O'REILLY RADAR

Web 2.0
Principles and Best Practices

John Musser

with Tim O'Reilly
& the O'Reilly Radar Team

O'REILLY® radar

Contents

O'REILLY radar

O'Reilly Media Inc.

Web 2.0 Principles and Best Practices, Fall 2006

Introduction

In 2004, we realized that the Web was on the cusp of a new era, one that would finally let loose the power of network effects, setting off a surge of innovation and opportunity. To help usher in this new era, O'Reilly Media and CMP launched a conference that showcased the innovators who were driving it. When O'Reilly's Dale Dougherty came up with the term "Web 2.0" during a brainstorming session, we knew we had the name for the conference. What we didn't know was that the industry would embrace the Web 2.0 meme and that it would come to represent the new Web.

Web 2.0 is much more than just pasting a new user interface onto an old application. It's a way of thinking, a new perspective on the entire business of software—from concept through delivery, from marketing through support. Web 2.0 thrives on network effects: databases that get richer the more people interact with them, applications that are smarter the more people use them, marketing that is driven by user stories and experiences, and applications that interact with each other to form a broader computing platform.

The trend toward networked applications is accelerating. While Web 2.0 has initially taken hold in consumer-facing applications, the infrastructure required to build these applications, and the scale at which they are operating, means that, much as PCs took over from mainframes in a classic demonstration of Clayton Christensen's "innovator's dilemma" hypothesis, web applications can and will move into the enterprise space.

Two years ago we launched the Web 2.0 Conference to evangelize Web 2.0 and to get the industry to take notice of the seismic shift we were experiencing. This report is for those who are ready to respond to that shift. It digs beneath the hype and buzzwords, and teaches the underlying rules of Web 2.0—what they are, how successful Web 2.0 companies are applying them, and how to apply them to your own business. It's a practical resource that provides essential tools for competing and thriving in today's emerging business world. I hope it inspires you to embrace the Web 2.0 opportunity.

—Tim O'Reilly, Fall 2006

Executive Summary

Web 2.0 is a set of economic, social, and technology trends that collectively form the basis for the next generation of the Internet—a more mature, distinctive medium characterized by user participation, openness, and network effects.

Web 2.0 is here today, yet its vast disruptive impact is just beginning. More than just the latest technology buzzword, it's a transformative force that's propelling companies across all industries toward a new way of doing business. Those who act on the Web 2.0 opportunity stand to gain an early-mover advantage in their markets.

O'Reilly Media has identified eight core patterns that are keys to understanding and navigating the Web 2.0 era. This report details the problems each pattern solves or opportunities it creates, and provides a thorough analysis of market trends, proven best practices, case studies of industry leaders, and tools for hands-on self-assessment. To compete and thrive in today's Web 2.0 world, technology decision-makers—including executives, product strategists, entrepreneurs, and thought leaders—need to act now, before the market settles into a new equilibrium. This report shows you how.

What's causing this change? Consider the following raw demographic and technological drivers:

- One billion people around the globe now have access to the Internet
- Mobile devices outnumber desktop computers by a factor of two
- Nearly 50 percent of all U.S. Internet access is now via always-on broadband connections

Combine drivers with the fundamental laws of social networks and lessons from the Web's first decade, and:

- In the first quarter of 2006, MySpace.com signed up 280,000 new users each day and had the second most Internet traffic
- By the second quarter of 2006, 50 million blogs were created—new ones were added at a rate of two per second
- In 2005, eBay conducted 8 billion API-based web services transactions

These trends manifest themselves under a variety of guises, names, and technologies: social computing, user-generated content, software as a service, podcasting, blogs, and the read–write web. Taken together, they are Web 2.0, the next-generation, user-driven, intelligent web. This report is a guide to understanding the principles of Web 2.0 today, providing you with the information and tools you need to implement Web 2.0 concepts in your own products and organization.

Perpetual Beta

When devices and programs are connected to the Internet, applications are no longer software artifacts, they are ongoing services. This has significant impact on the entire software development and delivery process. Therefore, don't package up new features into monolithic releases, but instead add features on a regular basis as part of the normal user experience. Engage your users to be real-time testers, and structure the service to reveal how people use your product.

Overview: End of the Software Adoption Cycle

"What version of Google is this?" Millions of customers use Google's software every day yet never have cause to ask this question. Why? Because In the Internet era, users think in terms of services not packaged software, and they expect these services to just be there and to improve over time. No versions, no installations, no upgrades needed. The traditional design-develop-test-ship-install cycle of packaged software is ending. Software has become a service—a service that is always on, always improving (see Figure 33).

Figure 33: *Examples of beta services*

For development organizations, this shift impacts the entire software development and delivery process. Success now relies on adoption of the perpetual beta development model in which software is continuously refined and improved, users become co-developers, and operations—the daily care and feeding of online services—become a core competency. It is Web Development 2.0.

