting with a simple mashup. First, we visit the conference call Web site and grab the participant's caller ID directly from the Web page. Next, we look up those numbers in the firm's online corporate directory (where we also get the corresponding email addresses). Finally, in case someone is dialing in from his or her home telephone, we use the search form on a public Internet site (such as whitepages.com) to look up any unresolved numbers.
>
> The entire process is hidden behind a simple Web front end with a single button, labeled "Get Attendees." No more misspelled names or missed participants. No more pausing to ask latecomers to introduce themselves. Meetings start on time and everyone is happy.
>
> As if this capability wasn't enough of a breakthrough, it opens up new possibilities for behavior tracking (also known as reality mining). You can click the "Get Attendees" button multiple times during the call to see not only who is present, but *for how long.* Perhaps you can tie that "duration" data to other sources. You might find that callers drop off the line in coordination with weather, traffic patterns, or surf reports.

Although the "conference call attendance" issue was experienced by almost all employees of the firm, it was never identified as a business problem. This is because developers and business users are conditioned to view their actions in discrete, isolated chunks:

- First, I sign into Application A to locate a customer’s account.
- Second, I sign into Application B to check item inventory.

- Third, I sign into Application C to create a purchase order for the client.

If you accept that Applications A, B, and C are immutable (perhaps because they were purchased from an external vendor), then you will never envision a solution where you can sign into Application D *once* and perform these three actions in a single step. The opportunity never appears on the Long Tail.

The greatest benefit of mashups may be their influence on our thought process. When we cast off our biases about the role of technology in the workplace, we discover the folly in applying IT to only the most obvious and well-understood problems. Once the blinders have been removed, you'll discover a world of missed and previously unknown challenges that you can tackle. Recognizing these opportunities is just the first stage. If you don't do something about them, then you've simply added to the tangle of unmet expectations. To achieve continuous innovation, it is essential to look outside the existing methods of measuring and meeting user demand.

Mashups and the Corporate Portal

The concept of aggregating data from multiple sites inside and outside the workplace isn't new. As companies struggled to share all of their disparate applications and information resources directly with their employees, many embarked upon a quest to create a single corporate portal. An organization's portal typically provides several features:

- Single sign-on (SSO), which allows users to authenticate only once to obtain access to multiple applications.
- Multiple "portlets" or "islands" that expose information and functionality from disparate systems.
- Interaction (or integration), which allows portals to influence one another's behavior. For example, a search portlet may cause the contents of other portlets to be filtered.
- Access control, which provides for the centralized administration of which information a user may access. A user's permissions on the portal are at least as restrictive as what the user would receive if he or she logged into the underlying application directly. Portals are unique in that they may bring content together from multiple sources wherein the user has varied entitlements.

- Personalization, which allows the user limited ability to customize the layout and presentation of the site to suit his or her own specific tastes and needs.

Of course, as our examination of the “80-20” rule suggests, portals will never meet the requirements of all users, all of the time. At best, they may meet the lowest set of common requirements across a broad audience (the 80%). The most specific requirements are typically the least general (the 20%), which explains why most corporate portals typically confine themselves to broadcasting company news, managing health and benefits information, and tracking the holiday calendar. Personalization, the latecomer to the portal infrastructure, was a desperate attempt to address this shortcoming. Unfortunately, users typically don’t get a say in choosing *which* content can be personalized or *how* it can be manipulated.

At my daughters’ nursery school, their teacher maintains order by telling the children, “You get what you get and you don’t get upset.” Those days in computing are pass . Whether we are talking about the corporate business user who wants to come to the office each day to a personalized workstation or a customer who wants to view your company’s information in a certain fashion that suits his Web-based applications, this is the age of individualized construction.

When the popular social networking sites MySpace and Facebook published open APIs to leverage their data and create interfaces around it, thousands of users became bona fide developers. They quickly learned to build their own personal portals. This same demographic is just now beginning to enter the Enterprise 2.0 workforce. They won’t be content to operate within the confines of a single, stoic portal that restricts how they consume and manipulate information.

