THE EXPERT'S VOICE®

Pro Website Development and Operations

Streamlining development and operations for large-scale websites

BEST PRACTICES FOR DEVELOPERS TO INTEGRATE WITH OPERATIONS ENGINEERS TO PRODUCE BETTER CODE AND DEPLOY HIGHER QUALITY SOFTWARE PROJECTS FASTER

Matthew Sacks

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

CHAPTER

1

DevOps Principles for Successful Web Sites

Because this is a book about web development *and* operations, you need to know more than just how to build web sites. You also have to understand how teams within a company interact, and which best practices it's important to follow. The interaction of software engineers and system administrators has given rise to the term *developer operations* or *DevOps*, because of the way these teams cross each other's boundaries to work as a single logical unit. I thought this would be the best subject to start off with, because without some kind of union or partnership between these two groups (and others, for that matter), you aren't going to get very far, particularly if you're building a large, complex site.

A healthy flow of information between web developers and operations engineers is crucial to establishing a solid foundation for any web site team. Most modern sites that have some advanced functionality are composed of different layers of complex software that may require different technical skills in each area of the Web stack—the layers of technology that make up a web site. That kind of complexity requires collaboration and interaction. Too often, however, engineers rely on computerized modes of communications, or they simply sit at their desks doing the same thing day in and day out.

It's important to encourage active collaboration between operations and development people. Toward that end, it's a good idea to come up with a set of principles they can follow. Here are some guidelines that may help increase collaboration between software development and operations teams:

- **Collaborate in person:** Get out of your seat and talk to the other operations engineers or developers face to face. There is something you can't get from an e-mail or a phone call that you can from in-person communication. Think of trying to have a party with friends over the phone. Okay, now go talk to someone about the next project, problem, or solution you have to deal with as a team.
- **Walk in their shoes:** If you really get to know and understand the tools and daily processes of the software developers or operations engineers, you'll be better equipped and more likely to find common ground for working better together. For example, if you're an operations engineer and you haven't taken the time to understand the source code management system, and the development folks are adamant about using git over Subversion, it pays to understand why they've taken this position. And it pays even further to learn such systems as much as time allows, because you'll be able to better apply your skills to support these systems or help build tools and processes that support software development.
- **Work for each other:** Make each other's lives easier. Build tools for operations and operations will build tools for you. As Tom Limoncelli, author of *Time Management for System Administrators* (O'Reilly Media, 2005), said, "We are all programmers now." Even so, we have complementary skill sets. In no way is everyone good at everything (although some folks might like to think so), so create a new tool that will help automate a process for your operations engineer or software developer. It doesn't even have to be part of the production systems, and could even be a simple tool for the local desktop. This kind of "tool exchange" helps boost productivity, and it also strengthens bonds and enhances the collaboration efforts among teams.

These essential principles apply both to companies with large development and operations teams, and to small startups as well. They form the basis of this chapter and are the guiding rules underlying this book. The chapter also contains a number of interviews that will shine some light on the various roles of software developers and operations engineers to bring into focus the interaction between the two.

A Closer Look at WebDevOps

Operations has its roots in the industrial revolution where factories began to take on the bulk of the work in producing goods. Today, operations is the application of resources (capital, materials, technology, and human skills and knowledge) to the production of both goods and services. Software development, on the other hand is more akin to the manufacturing process. Sysadmins and software engineers generally have been siloed in their respective departments instead of working as unit as was the case in traditional manufacturing.

In an organization that does business online, the software development department builds applications to power some kind of consumer-facing or business-supporting web site or service. Meanwhile, the operations team monitors and maintains those applications, to keep them running and serving business functions. For the most part, Web developers and operations staff

interact only during releases or when a problem arises that requires both groups to resolve. Today, however, as the number of web applications being developed continually increases and competitiveness among businesses requires that applications be immediately deployed into production for consumers (rather than into retail boxes as in the past), it's more important than ever for both groups to share a common skill set.

This has been happening since the creation of the Web. Tim Berners-Lee stated that when he created the Web, its main goal was to enhance communication through shared knowledge with collaboration as a driving force: "By building a hypertext Web, a group of people of whatever size could easily express themselves, quickly acquire and convey knowledge, overcome misunderstandings, and reduce duplication of effort" (*Weaving the Web*, HarperCollins, 1999). The DevOps idea is rooted in these core principles to this day, but with a more focused emphasis on developers and operations working together and using automation and tools to drive a cultural shift to produce and improve software at an intensified rate.

'The ideas mentioned throughout this book on how software developers and operations engineers can work better together can be applied throughout the entire organization. For example, most principles I outline in this book can also be applied to interactions between operations engineers and marketing, say, or between development and executive management or quality assurance. To keep things simple, I have focused primarily on the interactions between operations and software development teams.

Since the advent of Agile software development, modern web applications are developed very rapidly in a process that iteratively designs and launches code, lets it fail, and then fixes it rapidly. Agile has stretched boundaries, causing system administration and other operations professionals to ramp up their abilities to troubleshoot application and code issues, working more closely with software development, and essentially becoming more like software engineers themselves. The days of just watching graphs and rebooting the occasional application or web server are over for the system administrator. Now applications are being built and tested continually to keep up with changing business trends, and the operations teams need to understand not only how to write code, but also how the code being passed to them from a development team works, and how it's deployed and managed. Operations must be able to work closely with development to set up such processes, so that development, deployment, and management of web software are fluid. The developer and the operations engineer must be able to work at the same level, without relying on each other as much to accomplish the necessary tasks separately as in previous years, and they must work efficiently to avoid wasting time.

The walls between development and operations have begun to come down as a matter of necessity. Today's software is produced ever more quickly, with many large software organizations releasing daily or even multiple times a day, and a majority releasing bi-weekly or weekly. Cultural changes usually take years, and web development is only about 30 years old. But a web development culture is now beginning to take form with the proliferation of tools that enhance productivity and allow traditionally independent groups to function as one. This cultural change among Web developers had its roots in academia when the Web was born. Agile was the next significant set of "laws" set down to change the way Web applications were built, and DevOps is the important current movement in this cultural shift, based the growing similarities in the goals and activities of developers and operations engineers.

Operations engineers have always been programmers to some degree, although not like software developers who often have a formal background in computer science. Traditionally, operations roles have been essentially apprenticeships, with most working knowledge about managing a large-scale Web environment coming from on-the-job experience.

Operations engineers are now more actively focusing on becoming more like software developers—out of necessity. An operations engineer who needs to support a competitive,

face-paced software development culture needs to understand developer tools and practices, such as continuous integration, testing, and building their own tools. The current trend is that software engineers are less likely to adopt practices and processes that operations engineers have built over the years from their on-the-job experience. Without the apprentice-like background of operations, software developers are less inclined to adopt the practices of configuration management, automation, monitoring, and performance testing that are common among operations engineers' daily responsibilities.

Software developers are usually busy building software, which makes it more difficult to learn what operations engineers do. A software developer, for example, might be reluctant to learn how to build a script to deploy a new, prerelease environment, because he is focusing on building some new functionality into the application his team is working on. A developer will be less likely to learn the domain-specific language of a configuration management tool. But the developer today needs to be willing to learn some new tools, such as a configuration management tool commonly used by operations. Not only does this have the potential to improve efficiency between the two groups, it also gives the software developer a different perspective on the configuration management tool itself works, leading to a better end result in how it might be implemented for the software architecture in a given environment. Without this initial understanding on *both* sides of the equation, progress will evolve at a slower pace.

Clearly, the more software developers and operations engineers learn about each other's areas of expertise, the more likely they are to develop a shared perspective on what needs to be done and how to do it. Here are some common high-level topics that developers should study:

- Operating systems
- Network architecture
- Network security
- Web application security
- Configuration management
- Automation practices

On the other side, operations engineers who want to work more closely with web developers will need to understand the following in order to build and maintain a complex site with greater efficiency:

- Communications
- Configuration management
- Programming
- Software design and architecture

It's unlikely that the two groups will switch seats and become truly proficient in each other's skills. Web developers love writing code and operations loves to manage the infrastructure as a whole. What does have to happen is a transfer of knowledge through tighter cooperation between operations and development, training, and possibly even combining skill sets by eliminating the operations department altogether in some cases (which may be suitable for smaller organizations). To bridge the gaps, both camps will need to meet each other half-way in terms of skills and collaboration, and then work together once this sharing of knowledge and responsibility has been applied.

Bridging the Gap

Figure 1-1 shows an example of how this collaboration might evolve. The assumption is that some basic levels of automation and package and configuration management have been implemented.

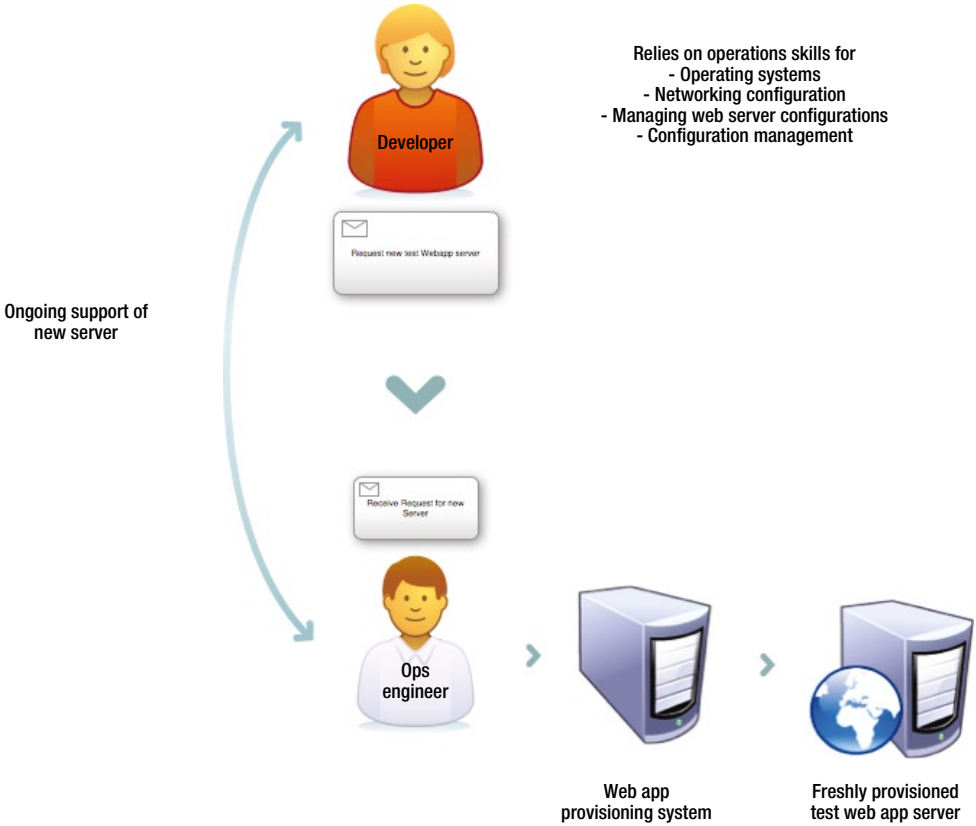


Figure 1-1. *Deploying web code with operations as a gatekeeper*

In Figure 1-1, there is quite a bit of collaboration between the web developer and the operations engineer. This might be because the Web developer doesn't have the knowledge to manage things like Web server configurations, or it might be something as basic as not knowing how to use the command line in the particular operating system. The operations engineer has a system to deploy and provision new application servers, and has probably coordinated quite a bit with the developer to get the application's environment set up correctly so that this "pushbutton" method of delivering Web environments works as desired. This is an excellent advancement for both the web development and operations team in terms of deploying new environments. However, as new environments are pushed out, there is an increasing amount of overhead for operations to track and manage all of the new environments. Unfortunately, ease of use comes at the cost of manageability, and this is exacerbated by the lack of knowledge on the Web developer's part in terms of operating systems, configuration management, and networking, which

puts further pressure on the operations engineer to support these environments. Figure 1-2 shows a more ideal situation between development and operations, in this case provisioning a new Web application server.

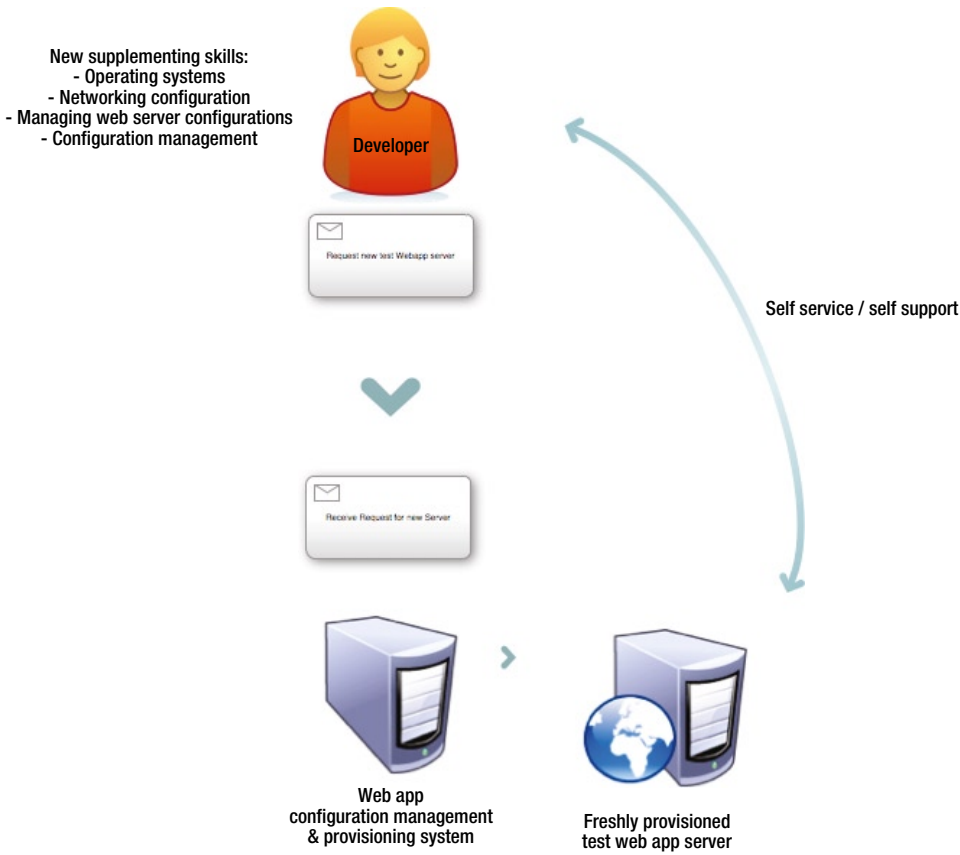


Figure 1-2. *Reduced reliance on operations to build and deploy code*

Figure 1-2 shows that the dependency on the operations department's involvement in the process of deploying and maintaining code and server environments has been significantly reduced.

The automated system for provisioning a new server hasn't changed; the system still responds by provisioning a new web application server for the developer requesting a new server. What has changed is that once the server has been provisioned, the need to interact with operations to make changes to the environment, such as web server configuration changes, logging on to the machine and deploying code to it are all in the developer's field of expertise now. Operations may step in to resolve specific problems that are outside of the developer's expertise, such as changing the configuration management or code deployment software used, but these types of requests become the exception rather than the rule.

Operations has become more adept at understanding developer areas like continuous integration, release management, testing, and debugging source code. Web developers need to become more knowledgeable about operating system internals, networking, configuration management, and automation.

Both developers and operations need to be able to take on each other's roles without any single point of failure in the knowledge necessary to build applications in *either* web development or operations. Trends indicate that the next likely shift is going to be an amalgam of the Web developer and operations engineer with a combined skill set, which will bring the Web development field to the next level.

Taking Output to the Next Level

From the concept of Just In Time manufacturing, a process developed at Toyota Motor Corporation in Japan 1945 by Taiichi Ohno (of which Kanban is a component), the idea arose to rotate workers in positions across the plant so that no worker has only a specific, limited skill set. This both prevents bottlenecks that might impede the flow of the production process and also ensures that workers won't become complacent and their skills will remain sharp. Although the dynamics of industrial manufacturing and software development are different, software development does have its roots in industrial manufacturing processes, and the same principles apply to the DevOps movement. Web developers must assume some of the responsibilities and skills of operations in order to be their most effective and increase their throughput in producing software. Otherwise, the reliance on operations for configuration management, operating system fundamentals, and managing server configuration will impede software production.

Collaboration and using open source software isn't something entirely new; there is just more emphasis on it as Web applications becomes more innately tied to our daily lives, and industries continue to rely on them more and more to function. Tim Berners-Lee describes this activity back in 1999:

> *Making collaboration work is a challenge. It is also fun, because it involves the most grassroots and collegial side of the Web community. All Web code, since my first release in 1991, has been open source software: Anyone can scoop up the source code—the lines of programming—and edit and rebuild them, for free.*

Advancing Collaboration

Open source software plays a large role in advancing web developer and operations collaboration. Many organizations have increasingly begun to either adopt open source software or build software from scratch and then open source it because it means a more streamlined process, less lock-in with vendors, and the ability to customize systems to their needs so the systems can be highly optimized to those needs. Open source software is a perfect match for DevOps practices because proprietary, closed systems don't work as well with rapidly changing environments, especially in the Web world. This has always been the case, and for-profit and not-for-profit businesses alike benefit from open source due to its flexibility and its intrinsic nature as it relates to Web development. The DevOps movement is not limited to Web site development, and is also taking place in traditional areas of software development, such as desktop applications, mobile applications, and enterprise systems. But DevOps does have its roots in

the Web development space because Web software is built and revised at a much faster rate than these other types of software.

DevOps is a recent cultural shift that modifies the interaction between software engineers and operations engineers. As noted earlier, it has its roots in the Agile software development movement, which is based on the Agile Manifesto (agilemanifesto.org). The most recent DevOps emphasis is based on improvements and adoption of Agile practices, such as working software being the measure for progress, collaboration being a focus, and change being embraced. Software is created and released faster than ever, so the need for efficiency and integration between development and operations has become ever more important, giving the DevOps movement traction and visibility.

Traditionally, most relationships between software engineers and operations have taken a "throw it over the wall" approach. Software engineering constantly looks to create new applications and products to meet the demands of business, and operations looks at how to manage and maintain that software in the most stable and risk-averse way possible.

Operations wants mainly to keep services performing, resolving problems as they arise. This focus is why operations can be reluctant to take on change, whereas change is at the core of software development—two different camps with two completely opposing perspectives. Software development promotes change, and it *must* make changes to meet the demands and needs of business. For software engineers, software is a living, breathing thing. Much like a farmer's crops, it requires constant nourishment, judicious pruning, and frequent replanting to provide a sustainable food source; otherwise, the software that provides nutrition to business withers and the business ceases to grow.[1]

On a farm, you plant a seed and watch it grow, and this is actually very similar to developing software. In software development, you start with a base design (the seed), plant it and cultivate it (iterative software development, bug fixes, product lifecycles), and then reap the harvest (keeping a business operating, with revenue and cash flowing). The software developer acts as the farmer, deciding what to plant, how to organize the code architecture (the crops), and making sure that a good yield results. In this example, the operations engineer's role would be similar to that of a farm hand, making sure that the soil is plowed, the soil chemistry is correct, and that the crops get watered. At least that's how things would have been traditionally. Today, the farm hands, or operations engineers, are becoming increasingly involved in working with the farmers, or software developers, to ensure that the farm is properly tended to and that the crops can grow. In *The First Book of Farming* (Doubleday, 1905), Goodrich speaks of "after cultivation." This can be thought of as the software development lifecycle, as there are many similarities between this and farming operations described in his book. He uses the term after-cultivation to refer to those operations performed after the crop is planted. The after-cultivation processes resemble what happens between software developers and operations engineers if you change farm to server farm. The way we cultivate and "work the crop" is changing in that the farm hands now must have knowledge almost equal to the farmers, because the "crop" cycle in a web-based environment happens on almost a daily basis.