Benefits

- Faster time to market
- Reduced risk
- Closer relationship with customers
- Real-time data to make quantifiable decisions
- Increased responsiveness

Best Practices

- **Release early and release often.** This edict of the open source development model[61] is now a critical success factor for Internet-based software. Use agile and iterative development methodologies to package bug fixes and enhancements into incremental releases that respond to user feedback. Use automated testing and a rigorous build and deploy process to streamline QA and release management. eBay deploys a new version of its service approximately every two weeks. Flickr photo-sharing service took this even further, deploying hundreds of incremental releases during an 18 month period from February 2004 through August 2005. Compare this with the traditional product release cycle as exemplified by Microsoft Windows (see Figure 34).

  It's not just new products that can benefit from this approach: Yahoo! Messenger went from 1 release every 18 months to 4 releases per year.[62]

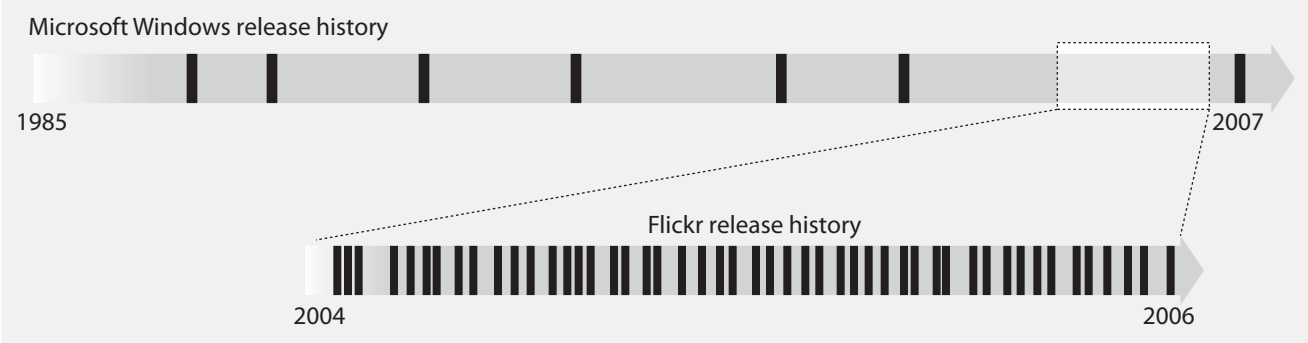

Figure 34: *Flickr versus Microsoft release cycles*

- **Engage users as co-developers and real-time testers.** Real-world user behavior provides a much more accurate model for assessing new product features than marketing requirements documents, prototypes, or any other form of non-production feedback. The nature of web-based applications and the creator's ability to actively monitor how the software is used in the wild is a dramatic shift from the days of desktop software. Use statistics and controlled experimentation to make informed product decisions. Establish feedback models such as dynamic A/B testing in which a small percentage of your site visitors are presented with alternative features and experiences. Amazon.com runs multiple A/B feature tests on its live site every day. The results of these tests feed a rigorous data-driven process that spurs evolution of not only the application but the business as well.

- **Instrument your product.** In the development process, you need to plan for and implement not only the customer-facing application but also a framework for capturing how customers are using your product. What users do often tells you more than what they say. This framework of instrumentation must be guided by business objectives and be as carefully planned for and thought through as the product itself. As with A/B testing, the data captured must answer specific questions as a means for measuring how well objectives are being met and driving product development (see Figure 35).

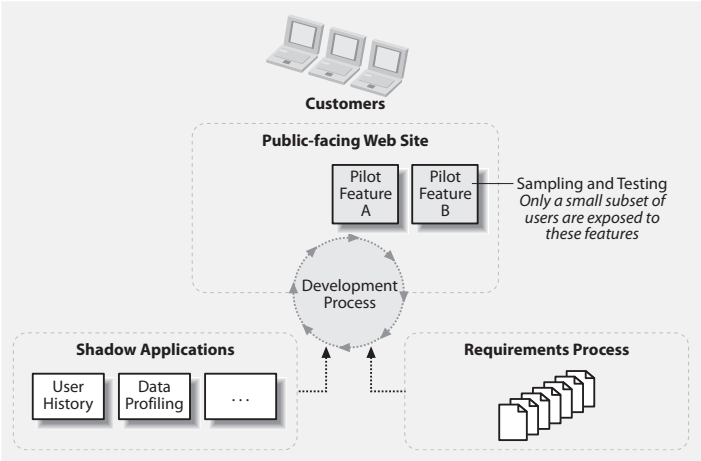

Figure 35: *Perpetual beta product cycle*

Shadow Applications

Shadow applications are private, internal-facing tools built to monitor and profile public-facing applications. They spot what is or isn't succeeding and ultimately drive improvements. Shadow apps don't have to be large, just meaningful. For example, Flickr developed a "Loneliest Users" report that allowed it to identify users who were not inviting friends to the service. Flickr then added itself as a contact for those users and taught them how to make better use of the service.