A new metaphor for user interaction has recently emerged that, combined with mashups, threatens the relevance of the enterprise portal. Whether you know them as widgets, gadgets, or snippets, they are the small plug-in components that originated on the Web and have migrated to the desktop (e.g., Apple Dashboard, Yahoo Widgets, Google Gadgets, Microsoft Vista Desktop Widgets). The tools for creating these “mini-applications” have become easier to use and more familiar to a much broader audience.

If enterprise mashups are the path to user-created data and widget platforms are the environment for presenting that information, the combination of the two represent the death knell for the corporate portal. At best, it will morph into a set of core services that provide information to mashup-powered personal environments.

Mashups and Service-Oriented Architecture

Service-oriented architecture (SOA) has come to be associated with Web Services, but at its core it is more mindset than methodology. The “service” in SOA shouldn’t be thought of in terms of a particular technology, but rather as a *business task.* The tasks are implemented in an environment that facilitates loose coupling with other services. This combination, in turn, fosters an atmosphere where developers can create new applications that reuse and recombine existing functionality. Because the services are based on open standards, they can be consumed equally well across independent development platforms.

The promise of SOA is that it addresses the Sisyphean[18] labor of building duplicate, siloed functionality across the enterprise. Better yet, you don’t have to build services yourself; you can discover and use third-party solutions. SOA is the equivalent of a home improvement store for application development. You simply fill up your shopping cart with all the raw materials and glue and nail them together in your basement to create a shiny new product. Using a traditional development mindset would place the burden on you to chop down trees for lumber or smelt the iron for nails.

The Common Object Request Broker Architecture (CORBA) was an early stab at implementing SOA—so early, in fact, that it predates the Internet explosion of the mid-1990s and even the SOA acronym itself. The level of complexity required to work with this technology was often found to outweigh its benefits, and while CORBA struggled to find its footing, newer technologies such as SOAP, XML, and Java (Enterprise Java Beans) arrived on the scene. They began to address the problems associated with CORBA’s steep learning curve and security shortcomings.

Web Services emerged as a technology-agnostic interoperable solution based on open standards such as XML, WSDL, UDDI, and SOAP. Although far from perfect,[19] SOAP-based Web Services have become the industry-preferred method for implementing SOA. The most popular method for exposing SOAP services across the enterprise is via a custom infrastructure

[18] Sisyphus was a Greek who was condemned by the gods to ceaselessly roll a rock to the top of a mountain, only to have it fall back of its own weight.

[19] Problems include interoperability issues and platform-specific implementation, testing, and security challenges.

known as an enterprise service bus (ESB). The ESB can provide additional data transformation capabilities, security, transaction support, and scalability, all while simultaneously reducing the degree of complexity exposed to service reusers. In an attempt at product differentiation, some ESB offerings service-enabled existing corporate resources (such as databases) and were themselves progenitors of the data mashup.

One point should be clear: SOA is not a revolutionary milestone but an evolutionary one. Open communication and technology standards, combined with the ubiquity of the protocols that power the Web, have finally helped SOA reach a level of maturity where its benefits exceed its costs.

Mashups represent the next leap in reuse. They initially came about when developers combined the published APIs of different Web applications to create interesting new content. The limitation of this approach was that resources had to be “mashup ready.” Robust SOA environments were a hothouse for mashup growth, as they exposed componentized functionality that could be mixed together to provide new services.

You may be wondering if mashups are the latest harbinger of SOA, or the beneficiary of it. The answer is a resounding “Both!” With most vendors now using the terms “SOA” and “Web Services” interchangeably, it has become obvious that for most corporations, implementing a successful SOA will require the service-enablement of their existing applications. Mashups are a completely valid method of accomplishing this (see the “API Enabler” section in Chapter 4 and the discussion of the Quick Proof-of-Concept pattern in Chapter 7). Most mashup products allow you to create and publish Web Services either directly or via a third-party application container (e.g., WebSphere or JBoss). Likewise, mashups are voracious consumers of Web Services. Mashups gladly leverage the Web Services that SOA-centric organizations already have in place. Because mashups can produce services with the same agility that they consume them, they are a valuable addition to any service-oriented environment.