In truth, the Web environment works more like a greenhouse than a farm. The gas levels and temperatures in the building must be maintained at just the right level and there is more emphasis and reliance on using automated systems for watering and fertilizing with exact proportions of sunlight and temperatures apportioned to each plant. Making too many changes to this fragile ecosystem (that is, properly running, well-tested production software) could

---

[1]Charles Landon Goodrich, *The First Book of Farming* (New York: Doubleday, Page & Co., 1905), Google eBooks edition, accessed July 15, 2011.

jeopardize the harvest. These walls of the operations greenhouse now need to expand to meet the rapid acceleration of software production. The greenhouse now has to be big enough to put the farm inside. Rather than keeping software developers out, the greenhouse now must invite them in, and coordinate with them the delicate balance of having an infrastructure supportive of rapid change, along with ensuring that the code crops are produced in an efficient manner.

Dealing with Change

With the advent of DevOps, software engineering and operations perform in similar capacities, and their skill sets are more closely aligned. Sysadmins must know how to write applications much as a professional software developer does. The reason for this is that both groups need to speak a common language, and software engineers must understand how operations works and sympathize with the sysadmin's change-averse mentality. Operations must learn how to accept rapid change, and learn how to build systems that accommodate such change by mitigating risk and identifying failures early, rather than trying to impose limits on change.

Traditionally, software engineering and system administrators have been in completely different departments, and had little to no interaction with each other. This goes back to the days of boxed software, when a product was developed using a waterfall approach, and then shipped off to shelves in the form of physical media. Rik Farrow, Editor of the USENIX *;login* magazine, describes the landscape of system administration as it was pre-Web:

> *I worked as part of a team of developers, and my sysadmin work mainly consisted of setting up many different workstations so they could port our software to it. I didn't support any other software packages other than what we got from the workstation vendors.*
>
> *When I did sysadmin consulting, it was pretty much the same. I did work to keep the Unix systems going, and ignored any installed software. As I was a 'hired gun', I was only brought in to solve difficult problems, and they didn't involve commercial software.*
>
> *Having said that, Unix, my focus, was just a useful platform for software that required multiple users or networked collaboration. SCO, for example, was widely used in dentist offices as a turnkey system. They rarely even knew they were using some form of Unix.*
>
> *Databases ran on Unix as well, again with the requirement for multiple users. Another example was publishing software, such as FrameMaker. In neither case was I ever asked what might help make it easier to administer the underlying system. That never even came to my mind. I just tried to keep the systems working reliably.*
>
> *When Windows NT came along, the users of many systems I had worked on fled Unix, believing that Windows would be easier to manage. If 'easier to manage' means GUIs, that is a true statement. The GUIs hid the complexity, but if you needed to do anything not supported by the GUI, you were in trouble. And in the Windows world, to my knowledge, no one consulted the 'workgroup manager' about software design. This would be late 90s, so my experience here is just hearsay.*

> *DevOps is a very new idea. And with the focus on rolling out new versions of software once or twice a week, like Facebook does, the developers had better work with sysadmins or they will face disaster once the new version goes live. In the past, people bought software they would be using for years. I think USENIX database dates from the '90s (scary thing indeed) as an example. With Web apps, usage patterns and functionality of Web applications can change daily, and if the developers make big changes without consulting with the sysadmins, they might find their new system failing because they did not discover if the new changes could be supported. Of course, measured rollouts, as experiments, help here. And that does rely on working with sysadmins.*

The Internet completely changed the way we used software. With the advent of USENET, people began sharing files and information over networked computers. And then the World Wide Web came into being. The Web was the first successful version of software as a service. As the Web matured, standards and browsers began offering more and more functionality that simulated desktop applications. Now with HTML5, CSS3, JavaScript, and Flash, many Web applications have the same features that once could be found only in a desktop application. Every type of application that was once limited to the desktop can now be done in for the Web, including finance, banking, publishing, communications, even graphical editing and design.

What this means to companies whose primary entry point of conducting business is their web site, which is usually an intricate Web application, is that now, instead of boxing and shipping software to hundreds or thousands of customers, and repeating on an annual cycle, changes to the application can happen weekly, daily, or multiple times per day. The rate at which code is produced is much more rapid than it used to be in the 1990s, or even the 2000s. Agile development came along and replaced the waterfall approach of boxed software, which no longer worked for the Web, nowadays the de facto place for conducting commerce and sharing information. Web development teams had to adapt their processes to accommodate the growth of the Web so that releases of new products, in this case new Web sites, could happen whenever necessary to keep up with consumer demand.

The Agile Manifesto states that teams should be self-organizing, and thus begin breaking down the walls between development and operations. Operations and development teams had to begin to work together to achieve the common goal of supporting increasingly larger and more complex Web applications, which included ensuring that systems were consistent, stable, and available. This was a new way of working, but as with any cultural shift, it happens over years, not months.

DevOps represents a new cultural perspective that drives cooperation between developers and operations engineers to support the ever-advancing speed and sophistication of the changes made to web sites and web applications. (Of course, these principles can apply to boxed or non-web-based software as well.) Businesses are now pushing new products at astounding rates and it is the software development teams that foster this growth. Operations has to make sure that systems perform reliably and scale well, and that the software development lifecycle is accelerated as much as possible through the use of automation, configuration management, and other tools and practices.

Looking Ahead

In the future, the role of the operations engineer will be much more like the software development role, though responsibilities will still differ. Previously, an operations engineer didn't have to have as keen of an understanding as a developer of how to manage and automate source

control systems, continuous integration systems, and debugging and testing methods. Their domain was related solely to the operating system, network, and data tiers and the system architecture, and there was little need to work in concert with software development except in the case of building out a new infrastructure or Web application stack. Those days are no more. There may be specialists in certain areas, such as operating systems, networks, and databases or datastores, but these are now part of the knowledge and expertise that all stakeholders must take into account. It is this distribution of knowledge that is crucial to having teams that are able to understand each other's roles in a rapidly changing environment.

The future of DevOps may be that everyone is, in some regard, a software developer, but the operations engineer's focus is more on operating systems, system infrastructure, and networks. The additional knowledge of configuration management, source control systems, release management, and application architecture is crucial in being able to speak a common language and complement each other's roles more closely for achieving optimal efficiency.

However, the blurring of the roles can make the lives of operations engineers and software developers more difficult. As systems are automated, it becomes imperative that both be well-versed in each other's responsibilities. For example, if a developer is debugging a problem in a web application that's running in production on his own local workstation, nothing should prevent him from deploying a fix to a test environment and having an operations engineer approve it for deployment to production. This is how Facebook operates in their daily operations. The way source code moves through the various tiers from test to production, along with the associated approval policies, might vary from organization to organization. Nevertheless, organizations generally all have these processes and the operations engineer should be able to understand what is going on in the code; the developer should be able to push changes to the test environment without approval from operations; and operations should be able to gauge on its own whether the fix should be approved. It is no longer the developer's responsibility to explain this to the operations team, which was generally a time-consuming process that impeded rapid innovation and development. Now it's the operations engineer's responsibility to have the knowledge of the code, application architecture and software development lifecycle so that he can become a conduit for getting changes integrated rather than a roadblock—which, unfortunately, is too often how operations is currently perceived by software development teams.

Insight from the Pros

We now dive into some examples of how software developers relate to operations engineers and vice versa. As we've seen, there used to be little interaction between software developers and system administrators, but that's all changed, especially in web-based companies. Paddy Hannon talks about DevOps and some of the principles discussed earlier in the chapter. Then Tom Limoncelli describes how being in operations has evolved over time, and how it has been affected by Agile and DevOps.

DevOps from a Software Engineer's Perspective

Interview with Paddy Hannon, Vice President, Architecture, Edmunds.com

When software developers and operations engineers work together, neither fully understands what it's like to sit in the other seat and perform the other engineer's job, although the two different positions have many similarities. Software developers build and maintain software, and operations ensure that it runs properly. Developers mostly build software for a customer

or an end user to use, whereas an operations engineer typically builds software that will only be consumed by other engineers within their department or company. However, the demarcation between what a software engineer and an operations engineer can do in code is no longer absolute. The days of the system operator who merely knew how to manipulate configuration settings and maintain a file system are gone. That system operator has been replaced by an engineer who would be equally competent building new applications or working in operations maintaining them and ensuring that they run efficiently.

What did the operations/software developer relationship look like 20 or so years ago in comparison to today (in terms of releases, troubleshooting, and working together)?

When I first started, I was a consultant at a small firm and was responsible for writing code, installing operating systems and software, and managing the database server(s). I had never written code before, but I had experience running Unix workstations so the operations side was, at first, easier for me. At later jobs we did have a more defined operations group; however, they tended to be more focused on networking, OS, and database administrator type responsibilities.

The relationships were always interesting. I remember one senior developer who always had the Unix `w` command running in a window. Whenever he saw one of the admins log into his box to install a patch, he would shut down his network services! He really didn't want anyone messing with what he had. I think the dividing line isn't so much how things were 20 years ago versus how they are now but in the size and complexity of the environment we work in combined with the culture of the company. I have worked at sites with large server farms where the operations team only managed to the OS level and developers handled everything else, and in other places where there was a strict division between what developers do and what operators do.

As a developer, what is your take on the DevOps movement?

In a lot of ways, it could be seen as "devdev." It appears that a lot of former ops responsibilities are being moved to developers. I think it's the right direction. If a developer writes the software, he should to manage it in production—the hand-off to an ops team is costly and prone to error. Eliminating the hand-off eliminates problems, and it makes developers responsible for the software they write. If they get woken up in the middle of the night due to an unforeseen and unmonitored problem, unless they like constant interruptions during their personal time, they will eventually fix the software. If they never feel the pain, they have no incentive, besides being good to their fellow humans, to pay attention to how they build their software. Also, any developer who, for example, only knows Java does not take their craft seriously and is not someone I would hire.

Is the DevOps movement similar to the Agile Movement?

In many ways it is. Agile promotes teamwork and a sense of shared responsibility. Usually people take this to mean developers moving between stories fluidly; however, the Agile literature often mentions QA as part of the fluid nature of an Agile team. DevOps brings operations into the scrum, so to speak.

What is the most important thing a system administrator can provide to a developer?

Access, data, and a stable, homogenous environment. For example, the Hadoop user from machine to machine should have a stable user ID. Treat your infrastructure and configuration the way I expect developers to treat their code, and test your changes using a framework like the Cucumber test framework.

What is the most important thing developers should keep in mind when working with operations?

The developer should remember that 90 percent of the time it's the developers fault if something is broken.

DevOps from an Operations Engineer's Perspective

Interview with Tom Limoncelli, System Administrator, Google
Tom Limoncelli is a system administrator and coauthor of "The Practice of System Administration" and "Time Management for System Administrators.

Limoncelli recounts some of his observations regarding how the working relationship between operations engineers and software developers has changed over time and emphasizes the trend of both software and operations engineers working more closely together and sharing similar responsibilities for producing and maintaining software.

What did the operations/software developer relationship look like 20 or so years ago in comparison to today (in terms of releases, troubleshooting, and working together)?

In the 1980s and 1990s, software was shrink-wrapped. Developers wrote it, tossed it to "manufacturing," which put it on floppies or (later) CD-ROM, and shipped it to stores. The stores sold it to people. If people had problems, they called customer support. Customer service's goal was to prevent customers from talking to the software developers. If sysadmins had a problem with scale (automated installation, or getting a server to handle more users), they had a manual with a few tips, but mostly they were left to reverse-engineer enough about the product so that they could create an ad hoc solution.

- Ratio of developers to customers: 1:many
- Interaction with customers: Forbidden
- Sysadmin tasks: improvised.

How did Agile change the way you interacted with development?

I look at DevOps as the natural response to Agile methods. If a software team becomes more effective because it is doing fast release cycles and greater communication, doesn't it just make sense to start including sysadmins in that process?

There are many similarities between the DevOps movement and the Agile movement. One is the evolution of the other, and DevOps certainly incorporates practices and ideals from the Agile Manifesto, the original manifesto that came about in 2001, when the Web was a mature and established platform. Its main concepts are:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Of course, there is quite a bit more to Agile processes and methodologies than these four principles, and the definitions of Agile and its implementations have grown and evolved since the manifesto's original publication.

Agile practices enabled a new way to produce software, and DevOps takes this a few steps further by building on those practices and then focusing on getting development and operations to work more closely, exchange common skills so both development and operations roles and responsibilities interface (which is now almost everywhere), and promote a culture of change with operations becoming more like software developers in how they resolve problems and how they take part in design and deployment processes.

Summary

Having an understanding of how software development teams have traditionally been organized and what they need to accomplish today is the first step in realizing what is most important in arranging teams to build and support a large scale web site. The roles of software developers and operations engineers are evolving to be much more similar in terms of technical capabilities, and the idea of the DevOps movement is to promote a better understanding of each team's responsibilities and a sharing of crucial knowledge to advance the process of producing software.

Working together is not just something to say to impress management. As Web developers and operations engineers, it's important that we truly understand the systems other teams work with, and that we work together to develop mutually enhancing tools and processes to increase efficiency. Understanding what these skills are and sharing them, or packaging them into tools and sharing those tools, whether they are used in production, or only on the developer workstation, are the kinds of actions that take DevOps from theory into practice.

CHAPTER

2

Aligning Engineering and Business Operations

Traditionally, website engineering teams have been the "back office," having little interaction with executive and business development teams. Too often, those running the business and planning the next phases are unaware of technical and engineering objectives and problems, and don't seem to realize that input from technical departments could help the organization function more efficiently as a whole, and save dollars and effort in achieving its objectives.

Many organizations find it very difficult to align business and engineering or information technology teams because of their very different subcultures and ways of operating. However, such an alignment can be very useful. In a world where a large-scale website is integral to the business, it can increase the flow of information within the company, which benefits everyone.

Creating Symmetry for Engineering and Business

The basic focus of business and marketing groups is to find new customers and generate sales. Engineering teams, in contrast, concentrate on building and supporting technical systems that allow the company to conduct business. Because of their different priorities, their perceptions can diverge wildly. All too often, engineering people think the business teams are thwarting their abilities to build and manage the systems needed to support whatever high-level company goals have been set by executive management, while the business teams view the technical teams as financial sinkholes, especially operations, whose expenditures, they may often feel, limit the growth of the company.

Clearly, in order for the two to work more efficiently together, these two differing camps must come to understand one another. The software engineering teams are most concerned

with how to do things better. With the exception of engineering management, engineering may not always keep in mind the operating costs and sales objectives of the company. It's just not in their frame of reference. The engineer gets paid regardless of meeting sales projections or company objectives.

One the other side, business-focused employees tend to see engineering as more of a manufacturing system than a group charged with solving business problems. This can isolate the engineering teams from company-wide knowledge needed for building functionality into a website that's going to support and advance the company. This sort of problem has existed for years even within technical departments, with operations engineers complaining about software development teams not communicating effectively. And it's equally corrosive and problematic no matter which teams are involved. The disconnect between business management and software engineering tends to be more prominent in larger companies than in startups, where everyone tends to pitch in to accomplish an objective and hopefully establish the company in the marketplace.

Note The term business in this context is used to indicate the departments in a company that have non-software development or engineering roles, such as executive management, product management, marketing, and business development.

Understanding Developer Culture

You can't really think about how to align software engineering teams with other engineering, marketing, product, and executive roles in a company without understanding how they think and work. Whether you refer to these individuals as software developers, software engineers, or programmers, their primary role is building and managing software, though some, such as software architects or performance engineers, might have a more specialized role. Essentially, the three titles can be used interchangeably; they all more or less mean the same thing. To an engineer, however, they may mean completely different things.

Traditionally, software engineers might focus on very complex systems, like building real-time software for jet-engine systems. This type of engineer might become offended if someone calls him a software developer. That someone, of course, would not be incorrect. The only real difference is the specialty of the particular software developer. With regard to Web development, a Web engineer might be writing specifications and protocols for the Web, such as the RDF specification for semantic data in Web pages, or be developing infrastructure for the Web as a whole. There is a certain distinction between engineering and software development, though they are essentially the same thing.

In contrast, Web developers are a breed of engineer who enjoys the creative process of developing a website and thinks about how people access information via the Web. This doesn't mean a Web developer doesn't enjoy Web engineering or architecture; these are not necessarily mutually exclusive. But Web engineers and Web developers often have a separate focus. Both are technical people, however, and technical people usually prefer to use tools and processes to communicate and solve problems, rather than face-to-face interaction. Both generally prefer sending an e-mail or instant message rather than calling someone on the phone.

Business people, on the other hand, understand the importance of face-to-face interaction, and follow-up. They understand how enthusiasm and rapport translate to bringing in new customers and generating sales. Often, their ways of working have little in common with the ways developers work, at times resulting in a cultural clash between the two groups.

So how do two such disparate types understand each other better, then? There are many ways to break down these barriers, of course, but without some effort to align business goals on both sides, interaction between the two groups is not likely to improve. Some people will come out of their shells, others will not. The engineers who come out of their shells and start interacting more with business people are the ones who will become instrumental in helping drive forward the objectives of the company.

Cataloguing Expertise

Web developers might have varying specializations. Some might focus primarily on Web application performance, which means they pay attention to how quickly and efficiently an application loads information from a database, for example. Another might work with embedded systems; this is typically someone who knows the C programming language, or assembly language and is familiar with electronic circuitry on a detailed level. Developers are business resources, and knowing what their areas of expertise are can help the business make optimal use of their abilities. An internal profile of all staff is one way for people to gain an understanding of what everyone does within an organization.

This can be beneficial for the company as a whole. Engineers want to build new things and make things better, and business people want to facilitate ways to increase revenue for the company, increase customer satisfaction, and improve employee productivity. Thus, it's increasingly important for technical and non-technical groups to interact, which can help generate new initiatives and products that can have a serious impact on a company's output. It also improves the general health of an organization because there is a sense of camaraderie and shared objectives between the technical and nontechnical sides of an organization.

In companies with hundreds of people it may be months before a new hire knows who the "go to" person for a particular subject or area of functionality on a large web site. This may lead to a waste of precious time, which could be avoided by having such information easily available. Not having this information available may prevent a connection being made between a business person and an engineer, and it could mean that the loss of a great opportunity for a new product, service, or way of improving things at the company. Having a proper system in place makes it easier for connections to be made, and potentially better productivity and output.

The idea here is a sort of skills catalog for the organization such that individuals can tag their skill sets, much like employment services do to showcase specific skills recruiters are looking for. There may be some talent or skill a person has that could be useful to another group. Having a system where staff fills out a profile or a publicly updateable database directly available to all employees in a company could maximize the use of available skills within a company and make interaction between business and engineering groups more effective. Engineers and business people should not be shy about finding out who can do what within a company, and providing tools that facilitate these interactions will ensure the most skills are leveraged in a company at any given moment.