- **Incrementally create new products.** New and existing products should evolve through rapid releases, user feedback, and instrumentation. Experiment with new product ideas through planned, but incremental processes. Google has launched some of its most successful products including Google Maps and GMail following this approach. The Google Maps beta was publicly launched in February 2005 and stayed in beta for eight months. During that time, Google gained significant feedback from users, incrementally added new features, and gained valuable early-mover advantage, which put it far ahead of slower competitors like Microsoft and Yahoo! (see Figure 36).

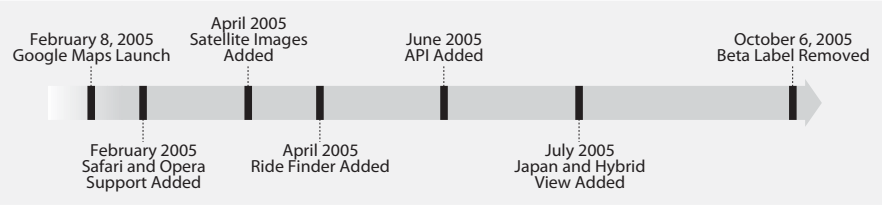

Figure 36: *Google Maps beta timeline*

- **Make operations a core competency.** When software is an always-available online service, it is no longer just software development that determines success, it's operations—that day-to-day ongoing management of data and services. Google's success is due not just to its patented PageRank search algorithms but how well it builds and runs its data centers. Doing this well creates competitive significant cost and quality advantages. These operational strategies and competencies include:
  - — Using horizontal scaling techniques and commodity hardware components for simplified fault-tolerance and high availability
  - — Using low-cost software (typically open source) to leverage large support communities and resources
  - — Ensuring that adequate systems monitoring and management is in place
  - — Ensuring that operations planning and staffing are first-class priorities
  - — Feeding lessons learned from operational experience back into the core product—features, stability, and scalability

  At an application level, this means no longer having the development team throwing it "over the wall" to operations and forgetting about it—they must actively integrate deployment, data management, feedback loops, and metrics.

- **Use dynamic tools and languages.** Rapid release cycles and agile, responsive development models benefit from appropriately flexible development tools and languages. Employ platform-independent, dynamic languages such as Python, PHP, and Ruby to enable adaptability to change, speed, and productivity. Consider development frameworks that focus on simplification and productivity, such as Ruby on Rails (initially created as part of 37signals' Basecamp and later released as open source) or Django for Python (developed as part of the project Ellington and also released as open source code). 37signals often notes how the strengths of the Ruby programming language helped enable it to build Basecamp in four months with a team of 2.5 people.[63]

Misconceptions

- **User testing replaces quality assurance.** Do not use the perpetual beta as an excuse for poor quality, stability, or a lack of accountability. This risks alienating and losing valuable customers. Engaging users as real-time testers is about validating and refining functionality, not quality.
- **Versions no longer exist.** Users may no longer be aware of versions but underneath the covers they are as vital as ever. Some companies with extremely short development cycles "ship timestamps, not versions," yet source code control is used for both. Development tools need to support high-quality rapid software development; the more frequent release cycles require disciplined build, deployment, and support processes.

Issues & Debates

- **Beware of excess.** Just because you can quickly deliver new features to users does not mean you should. Avoid creating confusion or feature fatigue with your customers.
- **Beware of release thrashing.** Rapid release cycles quickly become counterproductive and inefficient if not supported by appropriate internal tools and processes.
- **Uptime is not cheap or easy.** Do not underestimate the cost and effort necessary to achieve high levels of service availability (e.g., "five nines"). As seen with Salesforce.com's high-profile reliability issues,[64] any service-quality failures can lead to customer- and public-relations challenges. Because every application has its own level of criticality—an air traffic control system and an in-house collaboration tool are quite different—so look to match service-level requirements to needs.
- **Privacy.** Instrumentation of applications and profiling user behavior must be done within appropriate privacy and security guidelines.
- **First impressions.** There is always tension between the desire to release a product early and the reality of making a good first impression. This requires rigorous focus on feature prioritization—understanding what's most important—as well as ensuring that what is released is adequately functional and reliable.

Enterprise 2.0 Recommendations

- **Seek suitable enterprise process models.** Look for development and operational models that suit your organization's culture but move toward the perpetual beta. On the development side use agile, iterative approaches. On the operations side, consider best practice-centered models, such as the IT Infrastructure Library (ITIL).[65]
- **Start with pilot projects.** As with any new approach, begin with select projects and teams to learn adoption processes.

Related Patterns

- **Lightweight Models and Cost-Effective Scalability.** Agile software-development techniques are ideally suited to support rapid release cycles, so they have a readiness for change. Integrate lightweight development and deployment processes as complements to the perpetual beta. Combine this with low-cost, commodity components to build a scalable, fault-tolerant operational base.
- **Innovation in Assembly.** The perpetual beta is the process underlying the development of the web platform and it relies on many of the same core competencies.