How do SOA patterns and mashup patterns relate to each other? SOA generally focuses on server-side architecture and internal corporate resources, whereas everything is fair game with mashups. Because of SOA’s maturity and association with Web Services, it has achieved greater clarity regarding its capabilities, protocols, implementation, and use. This allows SOA pattern discussions to focus on high-level abstractions. Indeed, several excellent Web sites and books[20] discuss the process of SOA-enabling the enterprise. Mashup patterns, which remain in a nascent stage of development, must focus on more practical examples. This will drive broader

[20] Author Thomas Erl has written several good books on this subject, including *SOA Design Patterns.*

adoption, which in turn should to lead to consolidation and standardization similar to what SOA has achieved.

Mashups and EAI/EII

Enterprise application integration (EAI) is the practice of connecting corporate systems at the application level rather than at the data level. EAI solutions seek to streamline business processes and transactions, whereas mashups typically combine applications with the goal of providing new functionality. EAI tools rely on support for open standards such as Web Services or CORBA. If an application doesn’t expose an API, one needs to be constructed programmatically. As systems and requirements evolve, there is an inevitably large carrying cost to maintain the custom integration code. When managed and funded correctly, EAI can provide the most rock-solid method of application integration. For business-critical solutions, EAI is recommended over mashups, which permit some fragility as a trade-off for the benefit of agility.

Enterprise information integration (EII) is a data management strategy for providing uniform access to all the data within an organization. The rise of “big box” stores that sell everything from baby clothing to car tires has demonstrated that patrons appreciate the convenience of one-stop shopping. Collecting data from multiple sources and providing a single point of access has similar appeal in the enterprise. EII is often easier to achieve than EAI because it simply attempts to unify information and not applications. If you think this approach sounds similar to a data mashup, you’re correct. A mature EII implementation can provide new insights into data associations and facilitate rapid solution delivery. EII tools have historically focused only on back-end databases,[21] which limits the range of information that can be collected. By comparison, mashups surpass EII in their ability to obtain data from both structured and unstructured sources.

The knowledge requirement for successfully applying EII technology is higher than that for mashups, but as with EAI the advantage is stability. You can measure the benefits of a complex EAI/EII project empirically by developing a quick mashup-based prototype (see “Quick Proof-of-Concept,”

[21] These databases include relational databases, message queues, and data warehouses.

Chapter 7). This effort may help determine whether the potential benefits justify the considerable cost and time required to carry out a formal implementation.

Mashups and Software as a Service

In contrast to the architectural style and Web Service implementation strategy of SOA, software as a service (SaaS) is a business model. SaaS is the latest incarnation of the Internet-boom idea of an application service provider (ASP). Under the SaaS plan, businesses do not invest money to develop and host applications internally, but instead rent the functionality they need from an external service provider. End-user interaction with applications typically occurs via a prebuilt Web interface. The customer’s business data is then fed into the system manually, using Web forms, or programmatically, using a Web Service API.

To appeal to as broad a market base as possible, most SaaS providers have focused on generic services and priced them competitively (a fee of less than $100 per service is not uncommon). Exposing macro capabilities and parameterizing functionality allows customers to achieve some degree of customization.

One of the most prominent success stories in SaaS is Salesforce.com. This “zero-infrastructure” customer relationship management (CRM) platform provides services to thousands of businesses worldwide. Small and large customers alike are able to start using the hosted service almost immediately without deploying custom hardware. The success of Salesforce.com has led many to assume SaaS is particularly well suited to CRM and sales force automation. In reality, this isn’t the case. WebEx, a Web-based conference and collaboration solution, has achieved adoption on an even larger scale. Google Apps is an example of a viable alternative to traditional desktop software. It serves up a business-focused mail, spreadsheet, and word processing suite at a fraction of the cost of Microsoft Office. Many commercial vendors are exploring SaaS to create new revenue streams.