Talent and Motivation

Motivation is an important factor in employee productivity. Web developers sometimes become entrenched in very monotonous, same thing over and over again kind of work and become bored. This can be especially true in large companies because the ability to do something new or create something new, which is a core attraction for someone who enjoys building things for a living, can be more difficult to accomplish. The more a developer is encouraged to do new

things, the more motivated he'll be, the more motivated his team members will be, and the more likely they they'll be to work diligently to really solve problems the right way the first time, or build new websites and applications. The more motivated teams are the ones most likely to interact with business people to further company objectives; others will simply be tasked labor that will need to be done in order to move forward. This plays into the hiring practices of a company, the available amount of talent in a given area, the available budget allocated for attracting talent, and so on. In most companies, there will always be those who are motivated and are top performers and those who are simply getting the job done. There is really no way to have everyone 100 percent involved and enthusiastic about initiatives for building and rolling out new features for a website, with the exception of startup companies.

One way to ignite, or reignite top talent's motivation is to launch a mini startup within a company. This might include assigning a dedicated team recruited from within the company to build a new site or new features for launch and involve collaboration with a dedicated marketing team as well. This presents a company that isn't afraid to try new ideas to promote the growth of the company. Risk is something that will be carefully balanced within most conservative and larger companies, because the brand identity and reputation of the company are sensitive issues that must be taken into account. But by taking advantage of the excitement a marketing department can generate, a company can spark motivation within engineering teams to undertake a new effort to test and launch new products and features for a website. Such an endeavor, in which inspiration, energy, and new ideas are derived by leveraging the excitement of the marketing engine, can use engineering teams as a catalyst to keep talent within an organization sharp for future objectives or for generating new ideas for company growth and development.

What a Healthy Relationship Between Business and IT Looks Like

As I discussed, it's important for business and development teams to be aligned both in terms of individual goals and business objectives. This is achieved when a company takes advantage of the talents and skills the engineers have to offer. Bringing about the kind of cultural and organizational change that allows this to take place is difficult, but it can be achieved when both groups beginning make a sustained effort to understand each other, something that is not always easy between technical and nontechnical groups.

Business Understands Technical Capabilities

In any company that conducts or promotes business through a website, interaction between business and technical resources will remain paramount. In some organizations with excellent talent, there may be a habit of relying on the engineering team to execute business objectives, but it's important that the relationship be two-way rather than one-way. Most of the time, engineering is not going to begin to make business decisions on behalf of the company, except perhaps in some startups that have limited staff and resources. Instead, engineering teams should know that the business understands *how* engineering can be used to make better business decisions, and that they rely on their engineers to provide supporting data and information about how a website or collection of applications can be used to better achieve business objectives. Most web development and software engineering teams don't think in terms of how their abilities can better support the business.

Engineering Has a Vested Interest in Seeing the Business Succeed

Especially in a mega-company with hundreds or thousands of employees, there might be staleness within engineering teams. This mainly results from the perception these teams have developed over time that the company or the executive management doesn't really understand or care that the engineering teams work day and night to support company objectives. To remedy this, the business has to become interested in engineering, just as engineering has to take an interest in the business.

Project managers are often expected to be a buffer between the business and engineering teams and to translate and relay objectives between teams. This is a terrible mistake, and introduces a bottleneck in the communications process. Project managers are crucial to making sure objectives are met on time; but they should not be solely responsible for transmitting progress reports to the business, or substituting for the involvement of a business person. This is how rifts between business and engineering teams are slowly introduced over time.

Achieving Understanding between Business and IT

The company needs to map software projects to the business objectives those projects support. For example, suppose the company has an initiative to increase advertising revenue by 10 percent within one year. There may be several projects in the software engineering department that are related to this objective, but not everyone in the department may be aware of this. Putting an objective into the context a software engineer can understand drives home the importance of the project an engineer is working on, much more so than simply giving him a deadline and a task.

Business Management Involves IT in Decision-Making Processes

Not everyone is vocal in the Web and software engineering departments. That's why it's crucial that the business side of a company, which may be more adept at interaction, invites the engineering teams into the decision-making process that sets company objectives. In some companies, a board or executive committee makes these decisions, and in others, there's an open forum and collaboration drives the direction of the company. In either scenario, having software engineering teams involved in decision-making initiatives about what projects to focus on in terms of Web or software development efforts, competition in the marketplace, and long-term company objectives, will round out the input into and improve the results that emerge from these standard company practices. This could take the form of a review of the various products or features a company plans to put on its website in the next year, and inviting the various software development and operations teams to provide their input on these projects. This would provide a more realistic view of what is actually achievable. It could be that the operations team informs business that it's already so swamped that trying to put out any new feature would cause the existing features and core business to be neglected. Without the early involvement of the operations or engineering teams, this information might surface only after the project has already been budgeted and put into play. At that point, the engineering teams might feel like they aren't heard and aren't respected and this affects the quality of this new feature, causing it to be a massive disaster, and to be decommissioned later on after not turning a profit.

In contrast, suppose the executive management teams invite the engineering team to review corporate objectives for the next year. Management has done an excellent job of researching the market and determining the potential for growth. Let's say this is a company that allows

homeowners to track their energy usage through intelligent systems in their homes with a website that reports home energy usage. Let's also say that a new open source software package for measuring energy usage has just been released and the web engineering team can effectively leverage this new software to support such an initiative. This allows the company to release a new feature for their existing suite of offerings within a year and it sets them up as a leader in the green software energy space. This new feature is not ground-breaking, but the sentiment that the business and engineering teams are working together to accomplish goals means that if they do decide to go forward on this project, they will have been on the same page since day one and are much more likely be effective when it comes to executing the project.

Common Vocabulary Through Better Tools

Software engineers and operations engineers build systems and tools internally, and business intelligence analysts and engineers look at data and present it to business and management for decision making, but building tools to make the business more effective is something altogether different, and it is one of the most powerful ways of aligning engineering and business operations. For example, when an engineering team takes website hits data and compares it to historical annual revenue data, it can build a graph to show, say, average value per visitor. Building a dashboard that is available on the company intranet that shows the trend in dollars per day that the company is making in relation to website hits is a valuable metric. It indicates how changes in functionality to the site or other efforts, such as strategic partnerships or an SEO initiative, affects the revenue of the company. When engineering teams build such tools for business operations, the relationship between the business and engineering teams is enhanced, and it helps reinforce these two teams working together to meet a common objective.

Building a common vocabulary is another way to enhance the relationships between teams. In this case, vocabulary doesn't necessarily refer to technical terms and acronyms. Instead it has to do with any method, tool, or actual vocabulary that helps build a bridge between two groups that don't typically understand each other.

Dashboards are excellent for creating a common vocabulary between software engineering and business people. Business folks usually speak in terms of revenue, impressions, conversions, leads, and other sales-related metrics, whereas engineering teams talk of technical metrics such as Web site hits, operations per second, response time, round-trip time, and the number of errors per hour. There is not enough time for both teams to learn each other's lingo, ways of operating, or to become immersed in each other's culture, so how can they establish a common vocabulary?

One of the best ways is to produce a graphical description of how technical metrics relate to business-oriented metrics. This could include creating graphs and reports on a periodic basis, but even better would be to develop tools that automatically present a picture of the most interesting data sets to the business. For example, knowing which pages of a site or application generate the most revenue can help align business and engineering efforts. Often, once business sees *how* engineering teams can empower them with information, both groups become motivated by the success of the effort and an alliance and sense of community is strengthened.

Effectively Meeting Deadlines

Unrealistic deadlines are a common problem, especially in an agile development environment. There may be multiple deadlines with multiple products, and multiple conflicting responsibilities for developers, all of which are colliding simultaneously. This can result when there is

fragmentation on the business planning side. Each business department may be in charge of a separate product, feature, or market focus and are all competing for resources, causing an overload on the engineering side and perhaps a lack of understanding about which objectives are a priority. When everything is top priority, there's no way to effectively meet deadlines.

One way to resolve this problem is for executives to consult with engineering and project management teams collectively to determine the level of effort and resources necessary to attain certain objectives. The executives should then take this information back to the upper echelons of the company and assess the associated risk of each objective, project, or feature, and compare that to the available resources within the company. Overworking employees makes them less effective over time. Ensuring that only the projects and initiatives that the business is most confident will produce a positive return or effect on the company, without burning out staff, is the best way to ensure that deadlines are met. And it's important to build some slack into the development schedule, to take into account unexpected changes in the market, economy, or within the company itself.

Letting Steam out of the Pressure Cooker

When engineering teams feel respected by the business, without being pressured to meet unreasonable deadlines, this creates an environment that fosters success. When tackling complicated objectives, such as how to make a new web application or feature a success or support revenue-generating initiatives from the business side, such an environment makes it easier for the engineering teams to focus on the quality of their work and think more clearly about problems before they are worked on. It also makes them more receptive to input, and more interested in the wider objectives of the business, because they actually have time to think of long-term objectives instead of just "fighting fires."

There will always be things that come up unexpectedly, but balancing available manpower, being flexible with planned objectives such as release schedules, and building in slack time in case an emergency arises or a last-minute objective is handed down from management, will keep engineering staff feeling like they can meet attainable goals, which boosts productivity over the long term.

Greater Sense of Empowerment on the Business Side

Once the business and engineering teams have developed some rapport through mutual involvement, building tools and sharing information to gain common understanding and open the flow of communication, a sense of empowerment will emerge on the business side. When the business is more mindful of how it interacts with engineering teams, a sense of mutual respect and connectedness develops, and this will be reflected in greater customer satisfaction, reduced complaints, and more frequent releases of new products and further development of existing services and offerings. You can always tell when a company has achieved this sort of synergy because there is an air of electricity and collaboration within the company walls.

The Enemy Within

Though it's much better for a company when business and engineering teams begin to speak the same language, better understand each other, and develop a healthy coexistence within an organization, it doesn't always work out that way. Sometimes, business management doesn't

play nice, especially those in executive roles who have a lot of sway. I will cover some of the common ways those in charge undermine the alignment of business and technical teams, and what you can do about it.

Know the Lay of the Land

Becoming familiar with the culture and environment of a company and how it operates is beneficial before trying to make any changes, even if you are a director or supervisor. This is because trying to change things when you are new and unfamiliar with the people and how things are currently done will likely just cause backlash and rebellion, which you may never be able to recover from. This goes for procedural as well as technical changes. You may come in and say, "We can't use technology X. I've used technology Y for years and it always works better." This may be true and may even mean cost savings and other benefits, but without knowing why technology X is being used in the first place, you have no legitimate basis for proposing change.

You also need to know how people work. Is the environment a constrained one where code takes months to deploy, releases are tightly controlled, and new features to the website don't roll out very often? If that's so, you'll need to bear it in mind. Trying to facilitate change in this kind of environment might be impossible. You might want to research this in advance before starting out on a new development project or even before joining a new company.

Knowing your manager is also important before trying to propose changes. Some managers will never pass your recommendations, suggestions, and reports up the chain for a number of reasons:

- You might be more talented than your and have a chance at taking his job.
- The manager simply doesn't like you or favors someone else to make a star.
- The manager doesn't know a good engineering recommendation when he sees one.

Some companies have an open forum for facilitating innovation. These companies tend to do well and their employees and engineers have a feeling of participating in the company, rather than being a worker bee hired just to do a job and without an active part in the success of a company. These are important aspects to understand before trying to align business management and engineering teams. There simply may be no realistic way to do it. Where you are in the organization also makes a difference. If you are a director or manager, you may want to consider how you can start creating a culture of collective feedback and discussion. Such mechanisms help shape and define the success of a company and its development efforts, and this is reflected in the quality of the employees' work.

Making Suggestions to Executives Can Be Difficult

Depending on your role within the organization, you may not have enough rank to bubble suggestions up to the upper echelons of a company. This means you will have to channel suggestions up to your manager and hope your relationship with him is good enough that he will take them seriously and pass them along. In some instances, executive management makes it virtually impossible for suggestions and change to be made. For a Web developer, these kinds of

environments should be avoided if at all possible. Even in cases where an improvement *could* be made, or where the engineering staff at least feels it is being heard and taken into account, it's not easy for the engineers to suggest a new feature, product, or change, without feeling like they are overstepping their bounds or trying to one up their manager. If you're in an environment where useful suggestion and the desire to improve a company's website and product offerings are not encouraged or taken seriously, my advice would be to leave such an environment for a more rewarding experience. Nobody likes to feel like he doesn't matter and isn't heard, and when it's your job as an engineer to improve things, being in a work environment that discourages making suggestions can damage your desire and gusto to do so.

Override

A common practice is that the board or executives of a company or organization leaves it up to a technical director to represent the technical voice of the organization, and to make all of the technical decisions. This is a massive flaw because there may be concerns and valuable input down at the lowest rungs of the organization that are not being heard that might benefit everyone. The CTO might be making decisions on behalf of the engineering teams that go against the majority opinion regarding a technical issue.

For example, let's consider the use of virtualization in production. Many would say that you can never use virtualization in production. The engineering team may refute this, having spent months researching it and finding it would result in cost savings, better performance, and easier manageability of the server farm in the future. However, the CTO can simply "disagree" with the evidence presented, and refuse to use virtualization in production based on superstition or personal opinion, and executives will never know about this errant behavior.

Executives are an important part of any organization. They are in charge of leading an organization and its employees, defining the direction of the company, and if the company fails, it's essentially on them. One of the mistakes that the employees below the executive management tier make is not questioning executive management. This has a lot to do with the corporate culture of an organization. If there is a history of people getting fired for disagreeing with executives, few people risk bringing something up, even if it would, in fact, be beneficial to the company. When bringing something up that should be bubbled up to the executive tier, make sure to go through the proper channels. When it comes to large-scale websites, there is often a huge disconnect between the business side of a company, whose primary mode of generating sales is through the website, and project management and engineering teams. In such environments, it becomes very difficult and time-consuming to make changes that might otherwise a benefit a company. Ultimately, it's the owners and shareholders of a company who make the majority of decisions, because without revenue, there is no job for the engineer or anyone else. This doesn't mean, however, that business owners should be driving engineering and technical decision making. Sometimes you might need to venture into uncharted territory making technical recommendations.

Suppose an engineer spends three months testing, trying to prove that virtualization can be used in a production environment. That engineer then provides all the results of the testing as evidence that virtualization can very satisfactorily serve a production website. She also collects facts and evidence about banks and other mission critical websites and applications running virtualization in production and offers that data and research to support this claim. An executive manager then reviews this information and decides that virtualization will not be used in production as a companywide policy, in spite of the evidence, cost savings, and efficiency gains, because he doesn't like it as a matter of personal preference. This kind of situation is not uncommon, and after taking the time and initiative to perform all of these tests and research to find out how to improve the performance and functionality of a website, an engineer might become

quite discouraged, and rightly so in such an environment. In such a situation, it would be best to leave the company and go somewhere else where a culture of feedback, open discussion, and the willingness to try something new when hard evidence is presented is appreciated. Some companies are just unwilling to take a chance with something new when nothing is actually broken. Such companies usually go out of business in the Web world because on the Web, innovation is a competitive advantage, and staying the same is a surefire way to have people stop visiting your site. The same principle applies to software. Software that doesn't change or is never improved generally goes stale and is deprecated in favor of newer, more current systems.

Improving Communication Between Business and Engineering

We've seen some of the ways executive management can block change, and even suggestions about change, related to innovation and improvement of a website. Corporate culture, comfort in doing things the usual way, and neglect are all ways to prevent the alignment of business and engineering teams.

However, a company whose executive management encourages employees to take an active role in the planning of the organization through regular meetings and communications will find that the staff is more than willing to provide suggestions and recommendations. For web development and engineering efforts to succeed, management has to be clear and straightforward about what it is requesting.

Define and Execute

In what I call a "define and execute," scenario, the business owners and executives define what they would like the engineering teams to produce. This might include an entirely new website with new branding under a parent company, a new feature, or a redesign of an existing brand. The executive team might lay out a one- to five-year plan of the high-level objectives for the site, such as "build a data warehouse of user activity on the site that can be updated on demand for business analytics," and ask the engineering teams meet those goals,. This initiative might give business management the ability to look at what is going on in the organization in near real time to facilitate selling and purchasing decisions or reach out to potential suppliers, advertisers, or the like to make more competitive decisions about the business. This type of goal would be a massive project for the engineering teams and, in this case, suppose executive management is requiring that the engineering team "just do it." There is no discussion, and the feedback presented by the engineering team to executive management would be in the context of this one high-level goal. Suggesting something such as, "We don't need a data warehouse on user activity because we have this third-party service that does that for us" would be a possible recommendation back to the executive management team about this initiative.

Full Circle

A full-circle feedback environment is a more collaborative and open approach. Not all companies would adopt such a platform, and it might not work for these kinds of companies. But companies that adapt to change succeed, and the companies that listen to feedback from all levels

are the ones that will adapt more quickly. A website is much like a living, breathing organism and must be cared for as such. That means paying attention to the engineering team's feedback and recommendations about the website in a Web-based business.

Small issues and concerns might not be apparent to the upper echelons of a company. This could mean that a great idea or research and development project might never come to fruition, and it could have meant revenue or increased market share for a company. Having a suggestion box on the corporate intranet not only allows engineering teams to provide input, it also lets them blow off steam. An anonymous feedback mechanism like this gives a voice to teams that otherwise might not voice their opinions. These kinds of tools facilitate open discussion and when employed specifically between engineering and business operations, they can be the first step in establishing communication. Many suggestions may never be heard or acted upon, while others may initiate cultural and procedural changes across a company; the important point is that providing asynchronous, anonymous feedback mechanisms between departments will promote an environment of collaboration between engineering and business, rather than the isolation that impedes the process of innovation and successful development.

Summary

Web developers, operations, and other technical professionals don't always have an understanding of how their counterparts on the business side work, and vice versa. If these two groups can begin to work more closely together and learn how both sides work, the result will be beneficial to the entire company.

It's best when both groups figure out where their talents intersect, and then nurture those relationships and skills. For example, engineers can create dashboards that empower business users, and business users can include engineers in organizational and product planning. The success of an endeavor, indeed of the entire company, is much more likely in an environment that promotes real interaction among all teams.

CHAPTER

3

Web Testing Practices

Testing a web application requires not only testing the site itself, but also looking at the various application metrics at every layer of the stack. It's like building an aircraft: each part of the aircraft has to be engineered and tested for safety before it is made a part of the whole. Once each subsystem has been developed and tested, they can all be assembled into the finished product for a test flight. With such a complex system, it only makes sense to be sure you can trust the individual parts before you assume the finished product will get you off the ground.

A website is similar. It too is made up of various components and subsystems, such as the network, database, application logic, and front end, arranged into layers or tiers, and may even have multiple systems interacting at each tier. In general, here are the steps to follow to test a website:

1. Decide where to test (i.e., which layers to test).
2. Identify which metrics are of interest to the business and engineering teams, respectively, before testing them.
3. Implement different, separate testing techniques for each tier.
4. Test the entire Website as a whole.

Following these steps will make it easier to pinpoint problems in the site when errors occur or performance is less than desirable.

It may sound simplistic, but you need to know what to look for before implementing any kind of testing methodology. This requires understanding the nature of the business and what metrics are of importance before setting up and scheduling the tests and allocating testing resources. You have to consider which test cases will yield the most value in the least amount of time. Ideally, all software should be tested, but it's not always practical to test every part of the software. We test because we want to improve our software, and simply testing everything wastes time and money and is counterproductive.

It can be very helpful to review historical data from testing phases and cycles on a tier-by-tier basis. Also, looking at the entire website—exercising the full stack—from the perspective of the end users will reduce the amount of time it takes to identify and locate issues that users are experiencing.

Web software testing is changing quite dramatically nowadays with the advent of Behavior Driven Development (BDD), a method of writing software and unit tests in terms of desired

outcomes or acceptance critera for software to perform a specific goal. But there are many different methodologies that have a place and purpose in Web testing. I've outlined some of these tests and how they apply to various kinds of outcomes.

Web Testing Practices

It's not the responsibility of one department to have the responsibility for ensuring software quality. Web developers, operations engineers, and quality assurance engineers should all have the ability to execute any types of tests, as long as they all use some common tool set to do so; all stakeholders should be involved in assuring the quality of the software. This may require integrating the tests into your testing framework and continuous integration processes, or having some way of automating the testing so that gaining visibility into the web stack or application's performance is a fast, efficient process.

To find out how the software is performing, it's important to run different kinds of tests, including functional tests and stress tests, especially when developing a new application. For well-established applications, you'll want to know how an application has performed historically, which may reduce some of the need for performing more involved stress tests, such as maximum capacity and sustained load tests. By amassing data through gathering baselines about how each tier performs individually, the need to perform these invasive tests at every level, such as the Web, application, and database tier may eventually be reduced or eliminated altogether. These practices can be integrated into the software development life cycle if they are well-automated, or they can be run on a periodic basis.

There is no hard and fast rule for how an organization should test its software, although following accepted practices will help any organization produce better software and reduce the number of errors customers see in production use. The degree, frequency, and level of detail and automation used for each type of testing will, of course, vary depending on the complexity of the web application, the amount of usage the application receives, and how much revenue the application generates for the business. Applications that are complex and highly used, and that generate lots of revenue, will require that web developers and operations engineers work closely together to ensure that high-availability and failover are present in both the application and the infrastructure and are thoroughly tested.

The following rules determine how much testing you need for your Web application:

- Complexity. The more complex the website or application, the more testing it is likely to require. For example, applications that do data mining or financial industry calculations may need more testing than a simple LAMP (Linux Apache MySQL) based Web application, for example. Thoroughly testing a web application that is only going to be used by a small set of internal users requires less rigorous and thorough testing than a website or application that's accessed by millions of users every day. The more use an application receives, the higher the probability that strange edge cases will cause it to fail; this should be investigated before the application is put into production.

- Cost. Even if a website receives a lot of use, it may not necessarily generate a large amount of revenue. Websites and applications that generate millions, hundreds of thousands, or even tens of thousands of dollars per day are generally more important to a company's bottom line and so require more testing due to the financial risk of them being down.

- Culture. If your engineering teams are accustomed to embracing and writing their own tests, you may not need as much testing by another group, such as a dedicated QA department. Still, other departments may have to be involved in some testing.

Maximum-Capacity Testing

Maximum-capacity testing (stress testing) involves pushing a load out to an end-user service to find the point at which a web application or site breaks and stops functioning. This practice is critical to planning for capacity and to determining how much stress or load a given application can handle. This is always a synthetic test performed internally in your test environment. You should never perform a maximum capacity test on a production site because it might cause an outage, impacting the business's revenue. Maximum capacity testing can potentially reveal problems in the code. An application that buckles under a relatively light load or a relatively low number of sessions probably has some major functional problem that needs to be addressed before testing continues.

What it's good for: Every application will break when the number of requests exceeds the available resources and the capabilities of the hardware or, more likely, of the software that's being tested. Maximum-capacity testing can reveal blatant coding errors without requiring heavy investigative work or waiting for a sustained load test to finish. With a problematic application, you'll typically find a large number of errors in the log files after testing, which can be reviewed and then processed by a log analytics engine to reveal the most common errors in the application.

Sustained Load Testing

Sustained load testing (soak testing) is the practice of testing a website or application over a long period of time, under varying types of load. This helps to flush out any issues before pushing code to production. Sustained load testing is typically done only for a major software change or release and can take from 24 hours to a couple of days.

A more aggressive tactic for performing sustained load testing is to put the new code on a small set of servers and offload some traffic to these servers using a load balancer, and then put those servers with the new code into production first to a percenatage of servers. This gives developers and operations engineers a chance to see how the new code is going to perform in production.

For example, 10% of all production traffic should be put into the new codebase to see how it performs without risking running 100% of the new code on production an watching the Website collapse.

What it's good for: Sustained load testing reveals information about how all of the various components of a Web application perform over an extended period of time. This is especially useful for exposing slow memory leaks, time-dependent bugs (those that appear only at a certain time of day, for example), and sometimes finds problems that aren't flushed out by high-load testing. This type of test is important, but limited because of its long running time, which can be problematic, especially on short development cycles.

Behavior Driven Development

End-user metrics provide a picture of how the Website as a whole is functioning, and they help reveal problems quickly and effectively, at which point further and deeper diagnoses can be made. Behavior Driven Development is an agile development technique that incorportates user "stories" into the development process, making it easier to track metrics. An important aspect of BDD is that it encourages collaboration between Web developers and operations professionals and even nontechnical users. By using a BDD framework, such as the Cucumber Test Framework, Web developers, operations and even business users can describe what the software should do in a plain-English format, which is then mapped to programming logic that executes a given test on the Web application.

Aslak Hellesoy is the creator of the Cucumber Test Framework and Senior Software Engineer at DRW Trading Group. Chris Read is a developer and operations engineer at DRW Trading Group and an active contributor to the Cucumber Test Framework. They share their insights about testing in the following passages.

What is the importance of integrating testing into your development process?

Aslak Hellesoy:

Having tests allows us to change the software when we need to. We work in an environment where our customers are traders (of securities) and they demand a lot of features all the time, like several times a day, and we end up having to make a large number of small changes. Testing allow us to make those changes and make sure we haven't broken anything. That said, it all depends on the value of the tests. At some point tests can be hard to write and can become an inconvenience as well. If you have to make changes really quickly to your software stack, it becomes a lot more difficult to have tests for everything, and maintaining those tests can be hard. For example, the previous places I worked put much more emphasis on tests because the software wasn't changing so fast. But on the other hand, we were not able to get feedback very fast either. So, we put a lot of effort into tests. Whereas where I work at now, software changes very fast and we get fast feedback from the users.

Chris Read:

I think what's important to me in TDD (Test Driven Development) and BDD is they help you find that balance that Aslak's talking about. You need to think about *how* to test your code, and also considering *where* to do your testing helps good quality code evolve. You'll be able to very easily test the important bits you need when you have a balance in your tests. For example, if there's a web service you are providing to someone else, it's quite easy and straightforward to think about how people are going to use that web service. And a framework like Cucumber helps you to design a good web service because the very first client for that service is actually going to be your tests. When it comes to thinking about balance, consider whether you really need to test that the background color is green on every single page. Probably you don't. These are the kinds of things you need to think about when balancing your tests.

What is the difference between Test Driven Development and Behavior Driven Development?

Aslak Hellesoy:

BDD came about five years later as a response to TDD because TDD was very hard for people to learn. I think of BDD as several additions and tweaks to TDD. BDD adds tweaks to how you talk about your tests and how you name them; but also the level of understanding required for writing them and who can write them. BDD has evolved over the past six or seven years; eight years

maybe, I think it started in 2003. So now, BDD for me is more about communication between stakeholders, testers, programmers and users.

How does continuous integration and testing come into play in fast-moving web environments; does it always make sense to use it?

Aslak Hellesoy:

We get requests multiple times a day to make changes to the code, and we make changes several times a day and deploy several times a day. In that kind of rapidly changing environment, you don't really need executable specifications because you have other feedback mechanisms that replace what BDD or executable specifications set out to solve. But that doesn't mean executable specifications or BDD or Cucumber or whatever isn't important; it really depends on your environment.

Chris Read:

It all comes down to why write tests in the first place? We write them because we believe in continuous integration. What is continuous integration? Continuous integration is feedback that what I'm doing (a) is what the business is requiring, so it's a feature that is required by users but also (b) that the code I'm writing doesn't break any other features that exist. We write tests to reinforce and ensure that that's not happening and tests provide feedback. And the whole point of continuous integration is providing that feedback cycle to the developers on development cycles of certain lengths. The important thing is the feedback, not how you get it. So, in a slower-moving environments, the unit tests or Cucumber tests—whatever different testing frameworks you have—these are the mechanisms that provide most of that feedback. The difference in the environment we are in now is we get that feedback faster because we are in a very small niche with a very small number of end users. We get that feedback faster and more effectively by deploying to production and getting the feedback directly from the users than by writing a test. It's faster for us to put it up in front of the users and they say, "Oh I need this font tweaked, I need the background color of this cell changed, please." It may be quicker for the developer to just take care of it and push it to production than to spend the extra time writing tests to get that feedback cycle.

I do realize that, you know, many companies and most products don't work that way. That's why I think BDD provides more of an executable specification. I think it provides a lot of value simply because when you can't work as quickly and get that close to the end users, you need some other kind of communication and a way to get feedback, and in that case a BDD framework such as Cucumber makes sense.

Operations as well as development can write tests, and with something like Cucumber, anyone can write a test. Is that kind of how you see tests evolving with the DevOps movement?

Aslak Hellesoy:

At a previous place I worked, some colleagues of mine were using Cucumber on a project for the Norwegian National Dairy or something like that, which was a cow-feeding organization. It was actually quite complicated testing their software because different cows need different feeding routines, and so forth. And there are complex business rules and they were writing a system where they needed to capture all of the business rules somehow. They trained one of the main experts, who wasn't a programmer. She was a business user in her late 50s and they trained her to describe how the software should work, and in Cucumber, actually, she just sat down and wrote it. What she wrote, the developers' then implemented as she described more features about the business rules. It was very easy training her because the tool (Cucumber) is

designed from the ground up to be used by people who are not necessarily programmers. So, I think teaching non-technical people to write executable specifications can work really well for a lot of teams.

Chris Read:

The whole point of Cucumber-Nagios is to take some of the final acceptance tests from a web app if they are written in Cucumber and then run them against the production system (using Nagios) in such a way that the tests define the monitors. Because they are now actually running in the production system, you know the production system is healthy. And, if it is not healthy, the test fails and generates alerts, telling you that something is going on with the production system. That's just one model of, well, the whole test-driven infrastructure. So you take what the developers have been doing for years, which is they've got their code and they test it. Now they have these methodologies around testing and developing their code and we're applying them to operations. Once we take our infrastructure and we treat our infrastructure as code, what we can then do is write tests to describe how the infrastructure needs to look and then make the changes to the infrastructure until those test go green, which then tells us that our infrastructure is healthy.

Automating Web Testing with Santiago Suarez Ordoñez

Web tests might be written by the developer responsible for a given application, or perhaps by a completely different engineer whose role it is to write functional tests. A tool like the Selenium testing framework (seleniumhq.org) can be used even by operations to automate real-browser monitoring or testing. The main point is that testing a web application using a real browser increases the level of accuracy and realism of Web tests, in comparison to a synthetic test using code that merely initiates an HTTP request and doesn't parse and render JavaScript in the browser. People use web browsers, so testing without one doesn't produce the most accurate results

Santiago Suarez Oroñez is a software engineer for Sauce Labs, the sponsoring organization of the Selenium testing framework. Selenium allows a tester to develop test cases using a real browser, which can then be replayed on a real browser to simulate actual usage during the automation of tests. Santiago gives examples of how they might perform Web testing in their software development cycle.

Do you see developers writing their own Selenium tests, or is it primarily QA?

Luckily, most of the times I see the same developers who are building the main application actually writing tests for it, which, from a productivity point of view, is definitely the way to go. In other cases, people specialize and get their own "automation engineer" title. This is not a bad thing, as long as a strong connection with the project development is kept.

Do you see operations using their own Selenium tests?

I do every once in a while. People doing active monitoring with real browsers, testing Flex/Flash applications and doing tasks that would be impossible without it.

Would you say that Selenium is part of acceptance-test-driven development?

I'd say Selenium should be part of every testing cycle, from the first round of tests, to acceptance and post-deploy. The ideal test suite, from my point of view, should be composed of all sorts of tests, from unit-level tests, to functional and end-to-end browser tests. Let's also not forget about the last manual check for fundamental layout and overall quality.

The number of tests of each type should be proportional to the level of specificity: for unit-level tests, there should be tons; for functional tests, several; for end-to-end/integration tests, a good amount; for manual testing, the basic 3 or 5 workflows that go through 80 % of the Web application.

Security as a Testing Practice

Security is an often overlooked aspect of web application testing, but it's critical. It is frequently viewed as a nuisance or unnecessary part of testing, but this is not true. By integrating vulnerability testing into the testing and continuous integration cycle, you'll find not only security holes, but also performance issues, functional problems, and unexpected bugs that would have otherwise never been detected, perhaps even during production use of the application.

Security testing a web application may uncover issues that would not be revealed by real customer interaction with the application or website. Testing often involves integrating a framework such as the various Open Web Application Security Project (OWASP) frameworks, Metasploit, or the Web application Attack and Audit Framework (w3af).

Such frameworks uncover performance issues, bugs, and other problems in addition to revealing security vulnerabilities. By integrating security testing into the testing process, you gain an additional layer of quality assurance.

Deciding Where to Test

Where to test depends on where the web application is currently located in the software development life cycle. In early development stages, you may want to test individual parts of the stack directly, in order to get a better understanding of how each individual component performs. This is the case because in a web application, many layers of the web stack play into the actual serving of a page or a result to an end user. The process typically includes the following layers:

- The front end, where HTML, CSS, and JavaScript are rendered in the browser.
- The Web tier, which is typically a Web server managing access to the Web application via HTTP.
- The application tier, which may be composed of a number of layers of the application tier serving various business logic functions.
- The data tier (or database tier), where actual data is retrieved from some kind of data store:
    - Relational database
    - Key/value store
    - Document-oriented database
    - Data-processing platform such as Apache Hadoop
    - Search servers such as Apache Solr

At the beginning of the project, it may be fruitful to test these various tiers individually, especially when making significant changes to any component of the stack. It then becomes easier to pinpoint problems in the future because of potential differences in performance; the

problem might even exist at the end-user level (a browser request by a user). Without having benchmarks for each individual component, troubleshooting is much more time-consuming because you could find yourself searching the entire web stack to find the culprit when problems with performance and functionality arise.

Metrics Meets Testing: Deciding What to Test

Tests simulate some kind of activity to give the team building the web application information—metrics—about how the application is going to behave or has behaved historically. These metrics are typically quantitative assessments: measurements and instruments and gauges that show how specific aspects or components of the application or systems supporting the application performed *during use*, that is, during the test. In a web application, there can be hundreds or even thousands of different measurements, all of which are probably not going to be relevant for a specific test. The metrics might be part of a fixed monitoring system designed to measure web application metrics or they might be part of a testing tool that specifically measures the response time of a web application, for example.

Metrics, and the dashboards that summarize them, are critical to performance. Performance metrics are ultimately driven by the business, although development and operations will have their own sets of key metrics. The only "important" metrics are those that are relevant to the business that owns the software and the users who use the website or application.

Developers know how the web application is supposed to function at these various key levels. Unless they share this information, operations will not have a clear picture of which metrics it needs to monitor on an ongoing basis. Operations teams tend to monitor as many functions as they can, on the assumption that they might need those details sometime in the future. This is not really good practice for tracking the performance of a website and its supporting applications, because it adds a lot of useless data and should be limited instead to a key set of metrics.

Business Metrics for Websites

Software development is typically an expensive endeavor, so one of the primary goals of any large-scale web project is to generate revenue as well as to provide a service. A web development project must be financially sustainable or it will not succeed. It must take in at least as much money as it expends to create and maintain the project. Given this requirement, the most important metrics being monitored will be business metrics.

Business owners and executive management need information about how a website functions as it relates to their business objectives. They are mainly concerned with concepts such as profits, operating expenses, growth of the business, and customer perception of the business on the Web. Most business owners and executives are preoccupied with these concerns, and aren't interested in the particulars of how the Web infrastructure works or what to look for in terms of a healthy measure. Nor should they be, as they have quite enough to deal with in running the business.

Developers and operations engineers, in contrast, must absolutely be concerned with whatever is important to the business. It is their responsibility to work in concert with the business owners and executives to determine what is of importance in terms of business metrics. They also must communicate which technical metrics would provide data to answer questions executives may have about how the business is performing.

For an organization with a large-scale website doing business via advertising or e-commerce, or whose primary source of revenue is via interaction with a website, one of the most

notable concerns is web page load times. A well-known metric that supports this comes from Greg Linden of Amazon.com, who posited that an increase in 100 ms of load time caused a 1 percent drop in sales on Amazon.com.[1] Of course, web page load times will not have such a dramatic effect on revenue for all businesses, but it is clear that increased load times affect revenue when the primary source of revenue is from a website.

Determining which metrics are important to your organization requires that both developers and operations meet with business owners or executives to figure out which are relevant to the ultimate business goals. Here are some common performance metrics that can be put into a business context:

- Page load times: Correlating a website's page load time with averages for revenue is useful for understand how site performanc affects sales.
- Conversion rate on a page-by-page basis: Finding out which pages promote the desired response or produce the most revenue on a site is useful for learning whether your site meets your business objectives.
- Power draw: This metric tells you the cost of the power required to serve your website. Some code releases may actually result in higher CPU use, meaning more power utilization and higher costs.
- Content delivery network bandwidth utilization: Having a graph or diagram that shows bandwidth use on a CDN and what that means in terms of cost will give a clear insight into value and can be used to compare against the monthly bill from a CDN.

Metrics should be encompassed in an easy-to-use dashboard prepared by the technical team for self-service use by the business. A business person should never have to request agreed-upon metrics from the technical team. These metrics should be available on demand and verified for accuracy on a periodic basis, as errors do happen in reporting of metrics.

Performance testing is a critical part of releasing new applications. Until software has been tested thoroughly from the end user's perspective, both software engineering and operations can have only a limited picture of how their applications will function under different load scenarios (that is, when stress is being put on the application by users).

It is not the responsibility of the quality assurance (QA) department to execute performance tests, although QA may run its own benchmarks and suites of performance tests once web applications have been handed off to them for testing. Performance tests are the responsibility of operations and development, which need to be testing performance constantly; software engineers need to test using microbenchmarks, and operations needs to develop and provide tools for performing these tests.

With a production Website that is being accessed by many users, the only way to measure how the site is doing from the perspective of a real customer rather than a synthetic tool is to monitor it using real browsers, from different Internet service providers, from different locations around the nation or globe, depending on your market. For example, if you conduct business only in the United States and your website is in English, then it makes sense to monitor your site's performance and availability from different corners of the United States rather than the entire world. Customers drive the business, and knowing exactly how your site is performing and being accessed is crucial to the successful operation of the business.

---

[1]Statistics derived from Greg Linden's presentation "Make Data Useful" (http://sites.google.com/site/glinden/Home/StanfordDataMining.2006-11-28.ppt), accessed July 15, 2011.

Setting up a monitoring infrastructure all over the nation or globe to track a website's performance and monitor and report on that performance is complex and expensive, and would probably be prohibitive except for the largest organizations, such as Facebook and Google. An organization would have to first develop a monitoring infrastructure to accomplish this and then have nodes in data centers around the country or the world and manage and maintain that system, which could potentially require as much or more work as managing and maintaining the core Web business. There are many third-party services that perform this function as a service, and it is critical to have such a service to truly understand how an organization's Website is performing on the public Internet. Web developers need to understand how the customer's experience of the site's performance affects the business, so that they can communicate with the business people and collaboratively determine which technical metrics are of the most importance to the business' revenue generating Website.

Ben Rushlo, Director of Performance Consulting for Keynote Systems (keynote.com) works with many large national and multi-national businesses that generate revenue through their websites. He describes what metrics he thinks business should focus on.