Assuming SaaS products can meet technical and functional user requirements, two key challenges must be overcome before SaaS can succeed as a general distribution model. First, firms must be comfortable with the notion that their data is housed externally to the organization. It seems that there’s a new story almost every day in the press about missing hard drives or accidentally leaked personal information. SaaS providers may have better security than many of their clients, but the abdication of data management to a third party is still a tough pill for many corporations to swallow. The second obstacle for SaaS is availability. For mission-critical applications, the network remains a potentially dangerous point of failure.[22]

[22] Service level agreements (SLAs) should be in place to ensure your applications are available when needed.

Mashups are a natural complement to SaaS. Perhaps there are SaaS solutions that appeal to your organization, but you have held back on implementing them because you couldn't get exactly the functionality you required from a single provider. Maybe the SaaS product is extensible, but you don't want to invest time and money in duplicating functionality you've already built internally. Mashup patterns such as Workflow (see Chapter 5) and Content Integration (see Chapter 6) can be used to link an external solution and internal products together. With SaaS and mashups, you may be able to maintain the bulk of your confidential data internally and send the hosted application only small subsets of data for processing. If the network link to the SaaS vendor fails, at least you will still have local access to your data.

If you're thinking about testing the SaaS waters as a *vendor,* then applying SOA via mashups can help you get started. The API Enabler (see Chapter 4) and Quick Proof-of-Concept (see Chapter 7) patterns are excellent means of creating a Web interface to your existing resources. You can use the Load Testing pattern (see Chapter 8) to see how your systems scale under heavy user activity.

SaaS shares another characteristic with mashups: It may already be in use in your company without your knowledge. Because this model requires only a Web browser and no special infrastructure, it is easy for end users to circumvent IT and obtain applications directly. It is crucial that an IT department doesn't have a monitoring and enforcement policy based solely on policing internal data centers. IT personnel need to engage with the business users and educate them about the risks and rewards of SaaS and the effects these decisions will have on future growth. Internal checkpoints with purchasing and legal departments are a necessity, too. All service level agreements (SLAs) should be reviewed and signed by appropriate parties, and attempts to expense software purchases that have not been vetted by IT should raise a warning flag. Otherwise, SaaS can sneak into your organization on a corporate credit card.

Mashups and the User

Make no mistake about it—despite the recent buzz around Enterprise 2.0, people have been creating mashups for many years. Of course, the process to this point has been overwhelmingly manual. Microsoft Excel is arguably the father of the corporate data mashup. For years, Excel end users have cut-and-pasted data to feed their calculation engines. Spreadsheet-based solutions have spread throughout the enterprise without the involvement of IT. Mashup tools enable the automation of this aggregation process, and a new clan of users is poised to run wild with the technology.

A culture of individualism is clearly emerging in today's world. People no longer plan their evenings around what TV networks schedule for them to watch, for example. Instead, they record their favorite shows onto digital video recorders (DVRs) or watch movies and shows on their computers and mobile devices. Similarly, the recording industry no longer has a stranglehold over music distribution. Newspaper readership is down, as more individuals choose to consult RSS feeds and blogs instead of purchasing the printed documents. People can even create personalized clothing and sneakers online.[23] Members of the public have evolved from docile consumers into "prosumers."[24] Products and services are moving away from mass markets and being shaped by the people who consume them. Likewise, a fundamental shift has occurred in software development. Armed with new tools and the skills to use them, users aren't waiting for IT to build solutions—they're doing it themselves.

Should organizations facilitate these individuals' efforts, or rein them in? For years, the mantra of professional software development was "Separate business logic from presentation logic." Programmers religiously structured their code around that principle but ignored the logical conclusion: *The best shepherd of business expertise is not the IT department, but the business users themselves.*

The inclination for IT departments to view user-led efforts in an adversarial light increases when IT experts believe that their "home turf"—application development—is threatened. IT needs the occasional reminder that in any development effort, it is the users who are the key to defining metrics for success. Besides, users are already creating mashups anyway, albeit human-_powered ones.

Gartner has said mashups will make IT even more critical to business operations,[25] so a knee-jerk rejection to their emergence is not necessarily in the best interests of the firm. Rather than deny business users the tools that can increase their productivity, IT needs to embrace a new model. Besides, starting with a mashup product won't get you a business solution any more than staring at a word processor will get you the next great novel.[26] Because IT personnel clearly cannot scale to meet the requirements of each particular user, they should leverage the potential of mashups and work in partnership with the business associates to train a new class of self-serve builders. This

[23] Nike iD lets you design custom shoes and clothing (http://nikeid.nike.com).

[24] Toffler, Alvin. *The Third Wave.* 1980.

[25] David Cearley, Gartner analyst.

[26] Or a *Mashup Patterns* book—trust me, I've tried.

effort is akin to adding hundreds of new developers at virtually no additional cost.

It's a common assumption that the latest generation of developers is intuitively suited to filling this role. Affectionately termed the "Millennials" or "Generation Y," these individuals came of age during the Internet boom of the last decade and are inherently comfortable with technology. Millennials, green with inexperience and giddy about tinkering, question everything. This behavior stands in stark contrast to that of the entrenched workforce, whose habits of working in a particular manner condition them to no longer question the "why."

Many companies are rushing to embrace Web 2.0 ideals such as mashups, social networks, wikis, and blogs not because they have inherent value, but rather because the firms think this practice will attract the "new thinkers." In reality, instead of abdicating responsibility for innovation to a younger generation or applying technology Band-Aids, firms need to cultivate an environment of creativity and collaboration for their employees regardless of their physical age. Any firm can realize the value of mashups and Enterprise 2.0 so long as its managers are capable of taking a critical look at their workplace and realizing they don't need to settle for "good enough" any more.

The "guerrilla-style" approach of mashup development is not without its drawbacks, of course. Most business users do not fully grasp the challenges in providing scalability, reliability, business continuity, disaster recovery, security, and fault tolerance. If users are permitted to develop ad hoc solutions, IT must provide an environment that cultivates these best practices.

A Patterns Primer

The benefits of enterprise mashups are communicated through a concept known as a *pattern.* If you've ever baked holiday cookies, then you already have some idea of what a pattern is and how it works. Suppose you want to make a tray of chocolate-chip heart-shaped cookies. After you mix the dough, you roll it out and grab your cookie cutter. You use the cutter to press out a series of identical shapes. Afterward, you decide some oatmeal raisin hearts would be nice, so you mix a fresh batch of ingredients and cut out another series of hearts. The cookie cutter is a form of pattern. The different types of dough are the specific situations, or "use cases," where the pattern is applied. A pattern doesn't solve a problem in itself. It's just a general form that helps you think about the structure of the solution (what shaped cookie, in this example).

The remaining chapters of this book present a number of patterns, along with some examples to illustrate how they work in an enterprise context. Don’t throw out the pattern if you don’t like the dough! Every business has a different flavor, and the key to success with patterns is figuring out which one is yours. You can use the samples that fill out this book to help identify the mashup ingredients your organization already has. Apply the appropriate mashup pattern and you have a recipe for success.[27]

[27] The classic reference for pattern-based design is Christopher Alexander’s seminal text *The Timeless Way of Building* (Oxford Press, 1979). Buildings, like software components and cooking ingredients, can be combined in an almost endless variety. Nevertheless, certain basic concepts govern which elements work well together and which don’t.

Mashup Patterns: Designs and Examples for the Modern Enterprise by Michael Ogrinz; ISBN 032157947x, Copyright 2009 Pearson Education, Inc.

The Fragility Factor

It may seem that the title of this book is an oxymoron. How can something as ad hoc and unstructured as Web scraping be coupled with something so formal and structured as a pattern? Ideally, the previous discussion of how mashups work under the hood will have made you more comfortable with the technology.

If you think reverse-engineering Web pages still doesn’t sound like the type of rock-solid approach that a professional developer should be using, I don’t blame you. One of the core tenets of software engineering is that applications should behave in a reliable and predictable manner. Web harvesting—although a great deal more reliable than screen scraping—is inherently unstable if you don’t control the Web sites from which you extract data. Because you can’t determine when a scrape-based solution might break, you should never employ this approach on a mission-critical system.