Which Web metrics are important to track?

The customer's experience is the most important metric. What I found in doing this for 13 years for Keynote is that businesses are tracking metrics that don't directly relate to the customer's experience; they may have some internal server metric or some other way of showing that they are doing a good job when in reality the customer might be suffering.

All of the business people are tracking conversion and click-through, but they sort of get a little weird when we start saying they should really be thinking about performance just as they do any other business metric.

We try to encourage business people to do this based on the context of their industry, and to make use of studies and best practices to come up with some reasonable performance metrics. A lot of times such metrics have to do with page load time. That's something we'd certainly look at. We'd examine their variability because averages can be very misleading. We also look at how long interacttions take, which is a relevant topic in the industry now, or time to first render. For example, we'd try to assess the customer's experience for questions like, "Can I buy something online?" or "Can I build a car?" or whatever the site is meant to do.

The primary metrics should find out if the customers are getting the experience they're expecting. And then whether we are able to acomplish whatever business goals we have—conversions or marketing or whatever.

Web Application Performance Metrics

The metrics measured at the application and system level will vary depending on the applications and types of systems in the web infrastructure. The main point to keep in mind is to track and trend only the performance metrics that are absolutely critical. It is much easier to add detailed application metrics later, rather than trying to sift through a slew of dashboards to find the specific metric that is important at that moment. At first, though, you might want to adopt a policy of monitoring many more metrics than you normally would on an ongoing basis, and gradually scale down what is monitored to the bare essentials. With a new project, it may be prudent to monitor all or a large set of application metrics. When launching a new application, information may be hidden in uncommonly used metrics that you otherwise might not notice. Once you've been monitoring the full set of metrics for a few months, it will likely become apparent which metrics will be useful and which provide little or no value.

Performance dashboards are monitors that keep track of a given application's or system's metrics at an instrumented level. Most applications expose some kind of information about their internal performance through an API or instrumentation. Application performance metrics are commonly accessed through methods that include SNMP, mod_stats for Apache Web servers, Java Management Extensions (JMX) for Java, the rails_metrics gem for Ruby on Rails, emetric for Erlang applications, and xdebug or dtrace (which can essentially be used with anything) for PHP.

Putting Application-Performance Metric Monitoring into Practice with Metric Profiles

Let's say I want to monitor a Java application server, which powers a web application running an e-commerce site called example.com. The primary functionality of example.com is a shopping cart application that sells products of any type. The first step is determining what the interaction of the user is with the website, and then monitoring and tracking over time the most common indicators of problems this type of application is likely to have. For example, I know that memory usage can be tricky due to the way Java performs. Let's say this is a memory-intensive application, so I definitely want to include the Java memory heap on the list of monitored metrics. Also, because this is a new version of the application, I'll include metrics I don't normally monitor in production, to see if there's any valuable information. Once I've used this metric profile for a month or so, it will become obvious which metrics to continue monitoring because they will be valuable for troubleshooting. Here are the basic steps I'll take:

1. Determine which metrics are of interest. In this case, my initial verbose set of metrics to be monitored are taken from the list of available metrics for my application and are as follows:
    - bytesReceived
    - bytesSent
    - requestCount
    - errorCount
    - loadTime
    - processingTime
    - maxTime
    - cacheSize
    - HeapMemoryUsage
    - currentThreadsBusy
    - currentThreadCount
    - threadBacklog
    - maxThreads

- threadPriority
- CollectionTime
- CollectionCount
- OpenFileDescriptorCount
- CommittedVirtualMemorySize
- TotalSwapSpaceSize
- FreeSwapSpaceSize
- ProcessCpuTime
- SystemLoadAverage
- CurrentThreadCpuTime

2. After monitoring these application metrics for a period of time, I have a good amount of information about how my application performs, collected in various scenarios and load volumes. Now I can launch the application into production with this monitoring profile.
3. After launching the application into production with the verbose monitoring profile, I decide I need only the following metrics, as the other metrics don't yield any useful information about how my application is performing:

    - requestCount
    - errorCount
    - loadTime
    - HeapMemoryUsage
    - currentThreadsBusy
    - currentThreadCount
    - CommittedVirtualMemorySize
    - TotalSwapSpaceSize
    - FreeSwapSpaceSize
    - ProcessCpuTime
    - SystemLoadAverage

■ **Note** After doing verbose monitoring during the testing period for your application, it will become more apparent which metrics to monitor. What any given organization decides to monitor will be based on different testing scenarios and ultimately seeing what information yields useful insights into the application's performance and provides help for troubleshooting during failures. Until this production testing has been completed, it will be difficult for a DevOps organization to say what is a relevant metric to be monitored, unless it already has data about how a given application or type of application performs and functions.

Testing the Individual Components of the Stack for Rapid Troubleshooting

For any website, testing the front-end tier involves simulating a real user performing real functions on the site. This is called real-browser testing and it's different from synthetic testing where a tool such as Apache-Bench or Jmeter is used to simulate traffic load on the front end of a website. A real browser takes a certain amount of time assembling the Document Object Model (DOM), which includes parsing JavaScript, and then rendering the page. Tools like Apache-Bench and Jmeter do not test these aspects of a real browser loading.

There are a number of tools that can programmatically configure and execute real-browser tests on a website, which can be automated and integrated into QA and continuous integration processes. Selenium, which I mentioned earlier, is a web application testing system that includes support for most major browsers, and Watir (`http://watir.com`), which stands for the Web application testing in Ruby, is a Ruby gem library that automates Web browser testing actions.

Testing the web tier directly is more of a synthetic test and gives you an idea of how many sessions will be supported, though it doesn't add a lot of value to understanding how an application will behave for end users. Synthetic tests such as Apache Bench and Jmeter, which can be used for spot testing and smoke testing as well as performance testing, are ideal for doing simple tests such as when tuning the KeepAlive variables on an Apache web server, and for other web server tuning exercises, but shouldn't be relied on for end-user performance metrics of the type Selenium or Watir can provide. The main idea to keep in mind here is that tests performed in controlled environments that simulate some kind of user activity, such as loading a photo album, will not perform the same way as in production. Even though tests done in a staging environment will reveal much about how an application performs or behaves, it is only once the application has been launched into production that its true performance and behavior will be revealed.

Selenium is a framework for automatic functional web application testing using real browsers. It allows you to configure workflows and interactions on a website that simulate what real users do. Selenium also records this interaction in a video, so you can later go back and see whether a test has passed or failed. Because of the overhead involved in launching a real browser, recording the interaction, and repeating that action, this process would require hundreds or

thousands of virtual machines to simulate the simultaneous load of hundreds or thousands of users accessing a site using real browsers, which could be difficult for even large organization to implement. There are services available that perform this activity and have nodes all over the world, such as Keynote Systems, Gomez, and BrowserMob (which uses Selenium), all of which use real browsers for geographically distributed load testing and performance monitoring.

Testing all components of the web stack, regardless of how it's done, is crucial, and it is even more beneficial once you've built up a record for each part of the web application stack, and a record of how the various tiers have behaved over time, so it can be compared to releases or changes in the web stack.

Keeping Historical Performance Data on a Tier-by-Tier Basis

Having historical performance data for every tier that comprises the web stack is crucial for quickly determining the location when a problem arises. A typical three-tier architecture includes a web tier, an application tier, and a data tier, and a performance problem could arise in any of them, often times making it difficult to pinpoint a problem. By keeping performance data about each tier, problems can often be detected and resolved before they are noticed by end users or, more importantly, before they affect the revenue-generating functions of a site or application. Web developers need to work in concert with operations to gain visibility into each tier, although it may be both groups' responsibility to determine *how* each tier will be tested. For example, web developers may be the ones who keep historical performance trending data on the application tier and web tier and have more of a say into how these tiers are tested and what constitutes a test in these tiers. With the database tier, on the other hand, it may be the database administrators who set up a tool or test that exercises certain queries and functions of the database.

For any business that generates revenue or leads through its website, monitoring end user performance is absolutely critical. Not knowing how the site is performing on a national or global level leaves the business without the insight to manage its core business. Simply monitoring end user performance, however, is not enough if you want to be able to diagnose problems quickly and effectively and have more control over the individual tiers or components that affect end user performance.

Figure 3-1 shows a typical three-tier web environment that has a global or regional performance monitoring service in place so the business can keep track of end-user and Web performance metrics. There is only minimal monitoring or no monitoring at all on a tier-by-tier basis of the individual components of the web stack, and when problems are detected at the end-user level with the global monitoring service, the development and operations teams have to scramble and search the logs to figure out where the performance problem resides. In this example, performance degraded for end users when a change was made that affected all three tiers.

The problem could, in fact, be due to external factors, such as a DDoS attack, network or ISP issues, or a sudden increase in visitors to the website. However, because there is no data about how each tier has been performing historically, it will be more difficult and take more time and collaboration to determine where the problem originated.

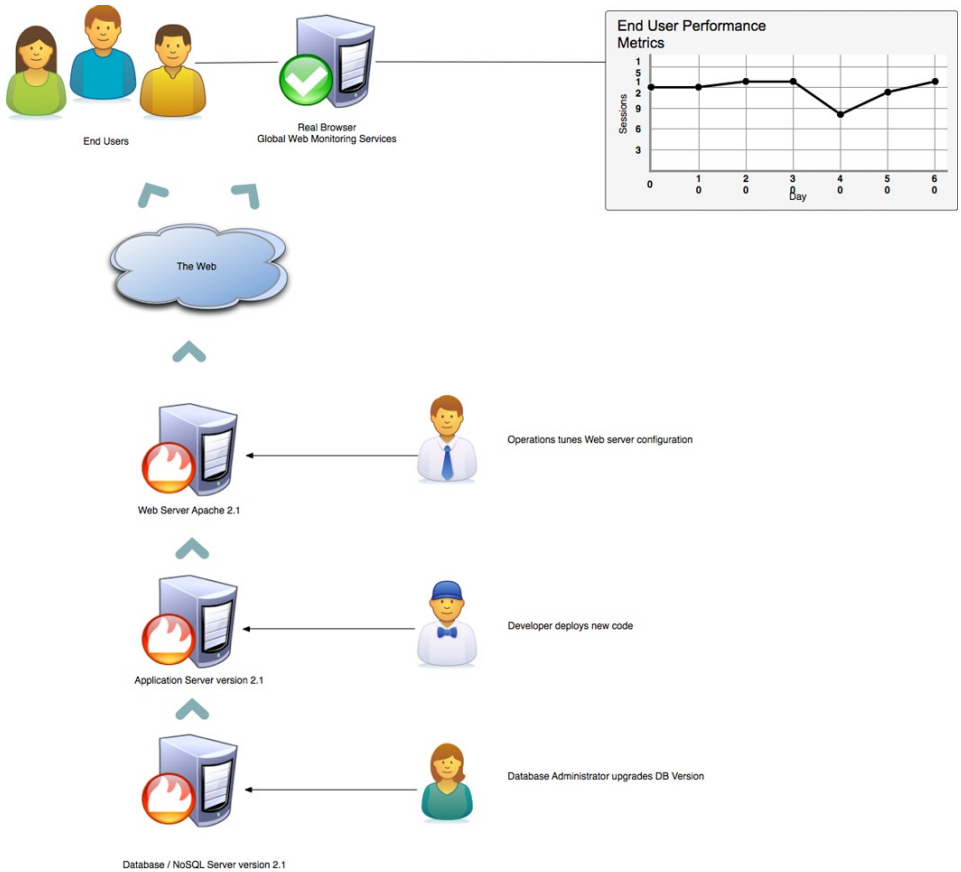


Figure 3-1. *Monitoring only high-level metrics*

Figure 3-2 shows the same environment, where historical performance data now exists for each tier. In this case, if a change made internally results in degraded end-user performance or problems, it can be detected almost immediately. The time taken to troubleshoot problems is significantly reduced because a change in performance will now be detected at a more granular level, and determining where a problem happened will be isolated to one specific tier. This performance data can be compared with changelogs and application log files, and the difficulty in isolating where to look for issues has been virtually eliminated. In addition, when an end user experiences performance degradation, figuring out whether the important factors are internal or external has now been reduced from an all-hands-on-deck situation to a simple glance at a few historical performance graphs.

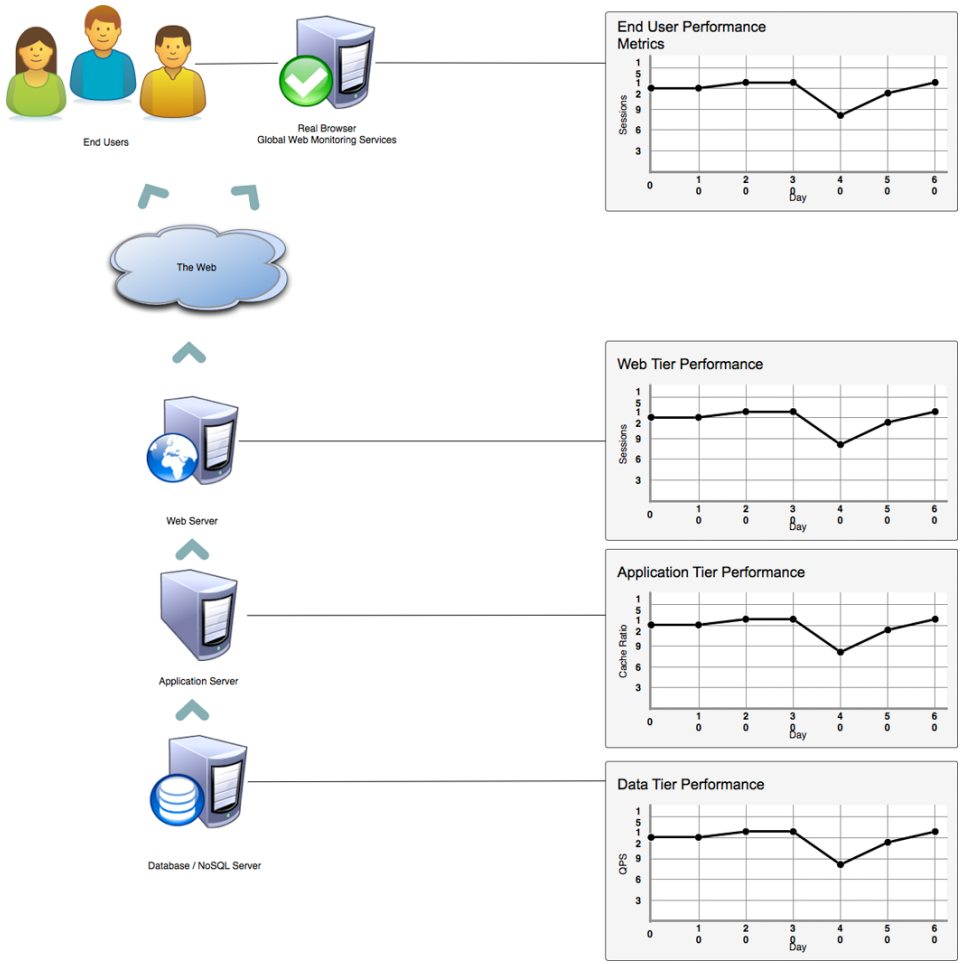


Figure 3-2. *Keeping track of historical data on a tier-by-tier basis makes it much easier to pinpoint problems in the web stack*

Summary

Knowing what to look for when implementing any kind of testing methodology is crucial to producing quality results and improving software. The first step is understanding the nature of the business and what metrics provide the most value. This is key to setting up, scheduling and allocating testing resources, and knowing which test cases will yield the most value and save the most time. Ideally, all software should be tested thoroughly, but even so, knowing the

specific metrics and features of the applications being tested before implementing testing plans and deciding on a methodology will help keep the focus on the desired results. Simply testing everything just wastes time and money, and is counterproductive.

Maintaining historical data from testing phases and cycles, and periodically reviewing the results of tests on a tier-by-tier basis, as well as from the perspective of the end users experience of the entire Website (exercising the full stack) will help reduce the amount of time it takes to identify and pinpoint any issues that arise.

CHAPTER

4

Designing Intelligent Documentation

Most organizations overlook the value of documentation and its role in accelerating and improving the quality of the applications and the efficiency of an operations team. Documentation is commonly viewed as something that slows Web development down. If used intelligently, however, it can be a powerful tool for enhancing the quality of code, the operational processes, and the overall efficiency of information technology teams.

Documentation is a critical part of the software development life cycle. It is just as important as testing and quality assurance. If you can't articulate how your application or operations process works or how someone can replicate the steps you took, other engineers will find it difficult to configure, build, and reproduce the software. Moreover, until you have someone else go through the steps you've documented, you won't find the errors or the parts that make absolutely no sense.

Documentation actually improves the efficiency and quality of output because everyone, not just the technical teams, has a clear picture of how an application's processes, architecture, and user experience all tie together.

In this chapter we'll look at how to create processes for documenting. At the same time, we'll propose a cultural shift in the Web development and operations teams that will mitigate the perception that time spent documenting is better spent coding.

The Unseen Benefits of Documentation

Once a culture of documentation is introduced—gradually— into a Web organization, it becomes a practice others grow to rely on. One significant advantage is that when staff changes through the process of attrition, good documentation makes it easier to bring in new development and operations people without having to put in large amounts of "hand holding" time. It is also another form of review, outside the standard code comments, bug tracking, and source control processes that make up the usual path of software development. Documentation is especially

important in organizations that produce proprietary software, where it can be difficult for newcomers to gain an understanding of complex, custom applications and how they fit together to build what a consumer sees as a Web site. If documentation becomes an integral part of the process, communicating how the software is built will reliably take place, and engineers will become accustomed to the practice of describing how the software works to other team members.

To see this in action, just consider the most successful open source software, which tend to have certain things in common: a thriving community of developers collaborating and communicating freely, and excellent documentation. When developers collaborate to produce excellent documentation designed for a wide variety of audiences, it becomes much easier to build and manage software. This holds true for all organizations, and is especially important for larger organizations.

Later in the chapter, we will examine various templates for creating standard documentation that will help you learn how to manage software built within an organization. For example, a senior software developer might be able to use just the API Specifications and Reference and begin implementing an application without reading the rest of the documentation. On the other hand, Getting Started Guides, which provide more information outlined in a simple way, will help junior software developers to ramp up and quickly gain an understanding of the application, platform, or library they need to work with. Use Case documentation explains to non-technical people what a particular application is used for. We will look at these and other templates in more detail shortly.

By having all of this information readily available, the time required to gain an understanding of complex Web and application architectures will be significantly reduced. Without a complete set of documentation to illustrate the various perspectives on how a given technology works, Web developers and operations engineers alike will be left trying to figure out how things work on their own.

Most importantly, the actual writing of the documentation by various stakeholders will bring to the surface unseen problems and opportunities for improvement. When, for example, an engineer begins to map a Web application on paper or in a design mockup so that others can easily get the concept, the result is much better feedback about the technical capabilities and design of the application. Documentation that targets various kinds of readers is also beneficial. Chances are, a business user won't be able to decipher an API specification, but creating a simple diagram on how complex APIs or applications interact with each other will help a non-technical audience. Such documentation may also make it easier for junior engineering staff to understand complex software architectures.

In this respect, documentation becomes a living feedback mechanism. It becomes a conduit for converting the application into words and visualizations that can be digested much more quickly than if you had to trudge through the code or implement the application. Writing and using documentation is a mechanism for eliciting feedback, discovering problems in code, and generating new ideas for application and process design; it is, in actuality, testing your code and Web applications.