If you have the chance to help your firm gain a competitive advantage or reduce costs—even if just for a limited time—you should explore the opportunity. *There is nothing wrong* with an application that has a short lifespan, so long as you don’t create a situation where the cost of remediating or retiring the solution exceeds the achieved benefit. The rapid speed with which mashups can be developed means occasional remediation isn’t a time-consuming task. Plus, quick release cycles translate into more chances for exploratory development, which in turn can lead to the discovery of new uses or solutions.

The patterns in this book all adhere to this basic premise. You won’t find examples of settling stock trades or sending online payments, even though mashups can facilitate those tasks. It’s simply irresponsible to use the technology in this manner. Like any development effort, a mashup solution will require regular maintenance over its lifetime. Unlike with traditional applications, you may not be able to determine the time when this work will be required. Web Service APIs can change, RSS feeds can be restructured, or site redesigns may temporarily toss a monkey-wrench into your application’s internal workings. Because of these possibilities, you should implement mashup-based solutions only where you can tolerate temporary downtime that may occur at unexpected intervals.

The fragility score is an ad hoc[28] rating based on a number of factors:

- A mashup pattern that relies on a single Web site (e.g., Infinite Monkeys, Time Series, Feed Factory, Accessibility, API Enabler, Filter, Field Medic) is less fragile because there is only a single point of potential failure.
- A multisite-based pattern (e.g., Workflow, Super Search, Location Mapping, Content Migration) is more fragile with each additional site that it leverages.
- Mashups that employ Web harvesting are generally more fragile than those that use feeds (RSS, Atom). Feeds are, in turn, more fragile than Web Service APIs. APIs are the most stable integration point because they reflect a site's commitment to expose data and functionality.
- Mashups that mine data from "hobby" sources have a greater risk of failing. For example, obtaining local weather data from the U.S. government-funded National Oceanic and Atmospheric Administration's (NOAA) weather site (http://www.nws.noaa.gov/) is probably a safer bet than obtaining the information from your local high school or radio station. For-profit sites may exert legal pressure to halt mashups (see the Sticky Fingers anti-pattern).
- Mashups that use boutique data not widely available on the Internet are at high risk. What are your alternatives if the site suddenly vanishes one day?

Each pattern template described in this book contains a fragility score ranging from 1 glass (the least fragile) to 5 glasses (the most fragile). No pattern receives a score of zero, because even the most rigorously tested mashup-backed application always has some degree of brittleness.

[28] Translation: "Your mileage may vary." The fragility score is based on unpublished observations of the technology and will vary according to the resources you incorporate in your specific implementations.

The fragility score is ultimately intended to encourage *thought* about mashup stability. It's possible to have five sites in a multisite pattern that change less frequently than an individual Web page used in a single-site pattern. This is particularly true when vendor products and internally created systems are involved. The user interfaces of commercial and in-house applications aren't frequently redesigned. Public Web sites, in contrast, must constantly reinvent themselves in the battle to attract eyeballs.

If you create a mashup-based solution and don't acknowledge that it encapsulates some degree of uncertainty, you are just kidding yourself. Worse, you are deceiving your users, who will not be pleased when the system "mysteriously" fails one day.

In case you think only mashups have this Achilles' heel, keep in mind that any distributed system (which is what a mashup is) contains an inherent level of risk. Each additional component and the infrastructure that connects it represent another potential point of failure. So before you think, "Why the heck would I build something that might break?" consider how you have handled similar situations in the past. You can address many of these fragility issues by thinking about redundancy, monitoring, and notification up front.

The Future of Mashups

> Mashups aren't just about mixing Web sites together to create new solutions—they're a tool for unlocking the treasure chest of data right under your nose.

The primary goal of this book is for the reader to scan at least one pattern and realize, "I never thought you could do that!" The examples that accompany the patterns are aimed at both the business end user and the technical user. When you understand how mashups can be used to mine new information or automate traditionally manual activities, you'll never look at your workplace in quite the same way. The morass of daily problems suddenly becomes visible—but now you'll have the inspiration and knowledge to tackle them. As with the classic *Design Patterns* text, *Mashup Patterns* is intended to provide a general language that developers and the business can use to succinctly communicate about a solution ("Oh, we can use a Time Series mashup here").