Documentation Roadblocks

Getting developers and engineers to document can be especially difficult, so it's helpful to understand why many of them don't do it. Once you understand some of the common roadblocks that prevent people from integrating documentation into the development process, you'll have a better idea of how to address the common scenarios that keep them from realizing their

maximum efficiency and quality in the applications and Web products they produce. The following scenarios represent some of the excuses engineers offer to avoid writing documentation.

Scenario 1: There's Not Enough Time

The best Web developers always write well-commented, well-documented code, but even they usually don't write documentation for non-developer audiences. The best engineers will be creating applications like mad, and don't want to take the time to write it out or explain it to someone else. They may argue that the code is the documentation and that a technically capable person will be able to understand the code, so it's a waste of time to write additional documentation.

The reality, however, is that not everyone speaks APIs or configuration management manifests. Business users, project managers, and other nontechnical people often need to appreciate the inner workings of the application so they can better do their jobs, and a technical document just isn't going to work. Those ancillary people are critical to the proper functioning of the IT organization, and they may feel alienated and unsure of how to interact with development and operations teams, or how to represent their efforts to others in their business, project management, or other less-technical departments without this information.

Moreover, software development is moving at a faster and faster pace, which means that developers and operations need to produce faster than ever, with less and less time for cross-training others, writing documentation, and being trained themselves. This is a mistake. If you don't document, you are introducing legacy systems into your organization. Legacy systems generally need to be replaced every few years, because the people who knew how to maintain them have either left the organization or simply don't remember how to maintain the systems and have no reference to fall back on.

Solution: Build Documentation into the Success Criteria and Share the Responsibility of Documenting

Most Web-based organizations use some kind of Agile or Lean process to produce their software, and that process usually includes periodic reviews of the work that has been done and checking off tasks to determine how well a project is advancing. Documentation should be part of the success criteria. By doing so, documentation becomes integrated into the culture of the development and operations teams, and the organization will reap the benefits, such as better code and bringing to light deficiencies in design.

This process requires buy-in from management to be successful. If management is purely focused on meeting deadlines, and not on creating a culture of documentation, then such a mandate is not likely to be successful. Simply agreeing that documentation is part of the software development lifecycle and not following up with the necessary tools and time for documenting means that the objective of integrating a culture of documentation will be reached.

Furthermore, both the development and operations teams must become involved in building a culture of documentation, as a group effort rather than the responsibility of just a few individuals. In very large organizations, it may help to have a full-time technical writer with a background in development or operations. A writer with a technical background is less likely to be looked on as someone who exists just makes the lives of the engineers more difficult.

Benefit: Accountability

Having a technical writer on staff helps encourage the development and operations teams to be responsible for documenting their own code, applications, and processes. An experienced technical writer can coach those who don't know how to write effective documentation or simply aren't in the habit of doing so. This can help documentation-averse engineers become comfortable with writing documentation.

Once individuals take ownership of their own documentation, the technical writer can switch roles from evangelizer and writer to editor and instructor, with the responsibility of making sure that documentation is organized and accessible, and helping the engineers improve the quality of their technical writing. Of course, the responsibilities of a technical writer will differ in different shops, depending on how a Web business is organized, but by serving as the point person for the documentation, he or she can help ensure that the documentation actually gets done.

Scenario 2: There's Only Technical Documentation

Not everyone in the organization is a senior software engineer or system administrator, and having only specification-based documentation means only highly technical people can read it. This is a recipe for dysfunction, with only senior technical members having any visibility into what is going on with the applications. Of course, it's good and necessary to have this documentation, far better than simply relying on picking through source code to figure out what's happening with your application. However, it does not work as a complete documentation strategy and culture.

Solution: Targeting Documentation for Your Audience

How a particular type of documentation will be interpreted depends on an individual's position within the organization. For example, an API reference has no value to system administrators, even senior system administrators. It's unlikely they'll take the time to figure out an API reference or even to ask how they might interpret or use the API reference to improve their operations processes. What system administrators need instead is a document written with the context of the system administrator in mind. Such a document may inherit much information from the API reference, but rather than containing just a list of functions it should include information about how many requests the API can serve, what network protocol it uses, and any software dependencies it might require. This enables the system administrator to provide information about how to deploy the application, along with any caveats and gotchas about the server environment in which it will be deployed. In this case, we started with an API reference and then came up with two specific documents about the *implementation* of that API for two different audiences: a guide for operations and a guide for development.

By having a complete set of documentation for a variety of audiences, documentation will become a living part of the culture of a team. It's important to understand who will need to rely on documentation, such as business users, system administrators, database administrators, software developers, network engineers, project managers, just to name a few. Perhaps the API specification needs to be put in terms of costs per application to support, for a business user; or which protocols an application uses, for a network engineer. There's no formula for what type of documentation should be produced; it depends on the needs of the business and the teams.

Benefit: Strengthening Bonds between Disparate Teams

Creating documentation for a variety of audiences results in greater understanding on both sides of the transaction, and fewer misunderstandings and errors, reducing stress levels on both sides. Moreover, documentation can be built from other documentation. For example, once you have information about an application's capabilities and limits, and the capabilities and limits of the operations infrastructure (servers, network gear, and so forth), you can put the information into the context of maintenance and capacity planning costs and scalability metrics. One document can beget the other, and the time required to produce successive documents is decreased because each document type is built upon the other. This doesn't work in all cases, but it does hold true in many.

Scenario 3: Documentation Quickly Becomes Outdated

Even in organizations with robust documentation and a healthy documentation culture, applications evolve and change much quicker than documentation does. The simple fact is that software moves much faster than documentation, and even with automated documentation systems, you still may sometimes need a human being to write out documentation for people to read, and this kind of documentation, which takes technology and puts it into a readable context for a particular kind of audience, is not maintainable by automated methods.

API references and other highly technical documents will most likely be automatically documented as part of the programming language's platforms (JavaDoc for Java, RDoc for Ruby, Pydoc for Python, and so forth). However, documents that require human expression, the most valuable form of communication, will not have as close a relationship to the application and infrastructure that the technical documentation does. For this reason, documentation may become dated in as little as a month, or even less in some cases, and some documents may not be updated for years. This is actually worse than having no documentation because instead of having to ask for information, engineers may think they have the correct information when they are in fact following out-of-date instructions.

In addition, once documentation is created, many engineers will not think to go back and update it. Having a documentation review on a periodic schedule will help keep documentation up to date.

Solution: Notify Engineers to Update Documentation

It's unlikely that anyone is going to browse through all of the documents in a document-management system, such as a wiki, for an entire organization, although some pruning and editing may occur on a departmental level. But a random approach is extremely inefficient; there has to be some kind of mechanism whereby the document management system notifies an owner or administrator that a document has not been updated in a defined period of time. A solution has been proposed for the TWiki open source wiki that pages should yellow like a newspaper as they grow older. Furthermore, a notification e-mail should go out to the document owner that an update may need to occur.[1]

[1] See discussion at TWiki.org (http://www.twiki.org/cgi-bin/view/Codev/ShowPageAgeVisually), accessed July 29, 2011.

But tools alone are not sufficient. There must be a perception shift in operations and development teams that embraces the use of documentation as a tool for improving software. For example, a page that requires updating can yellow forever, but that doesn't necessarily mean someone is going to update it. There has to be a response to a notification, and if the recipient doesn't understand the value of creating and maintaining documentation along with code and systems, a response may not be forthcoming. But if the engineering staff is aware that the system will be notifying them to update the documentation when necessary and that they should do so when a notification is received, they are more likely to follow through. This may sound trivial, but simply making the engineers aware will encourage them to be conscientious about documentation.

Benefit: Documentation Is Integrated into Regular Activities

When developers become accustomed to receiving notifications about necessary updates and responding to them, they will begin to see the value of documentation in improving their software, and this will encourage them further to respond to notifications in a timely fashion.

Document Types and Templates

The types of documents and their structure will vary from organization to organization. When you adhere to a template, creating documentation essentially becomes a matter of simply filling out a form. (This isn't the case with visual aids, like diagrams. These can be automated as well, but it's a lot of work.) Templates make it easier for engineers to become more involved in the process because they no longer have to figure out what to write and how to format it. A template structure also makes it easier for other applications and automated processes to consume the information in a document management system or wiki, for example converting a list of servers to be built into a configuration management file and vice versa. Remember, templates should be flexible and able to change their use according to your needs.

Here are some common document template types that can help developers and operations engineers to be more prolific in their documentation:

- API Specifications and References
- Getting Started Guides
- Use Case Documentation
- User Interaction Workflows
- Architecture Diagrams
- Infrastructure Design Documents

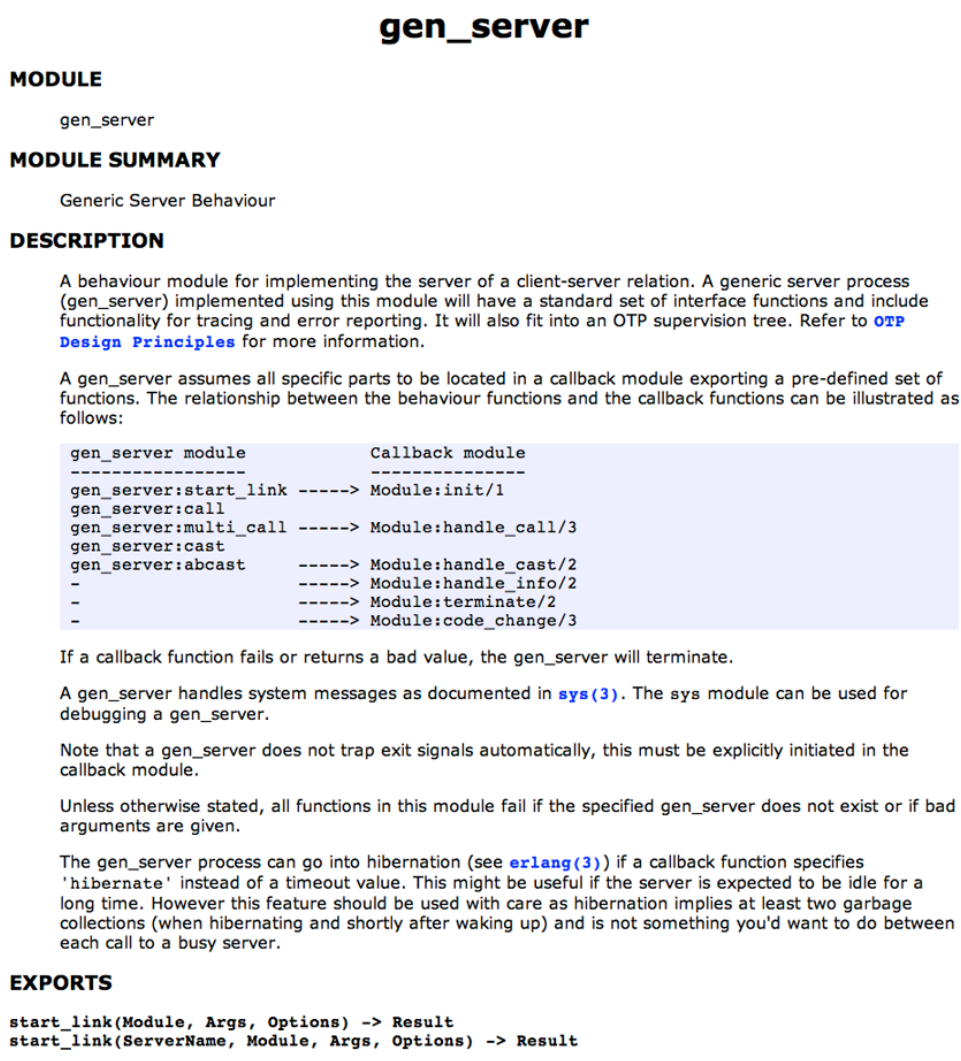
Each of these is discussed in the following sections.

■ **Note** The document types discussed below are primarily taken from open source frameworks, although the point here is to simply illustrate a type of document.

API Specifications and References

API specifications are generally terse by design, and are intended primarily to let applications developers know what functions they can implement in their code. Some API references are better than others; a good API reference not only has information about the function calls, it shows that information written in *plain language* alongside.

Figure 4.1 shows a well-written API reference. It is the first segment of a lengthy reference. For the complete reference, see `www.erlang.org/doc/man/gen_server.html`.



gen_server

MODULE

gen_server

MODULE SUMMARY

Generic Server Behaviour

DESCRIPTION

A behaviour module for implementing the server of a client-server relation. A generic server process (gen_server) implemented using this module will have a standard set of interface functions and include functionality for tracing and error reporting. It will also fit into an OTP supervision tree. Refer to **OTP Design Principles** for more information.

A gen_server assumes all specific parts to be located in a callback module exporting a pre-defined set of functions. The relationship between the behaviour functions and the callback functions can be illustrated as follows:

```
gen_server module            Callback module
-----------------            ---------------
gen_server:start_link -----> Module:init/1
gen_server:call
gen_server:multi_call -----> Module:handle_call/3
gen_server:cast
gen_server:abcast     -----> Module:handle_cast/2
-                     -----> Module:handle_info/2
-                     -----> Module:terminate/2
-                     -----> Module:code_change/3
```

If a callback function fails or returns a bad value, the gen_server will terminate.

A gen_server handles system messages as documented in **sys(3)**. The sys module can be used for debugging a gen_server.

Note that a gen_server does not trap exit signals automatically, this must be explicitly initiated in the callback module.

Unless otherwise stated, all functions in this module fail if the specified gen_server does not exist or if bad arguments are given.

The gen_server process can go into hibernation (see **erlang(3)**) if a callback function specifies `'hibernate'` instead of a timeout value. This might be useful if the server is expected to be idle for a long time. However this feature should be used with care as hibernation implies at least two garbage collections (when hibernating and shortly after waking up) and is not something you'd want to do between each call to a busy server.

EXPORTS

```
start_link(Module, Args, Options) -> Result
start_link(ServerName, Module, Args, Options) -> Result
```

Figure 4.1. *Well-written API documentation from Erlang.org/Doc*

Benefits

- Detailed reference for how the software works
- Acts as a reference for software inner workings
- Allows terse technical information to be expanded, perhaps explaining what a method does and the various parameters that can be passed to it

Downsides

- Only explains one small part of the software
- May be difficult to understand for someone working with the software for the first time
- Nontechnical staff, such as project managers, business management, and some operations engineers, may not be able to understand it.

Getting Started Guides

Getting started guides are primarily written for lower-level developers or operations engineers to help them get acquainted with a new technology, platform, or API so that they can quickly begin working with it. In a well-designed organization, this type of document could be replaced by developer workstations set up using a configuration management and developer workstation provisioning system such as Vagrant, Chef, or Puppet. However, not all organizations use such systems, and sometimes developers prefer to set up the application, system, or API manually to make sure they understand how it's installed and configured. In such cases, these guides are invaluable.

Note Tools can supplement a great deal of the documentation process via automation. For example, if you run a single Vagrant script to set up your dev box, you don't need step-by-step documentation.

Getting started guides free senior developers from having to walk new or more junior developers through the process of setting up a new application, system, or API, which saves time and boosts efficiency for all concerned. A well-written guide, such as the one shown in Figure 4.2, lets a developer or operations engineer quickly get set up and begin working with a particular technology.

Here are some items to note in this example:

- Author: The name of the person who wrote the document, or last edited it, in case someone needs to get in touch with that person. Many good documentation systems also have a revision history, to make it easy to track down the subject matter expert for a particular page.

Figure 4.2. *jQuery Getting Started Guide*

- Table of contents: A clickable list of the contents makes navigation much easier.
- Body: Step-by-step instructions show how to get up and running with a given technology so that a developer or someone new to the technology can begin working with it in a short period of time. An engineer should never have to fill in the gaps by asking someone or piecing together fragments of information on his own, which isn't an efficient use of time.

Benefits

- Someone who has never used the technology can quickly get started using it or developing for it
- Allows a fast introduction to something new so you can start working on it
- Acts as a stepping stone for learning more advanced topics

Use Case Documentation

Truly useful documentation will include more than getting started guides and API references; it will also contain information on how that technology should be applied. Use case documents provide detailed information and explanations of how a particular technology, API, platform, or system should be used in a specific business context.

An example of a use case document is shown in Figure 4.3. This example shows how Apache Hadoop, an open source library and platform for data processing and MapReduce operations, can be used for processing Web server logs or running Hadoop's Hive Platform on Amazon's EC2 Cloud platform for performing data warehousing.[2] This example of performing data warehousing on EC2 using Hive is targeted at developers and engineers who are doing large amounts of log processing for business intelligence purposes and explains specifically how the technology can be used in that capacity.

Use case documentation does depend somewhat on the API references and getting-started guides, because it's usually best to gain a general foundation and understanding of a technology before using it, but the more knowledgeable or impatient developer may be able to quickly begin evaluating whether a particular application or platform works for him by examining the use case documentation first.

Benefit

A user will understand *how* a particular technology can be implemented in a particular scenario or application, such as how to use a general technology, such as Hive / Hadoop in the case described above, on a cloud computing platform, Amazon Web Services, for example.

User Interaction Workflows

User interaction workflow diagrams represent the flow of a user's interactions with a Web site or Web application and are typically used only when designing a new Web application or modifying an existing application. These diagrams help plan how a user, whether Web site visitor or internal business user, will get a desired result from a Web application. Figure 4.4 demonstrates

[2] To see the full use case, go to https://cwiki.apache.org/confluence/display/Hive/HiveAws. (Figure 6-3 accessed July 28, 2011.)

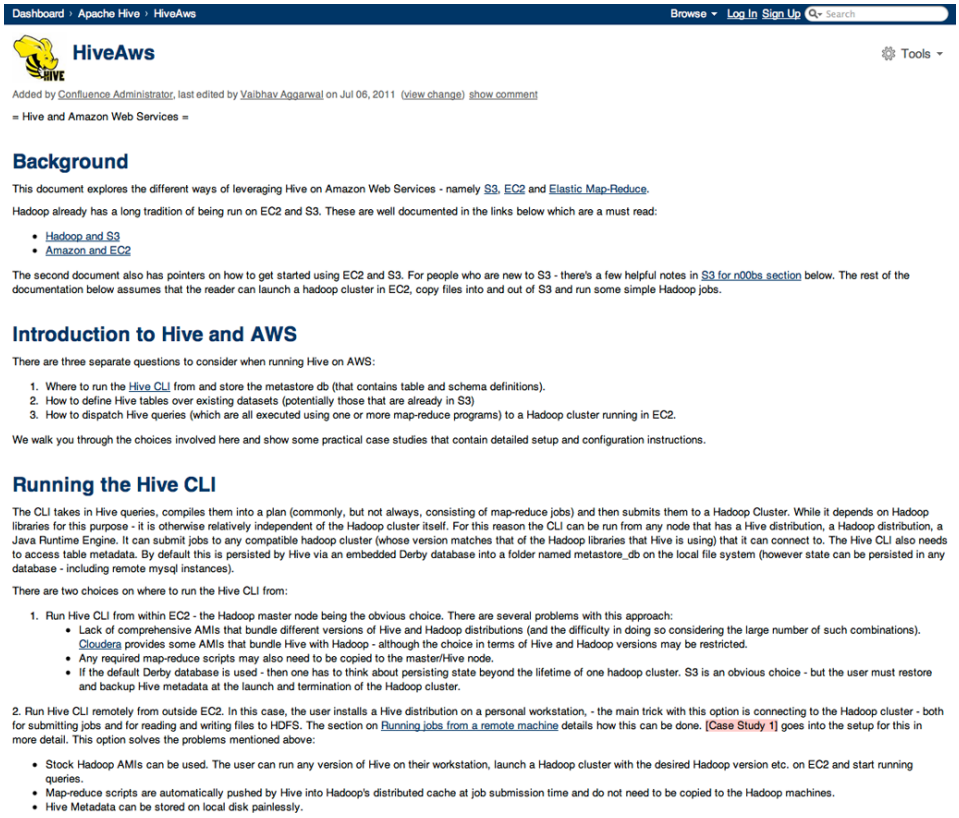


Figure 4.3. *Use case documentation*

the function of adding an item to a cart in an ecommerce Web application and shows the available paths the customer can take after doing so to reach the end result of buying a product.