It's not every day that we witness a groundbreaking advancement in application development. Most improvements occur gradually and can take years to snowball into something useful. They may require costly new investments in infrastructure or reengineering of existing resources. Or they may be confined to a narrow niche in our industry. Only the naive overlook the dangers that come with any great leap; only the foolish cite those risks as reason enough to ignore the potential benefits.

Mashup Patterns: Designs and Examples for the Modern Enterprise by Michael Ogrinz; ISBN 032157947x, Copyright 2009 Pearson Education, Inc.

Don’t let the hype surrounding mashups cause you to abandon the best practices that guide good development. Likewise, be open to thinking creatively about the problems that exist around you. Employees who face seemingly intractable problems or whose careers have trained them to ignore the breakdowns in their organization will be delighted to discover that practical solutions are now available. The patterns in this book will help you get started by demonstrating how mashups can help you achieve the following goals:

- Make money for your organization
- Fill gaps not met by the existing IT infrastructure
- Create a quick proof-of-concept to explore new solutions
- Gain a competitive advantage
- Avoid “information overload”
- Expose your applications to a wider audience

and more!

Enterprise 2.0 is all about You. And the potential benefits from mashups are as big as anything you can imagine.

1. From Tim Berners-Lee's first public Usenet post announcing the public availability of the first Web server and browser in 1991.

2. A contingent of which I am proud to proclaim myself a member.

3. The NeXT workstation, conceived by computer luminary Steve Jobs.

4. Tim Berners-Lee invented the World Wide Web in 1989 while working at the CERN Particle Physics Laboratory.

5. NCSA Mosaic, released in 1993.

6. Time magazine, December 13, 2006.

7. http://radar.oreilly.com/archives/2006/12/web-20-compact-definition-tryi.html

8. When Gmail (Google Mail) was announced in April 2004, it offered 1 gigabyte of message storage. This was well beyond the storage limit most corporate mail systems impose on their employees.

9. McAfee, Andrew. "Enterprise 2.0: The Damn of Emergent Collaboration." Sloan Management Review, Vol. 47, Spring 2006.

10. For example, putting a formal development process with relevant checkpoints and milestones in place.

11. Berners-Lee, Tim, James Hendler, and Ora Lassila. "The Semantic Web." Scientific American, May 17, 2001.

12. This naturally presents potential legal complications, as discussed in Chapter 10

13. http://housingmaps.com

14. Iyer, Bala, and Thomas H. Davenport. "Reverse Engineering Google's Innovation Machine." Harvard Business Review, April 2008.

15. Such as an IBM 3270 or VT220.

16. Anderson, Chris. "The Long Tail." Wired, October 2004.

17. The Pareto principle is based on empirical observation and isn't a mathematical certainty in all cases.

18. Sisyphus was a Greek who was condemned by the gods to ceaselessly roll a rock to the top of a mountain, only to have it fall back of its own weight.

19. Problems include interoperability issues and platform-specific implementation, testing, and security challenges.

20. Author Thomas Erl has written several good books on this subject, including SOA Design Patterns.

21. These databases include relational databases, message queues, and data warehouses.

22. Service level agreements (SLAs) should be in place to ensure your applications are available when needed.

23. Nike iD lets you design custom shoes and clothing (http://nikeid.nike.com).

24. Toffler, Alvin. The Third Wave. 1980.

25. David Cearley, Gartner analyst.

26. Or a Mashup Patterns book—trust me, I've tried.

27. The classic reference for pattern-based design is Christopher Alexander's

seminal text The Timeless Way of Building (Oxford Press, 1979). Buildings, like software components and cooking ingredients, can be combined in an almost endless variety. Nevertheless, certain basic concepts govern which elements work well together and which don’t.

28. Translation: “Your mileage may vary.” The fragility score is based on unpublished observations of the technology and will vary according to the resources you incorporate in your specific implementations.