Benefit

User interaction workflow diagrams clearly show how a user will interact with a Web site or application and describe all possible paths to an end result in the application. Knowing which "checkpoints" a user can go to in order to obtain a desired result, such as reading an article or checking out a shopping cart, will give the operations staff a better understanding of *how* the Web site is intended to function and the ways it can be interacted with, which helps when troubleshooting unexpected behavior from the end-user perspective.

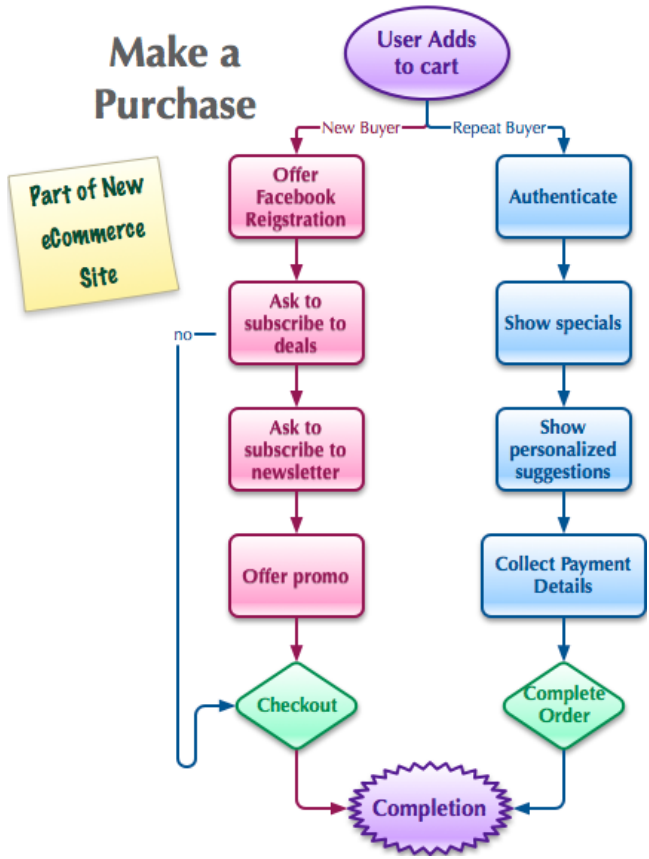


Figure 4.4. *An example of a user interaction workflow diagram*

Architecture Diagrams

Architecture diagrams illustrate how a complex Web application's components are ordered and interface with each other. This is a critical document for any Web-based organization because without this visual map, it's exceedingly difficult to grasp the whole of the Web site stack. In contrast, a map of all of the components that make up a Web site or Web service makes it much easier to understand the relationships among all of the various components, such as the Web tier, application tier, system infrastructure, network infrastructure and data tiers. Even a complex arrangement of applications and platforms can be expressed quickly and succinctly in a visual format, as shown in Figure 4.5.

As Figure 4.5 shows, even if you've never worked at this particular Web organization before, you can get a very good idea of how the Web architecture is configured and laid out from the diagram. Large-scale Web sites will likely have a much more complex architecture

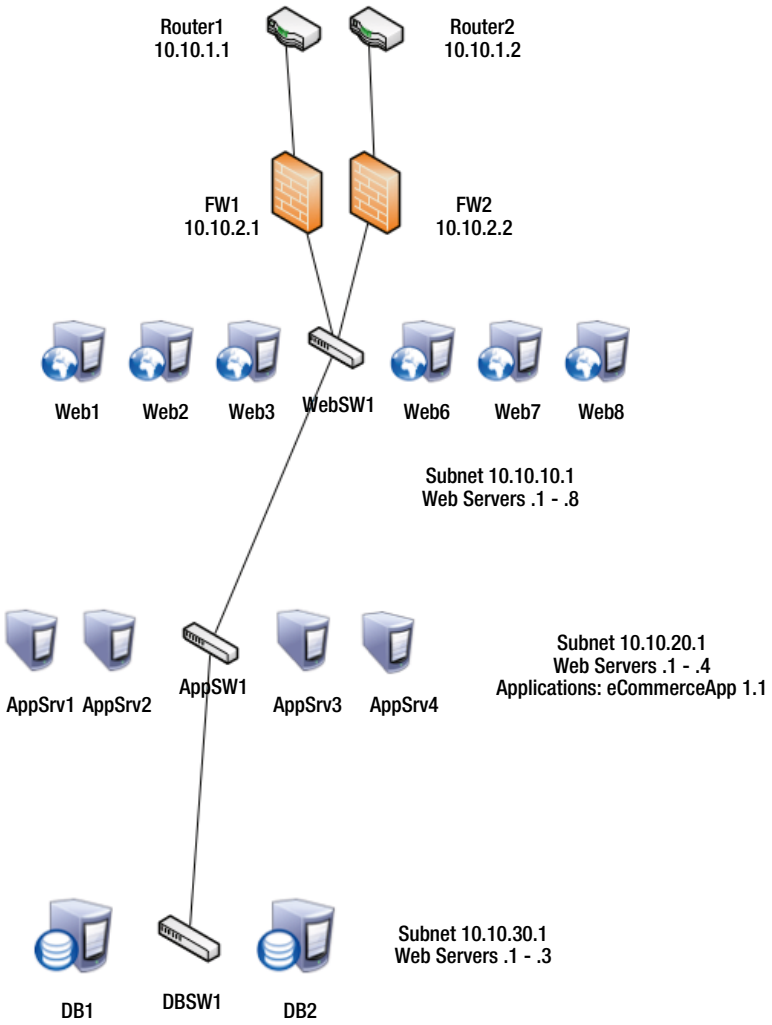


Figure 4.5. *A basic Web site architecture diagram*

than this, comprising many Web and application architecture diagrams, but even small organizations should have at least one diagram that paints the entire picture of the Web site or application.

Benefit

Architecture diagrams summarize complex interactions of various systems and applications in a single map, providing a high-level view of how the various components of a complex system work together. This helps new employees or those who haven't worked on the project before

understand the system as a whole as well as the individual parts make up the system. This makes it easier to troubleshoot, maintain, and improve the applications.

Infrastructure Design Document

An infrastructure design document represents the necessary hardware, server, and networking components required to build the system infrastructure that will host a given Web application. Infrastructure designs can be derived from an application architecture diagram or vice versa. Typically an infrastructure design will include the hostnames of servers, their IP addresses, the operating system, and any applications installed on the servers. This type of document allows both the Web development and operations teams to come to an agreement as to the necessary components required to build an environment for hosting a Web site or Web application. Using a template to fill out this information is recommended, and most wiki systems offer a template system so that different groups, Development, System Operations, Network Operations, Database Administration, or other operations teams, can get all of the information about what components are required in a structured format. The teams thus have the information needed to build the Web stack, which means less interaction is necessary and reduces the chance of errors.

Furthermore, this structured document can be converted or read into provisioning systems such as Kickstart, Cobbler, or Spacewalk, for example, and configuration management systems such as Chef, Puppet, Cfengine, and the like, so that configuration and provisioning can be automated. The level of automation required in going from document to actually provisioning a new server and deploying code to it will vary based on the size and complexity of the organization, but in most cases the specification of what components and configuration details the Web stack or environment will include will be defined in a document such as this.

In the simplified example in Figure 4.6, you can see that there is a name for the server farm that will constitute a new Web application, and also that there are server names, IP addresses, subnets, a load balancer, and firewall information, as well as an application server and the code will be deployed to application servers in this particular environment.

Benefit

Infrastructure design documents help describe various self-contained Web application server farms, and are essentially the recipe list indicating what it takes to build a functioning environment for serving a Web application or Web site. This list summarizes all of the ingredients needed to make up a Web site, including the networking, application, database, and software configuration information, which can even then be updated by or consumed by a configuration management database and configuration management systems in an automated or semi-automated fashion.

Automating Documentation

There are various degrees of automation by which documentation can become a living part of any Web-based organization or business, and that level of automation will vary depending on the size, complexity, and needs of the business. If you consider the different document types discussed earlier, you can see that information is clearly laid out, and the method by which information is populated into documentation will vary. For example, when I enter into an infrastructure design

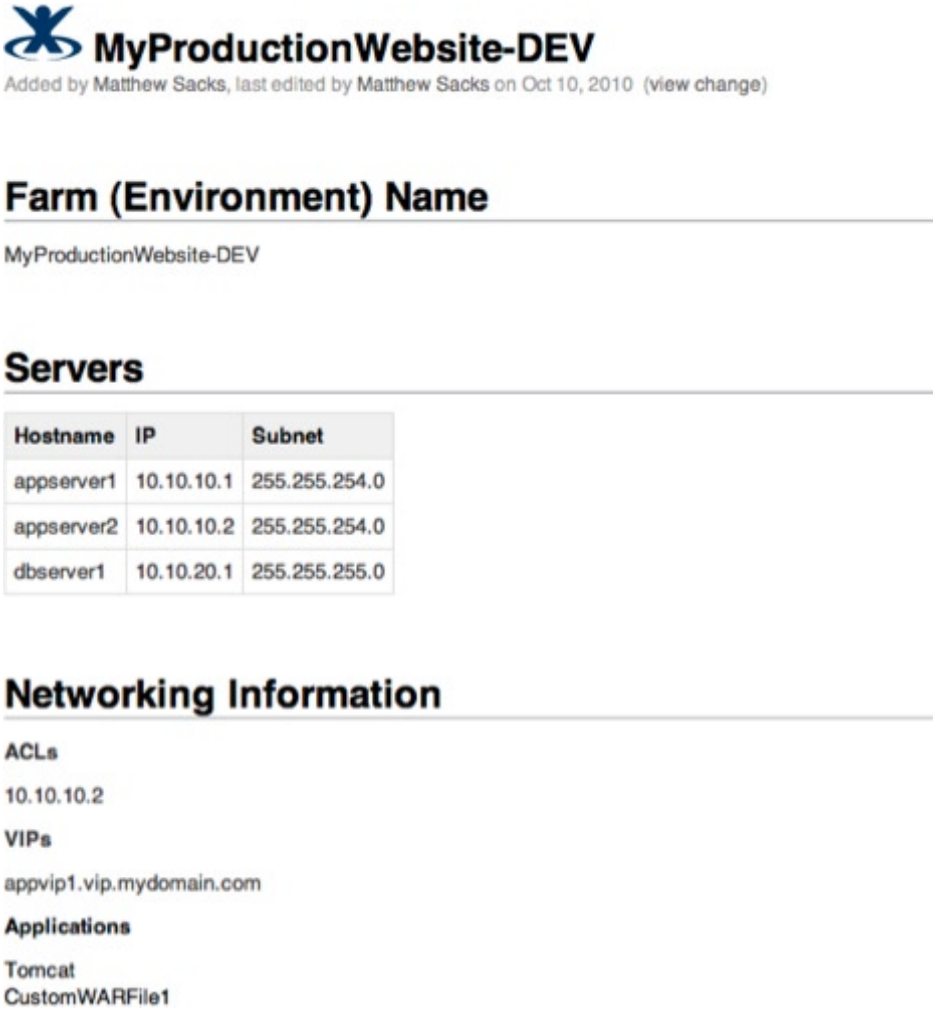


Figure 4.6. *An infrastructure design document created in Atlassian Confluence*

document that I want to deploy CustomWARFile1, there may be some information within that Web application archive file (used with Java Web applications), that I need to collect and pull into my documentation. I could enable my wiki to be written by my build-and-deployment system, so that when I push new versions of my code, my code deployment tool has a process by which it reads its associated configuration file, updates it with version information and other application metadata, and then pushes it to this wiki page. In this way, nontechnical users can get information about the application that is deployed in this specific Web environment's details, without having to ask someone or know how to access the source code repository or the build-and-deploy tools' changelog history. This requirement may vary among organizations, but documentation can be automated in the way it's integrated with the various configuration management, source code management, and deployment tools in an organization so that it brings new life to otherwise lifeless wiki pages that have stale information about a Web environment's configuration details.

Summary

Documentation is an often overlooked aspect of the technical staff's work, and to get developers to document regularly and consistently generally involves a cultural shift because it often seems to be a waste of time that could better be spent producing precious code or maintaining systems. This is a common fallacy that usually disappears when people or groups take the time to create systems for documentation and then evangelize the benefits of that process within their company. This is when a cultural shift in the way documentation is produced takes place, and technical staff begins to take it upon themselves to begin evangelizing the production of good documentation throughout the organization as a common practice. Having support systems and tools in place helps to overcome the reluctance many engineers have about integrating this seemingly time-consuming process into their daily repertoire.

There are a number of fairly standard types of templates a developer or operations engineer might look for when starting to work at a new company. With these templates, individuals can quickly familiarize themselves with the complexities of the various application, system, and network infrastructure details that comprise a large Web site. The various templates make it easier to become acquainted with new technologies and procedures without having to bother colleagues. Templates make it easy to follow standard procedures for documenting and describing how software or a process to support software is managed, and there are a number of fairly common types of documents and diagrams that help describe and educate engineers on how complicated software and infrastructure is arranged, and how it works internally to make up and support a complex Web site.

CHAPTER

5

Automating Infrastructure and Application Provisioning

The hardware and software that comprise the infrastructure of a large website typically consists of some number of servers residing in a datacenter. Installing and maintaining this infrastructure must still, to this day, be done manually to a degree. In this chapter, we'll discuss how virtualization (and cloud and private cloud) infrastructures are mimimizing that requirement as the field of computing advances, and how some of the processes can be automated. Before thinking about automating, however, it's a good idea to look at how processes currently work.

Reviewing the Web Stack

For a large scale website with hundreds of servers, tasks such as installing a new application server, deploying new code, or auditing systems to check them for consistent configuration settings becomes very time consuming. Think of installing an application server onto hundreds or thousands of servers, one at a time. This enormous task can be mitigated by writing a shell script that allows for installing the application server from a central data store using a single command.

For example,you might have several hundred servers in the subnet 10.10.20.0/24, each that needs an application server binary installed on it and a configuration file modified so the proper IP address and subnet are associated with it. Ideally, you'd use a script that installs the binary onto the destination server automatically and then modifies the configuration file. Using a script to automate common installation and configuration tasks is much more efficient and reliable than doing it manually. It's still error-prone, however, and doesn't have any sort of quality assurance checks or validation steps to ensure that the desired task was actually achieved.

Now consider what it means to put a large-scale website online. You'd need to install and configure your servers with an operating system and the services and processes that comprise the underlying system for running Web applications, including configuring network settings, time settings, disk drives and file systems, and then installing any necessary software packages. This entire process is known as provisioning, and the more you can automate it, the easier your life will be.

Still, you have to consider the effort involved. For a single deployment website, such as one web server with a single database and application server, it's probably not worth the effort of setting up a system for automating the installation and configuration of the infrastructure. A rule of thumb about automating is to do it when there's a high frequency of change or a large number of systems involved. For example, if you have 100 or more, it becomes important to automate configuration and installation of infrastructure and applications even if changes are made only on a monthly basis. Similarly, if you have only 30 systems, for example, but they are constantly being reprovisioned and updated, you'll also want to automate infrastructure and application provisioning and configuration, because performing so many tasks so often can be equally time consuming. Essentially, the only time it really doesn't make sense to automate is when you're building one-off prototype systems or temporary infrastructures used for research purposes, such as performance tuning or testing a new application platform. If a particular configuration is unlikely to be used again, automation should be avoided.

Automation Breeds Consistent Web Environments

When it comes to large website infrastructures, you want to, as much as possible, keep your systems consistent. You want the same hardware, software, and functionality across the entire infrastructure. However, maintaining systems has generally been a manual or poorly automated process, which meant that inconsistencies in software and functionality were often introduced, and perhaps never discovered until reported to the system administration team or development team at a later time. As the number of visitors interacting with the site increases, so does the number of likely errors.

Consistency is a byproduct of reproducibility—creating common tasks so that they can be repeatedly executed in an automated fashion on a large scale. Typically, this means designing systems and applications in a way that they can be deployed and managed without requiring much manual intervention. When automation is kept in mind from the very beginning, building in reproducibility is much easier, rather than trying to automate a number of disparate manual tasks after the fact. If you need to deploy a certain software packages to a freshly provisioned server, for example, and you currently have to manually select which software package should go to the which server type, that's not very efficient. In contrast, in an environment where automation and reproducibility have been taken into account, the server will have some kind of tag associated with it, such as a file on the server or DNS information, which tells a configuration management system to request deployment of the proper software package type. Clearly, this kind of system makes it much easier to reproduce the desired output—getting the correct software package to the server, and this in turn makes for a much more consistent Web application environment.

When you're thinking about implementing automation, it's helpful to consider which tasks are time consuming and should be addressed first. It's also useful to measure efficiency gains as you're automating, as this will give you a clearer picture of what processes to tackle that will benefit from automation.

The metrics in Table 5-1 are estimated and will vary depending on the size of the application's source code, data sets, and the speed of the machinery they are being installed on. We will see how looking for processes and producures that take a long time can be reduced in complexity and time requirements by applying automation practices.

Table 5-1. *Estimated Times for Completing Common Tasks with and without Automation*

Task	Time (manual)	Time (automated)
Installing Unix/Linux on bare-metal hardware	1 hour	20 minutes
Editing web server configuration files to work with a specific application deployment	10 minutes	10 seconds
Installing and configuring a Hadoop node and adding it to a Hadoop cluster	1 hour	10 minutes
Installing and configuring a Ruby on Rails application with a sync from a source code repository	30 minutes	1 minute

Calculate Automation Efforts before Automating

Let's say you're at a company that has an existing infrastructure, and 400 applications need to be reinstalled with a new version of the operating system, a new type of application server, and a new code base. According to the metrics in Table 5-1, doing all of this manually would take 1 hour per server for the OS installation and configuration, 10 minutes to update the web server, 1 hour to install the Hadoop node, and 30 minutes to install the Rails application. The metrics also show how much time you might save by automating.

Let's say for simplicity's sake that the Hadoop data store application, the web server, and the application server are all running on the same Linux server. For this basic example, it will take a total of 160 minutes to get everything installed and online. If all of the installation and configuration processes run serially, or one after another instead of all at the same time, it would take 160 minutes per installation x 400 servers, which would equate to 64,000 minutes, or 1066 hours, or 44 days.

If a system of automation is in place, this time is significantly reduced. The number of man-hours required to accomplish the same amount of work decreases from 160 minutes to 31 minutes and 10 seconds per system installation; rounded down to 31 minutes per installation that comes to about on fifth of the time it takes for a manual installation.

Keep in mind that these are just estimates and that there can be many factors that influence the results. If the cost and time required to implement a system of automation is greater than what it takes to perform manually, then the effort isn't worthwhile. You need to take into account factors such as the cost of the labor, the estimated lifetime of the automation system (automation systems require maintenance, updating and engineering efforts as well), and how quickly an organization needs to be able to adapt to change once it's up and running.

Suppose it takes 1,000 man-hours to implement a system of automation, test it, and deploy it in an organization. This should be factored into the cost of the automated system. In the previous example, it took 44 days, or 1,066 hours, to accomplish the task of building out 400 servers for an application environment. With automation, it may only take a fraction of the amount of time to build an application server to get a website up and running. However, suppose it took a month of work to get the automation system implemented and tested properly and the staff up to speed to where it could realize gains in efficiency. You might have to carefully evaluate its worth. On the other hand, if a second 400 servers were to be built the following month, then the cost and time savings would likely begin to take effect, not to mention the time savings on maintenance, and troubleshooting errors when you don't use automation. The point is that when you're looking at the cost and time savings of automation and optimization efforts, it helps to take into account the big picture over the lifetime of a project instead of *just* the efficiency

benefits that might accrue when the automation system is in place. In this second example, with automation, it now takes 12,400 minutes to build the same 400 servers, or 206 hours, or 8.6 days to build the same set of 400 servers.

Building, maintaining, and keeping 400 servers consistent is no small feat for any organization; even for a supermassive Internet company such as Google, it would take a great effort and expenditure. But once that automation is in place, the ability to move and change the entire infrastructure in a matter of days gives that company far more flexibility, which in the world of web-based businesses is paramount. A business that can move and change faster can often beat out its competitor or gain market share.

Still, for a company with only 10 to 50 servers it may not make sense to automate, because the staffing requirements don't warrant investing the time, effort, and skill that would be required. It might be better in that case to rely on simple methods of automation such as shell scripts and scheduled tasks rather than complex, full-scale configuration management and provisioning systems.

The principle is still the same at the outset, though. It is always necessary to analyze how much time is being spent on unoptimized processes, then take a look at the cost and time required to implement the automation and weigh those against the needs and goals of the organization, then take those needs and goals and weigh them against long- and short-term initiatives before making any decisions or following any trends or standards.

Choosing an Automation Process

It sounds obvious, but before you plan how your system will be automated, you need to get input from all the stakeholders, including business and executive management. Every group within the organization whose mission involves a particular website has interests specific to its own department, and those interests may conflict with other groups. Agreement must be reached regarding the goals of automating the building and management of the Web infrastructure, including costs, time savings, and priorities of the business itself.

When you're looking into automation, keep in mind that you need to answer the what and how of automating the Web infrastructure: what requires automation efforts and how will it be achieved. For example, *what* might be included the entire deployment process. For example, will code be pre-compiled or compiled at the request of a new deployment? Before installing any software, including configuration management or provisioning frameworks, you should think about *how* automation would ideally look in your organization and in particular, how engineers will interact with the process. This might be done on an organization-wide level or a few departments at a time. Regardless, all engineers should indicate how the process currently works and how they would like things to work. This vision is an important part of determining the automation techniques that will be unique to each organization.

Once these notions have been shared, it will become apparent which specific software or organizational changes should be made to accomplish this vision of ideal automation for an organization, though the aspects and processes that are most critical for automation will be different for every organization. In any case, allowing one engineering team to decide how automation will be achieved can be disastrous. It can result in the ripping out of implemented software and hardware by other teams later on. It can leave some engineering teams in the dust, unwilling to learn and adopt new practices and tools for introducing or revamping automation efforts throughout the organization.

The following section suggests some discussion points for determining what the automation process should look like.

Buy or Build?

Whether an organization should build the automation and configuration management frameworks from scratch or buy a ready-made system is a matter of preference, time, available budget, and most importantly, the size of the organization. Some organizations might use existing open source software and customize it to their needs; others might build their own software, end up open sourcing it, and have it become a hallmark for other companies to use for years to come. (Only a handful of very large companies will typically do this, but small ones or even individuals are capable of doing it as well.)

The costs associated with building an open source automation framework are typically pretty high, and the work would distract existing engineering teams from their regular responsibilities, such as supporting a consumer-facing production website and its features and new products. However, in very large organizations with teams dedicated to engineering new tools to enhance day-to-day operations, it's probably not uncommon that the frameworks for automation will be built in-house, due to the specific needs of these companies as well as the available engineering resources. In a small company, it could certainly be the case that one very driven and talented engineer has the skill to build the automation software and is able to engineer it himself in a few short days, weeks, or months.

What it really comes down to is the corporate culture and the time available to the engineering teams. If time is a scarce resource in an organization and engineering projects are continuously far behind schedule, it's unlikely the company will engineer, or need to engineer its own automation tools for their website.

The advantage of building a custom automation and configuration management framework is that it can be tailored to the exact needs and business processes of your website. The disadvantage is that such a system needs continuous maintenance and support by internal engineering teams as business processes and functionality change over time. Of course, this is more or less the same for any software decision.

Automatable or Not?

When it comes to figuring out what can be automated, the answer is that almost everything in software can be automated. The point is that automation introduces a layer of abstraction onto the building, configuring, managing, and repairing the software that makes up a website. Automation just adds another layer, or layers of abstraction on top.

With this in mind, here are some common tasks for supporting a medium to large production website in an organization, whether the website is publicly facing or for internal use:

- Building new application or web servers
- Deploying new code to application or web servers
- Processing log files for analytics or historical trends

Using a Configuration Management Database

A configuration management database, or a central database, is designed to hold records for all software and hardware resources in a Web infrastructure. This is a further layer of abstraction and allows for centralization of all of the objects and information that a configuration management system uses to operate. It further ensures consistency of configuration management

and other systems of automation. The reason for this is that configuration data that changes isn't stored in flat configuration files, or in the configuration management database itself, where information is decentralized and becomes outdated very quickly. Keeping information centralized means that configuration management systems can depend on receiving the most current data about infrastructure configuration, such as IP addresses, hostnames, and functions of servers.

Figure 5-1 shows an environment with a configuration management and provisioning system that does not leverage a configuration management database (CMDB). This makes it difficult to find and assemble the data required to install and configure a new server and applications on the network. There are IP addresses, DNS, and application configuration data that must be entered into the configuration management system for installation or modification. All of this information must be gathered from documents and e-mail messages that are passed back and forth throughout the organization and have no centralized source.

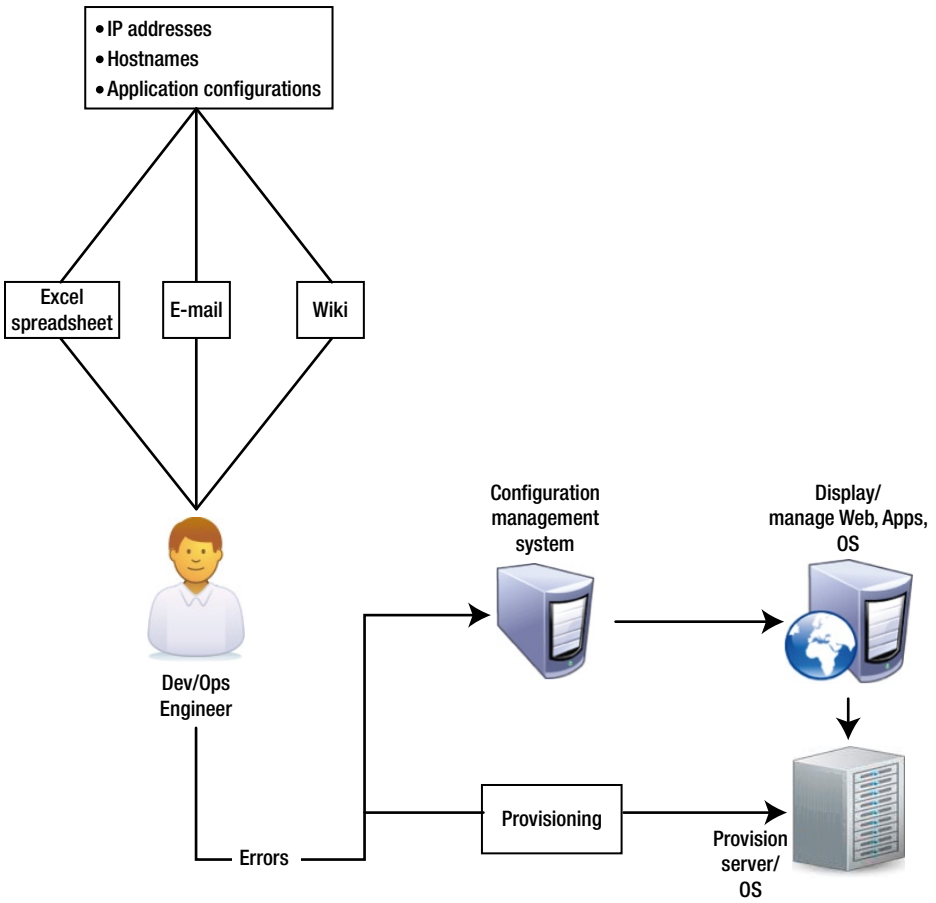


Figure 5-1. *Infrastructure configuration management and provisioning without a configuration management database*

In Figure 5-2, in contrast, the example shows configuration data being put into a configuration management database, a central source for configuration information. This ensures that all meta data about available IP addresses, hostnames, application data, and other components used to build the Web environment come from the CMDB, eliminating any conflicts or need to synchronize data from unreliable sources. Whoever is performing the configuration management and provisioning process can be assured of using the most up-to-date information to build the Web environment.

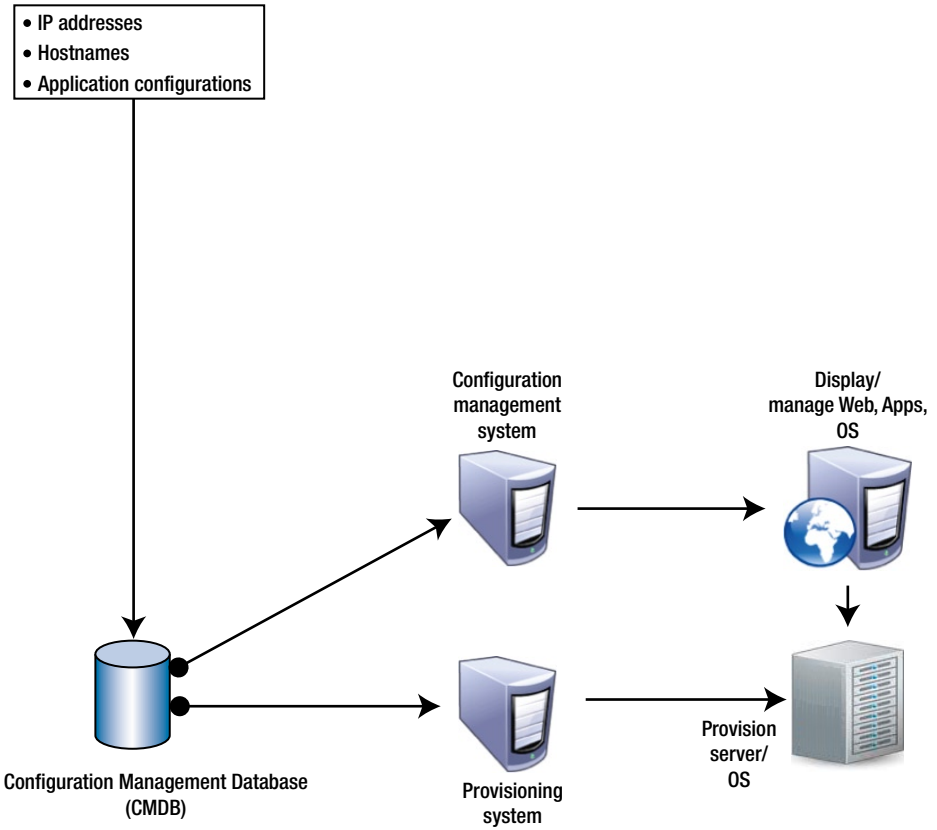


Figure 5-2. *Using a configuration management database*

A Scenario for Automation

Let's suppose we have a multiplayer gaming site, with a server farm containing a number of application and messaging servers used both to communicate with other players in multiplayer games and to produce the visual Web interface where the gamers actually play a space-shooter game.

The servers in this farm actually reside on a virtualization layer on a single 4U server. (U is short for RU, which is short for rack unit, a measure of the height of computing equipment mounted in racks. One rack unit (1U) is 1.75 inches.

In this case, rather than purchasing 4 separate 1U servers, the architect (me) decided to save on power and space costs in the data center and purchase one high-capacity 4U machine, which has just as much or even more computing, disk space, and memory power as four independent 1U servers that would be networked together. It also reduces the network overhead and latency (backplane throughput is faster than copper or fiber, not to mention the processing overhead of a switch), and it makes things simpler to keep track of. When you're automating, the simpler things are in the first place, the easier they are to automate.

Now that I have a single 4U machine, I want to keep track of the various hosts that perform different functions. For example, I don't want my database server or data store server to run on the same system as my application or web server, to make it easier to keep track of configurations. Furthermore, the database or data store server may require adifferent operating system or type of operating system entirely than the Web server or application server. Adding a virtualization layer on top of the 4U hardware layer allows more flexible management of server provisioning and configuration management. Virtualization enables the abstraction of physical resources, such as memory, disk, and CPU, which provides a lot of flexibility, such as dividing up a single machine into many machines. In this case, I'd find the number of CPU cores available and divide my single physical machine into multiple virtual machines based on that number. For example, if my 4U server has 16 CPU cores, I might divide it up into 4 or 8 virtual machines with 4 or 2 cores respectively, or perhaps 2 cores for the Web and application servers and 4 cores for the database/datastore servers.

Virtualization makes creating and designing a Web automation infrastructure automation system easier in another way. Every virtualization platform is different, and they are always evolving, but what remains foundationally most important is that any virtual machine is essentially a single file or simple set of files. The disk drive layout, the files within the virtual machine, the configuration information about the virtual machine's hardware are all stored within a file or a set of files. This makes it incredibly easy to back up state information, something many virtualization and storage vendors call a point-in-time snapshot. This is typically a block-level copy of the file system of a server or virtual machine and that is stored on a file system, which makes it very easy to roll configuration changes back or forward. Ideally, you'll never have to roll your system back or forward, but accidents do happen and you'll want that extra peace of mind and insurance when making wide-sweeping changes to your web infrastructure.

Once virtualization has been implemented in this scenario, you can move very quickly to building, decommissioning, rebuilding, and deploying of applications and server types in the infrastructure. Virtualization gives you the potential to change the functionality of your server farm overnight. If you have, for example, an application that is a gaming server, and then decide you want to sell branded products from the game in an ecommerce store, you can completely rebuild the entire infrastructure or shuffle around resources such that you decommission some gaming servers and set them up as the online ecommerce shop for the game. Thanks to virtualization, you no longer need to rack and stack physical servers or drive to a data center that may be some distance from your office. With a virtualization layer in place, it's now a matter of selecting which operating system image should be deployed by the provisioning system and what applications will be installed by the configuration management system before your new applications are online. In today's competitive world, it's critical that engineering, product, and business teams can move quickly and test and deploy new ideas into applications in days or weeks.

The next step is to develop a method for managing configurations and deploying software, and for automating quality assurance procedures as well.

Reduce Complexity

A best practice before making any automation efforts is to take a look at existing and legacy infrastructure and software and determine if anything can be eliminated or consolidated. The simpler your system is and the less duplication of effort that exists in unoptimized and unautomated code, the better the results that your automation initiative will produce.

For example, you've probably seen over-engineered, confusing legacy applications that consist of 50 different software components and 3 different databases that do nothing more than log a user into a website. Rather than try to automate the entanglement of manual effort that is most likely required for maintaining such a system, it might be better to replace such a system with a "part that fits" rather than a jumbled arrangement of adapters that then requires a massive automation effort. The more moving parts, the more chance there is for error, so taking a look at how the existing pieces of th infrastructure fit together before trying to add strings to the marionette, so to speak, will prove to be a huge time saver in the long run.

Systems of automation are excellent feedback mechanisms for bringing to light overly complex software, code, systems, and processes. Don't be surprised if while trying to automate the installation, deployment or configuration of some piece of software, the engineer asks, "Do I even need this in the first place?" In most cases, if this question is being asked, more than likely the answer is no. It can therefore be removed from the system altogether—one less thing to maintain and keep track of, which, as a general principle, improves performance, reduces costs and errors, and makes the world a better place for engineers and the websites and businesses they support.

Selecting Configuration Management and Provisioning Framework

Once the process and procedures for how a Web infrastructure should be automated have been agreed on by the various stakeholders, choosing which tools to use should be fairly smooth. Since all of the stakeholders have discussed *how* automation should look within the organization, figuring out *what* software will accomplish the desired result should not be too difficult.

There are a number of configuration management and provisioning frameworks to consider. When analyzing such solutions, I would say that the practice is the same for every website. Software changes rapidly, and typically open source software (or software that is based on open source software) tends to last longer over the years. Rather than get into the various "brands" of configuration management and provisioning frameworks, you should determine the needs for the automation system and decide whether it makes sense to build your own if the commercial or open source solutions don't meet your needs (provided you have enough engineering capacity to do so).

Here are some questions to consider when selecting a configuration management or provisioning framework:

- Is it based on open source software, or is it a closed system? (What happens if the company is acquired or goes out of business.)
- Does it automate the installation of servers as well as perform configuration management?
- Does it have auditing and reporting functionality?
- Does it allow for simulations of changes without actually executing them?

- Does it maintain a revision history and a way to roll back a change based on a point in time?
- How difficult will it be for engineering staff to ramp up?
- Can both development and operations staff learn it and leverage it quickly?
- How many systems does it support?
- How are conflicts with changes resolved?

Auditing Infrastructure

When automating systems, there may be no way to determine if a system is properly configured across a large number of servers. Having an auditing system helps in this regard by allowing an automated process to determine if a very specific attribute, such as the setting in a configuration file, is consistent across tens, hundreds, or thousands of servers. In the case of automation, auditing is simply the matter of verifying the integrity of some aspect of configuration across a group of servers. It may mean that a certain version of Apache is correct and consistent everywhere or that a configuration file is the same across a group of servers. The fact is that even with an advanced configuration management framework in place, things still get changed and out of sync, and having a way to verify the state of servers, applications, and configurations is still an important part of automation; it is a fallback to double-check things when problems arise across the infrastructure.

Keep in mind that every configuration management or automation framework will have a different way it audits an infrastructure or the collection of servers, operating systems, applications, and their configuration parameters within it. Typically it takes much too much time to audit all aspects of a web infrastructure, but rather than thinking about what must be audited when a problem arises, you should consider what you can afford *not* to audit. Suppose you have a critical problem with a production website. Perhaps this particular site draws about one million visitors per month and generates $500,000 per year in revenue. After all hands are on deck, it's a matter of figuring out what the problem is and how to solve it. In this case, let's say the problem is an inconsistent version of the application code. Most programming languages have some kind of manifest in the application's binary that determines what version of the application is deployed to a Web or application server.

Now, a given medium-size website infrastructure may contain between 50 and 1,000 web servers and application servers. That's a lot of operating systems, applications, configuration files, parameters, and knobs and dials to turn (so to speak) to figure out what is out of place that might be causing the problem and resulting in a website visitor site having a bad experience.

This is when you turn to your auditing software to determine which servers are out of whack. In this case, an inconsistent version of an application binary is causing the problem. The next step is to find which web or application servers have the problem so you can determine where to start resolving the problems. Good auditing frameworks, by the way, will allow you to create a patch or fix to be applied to the servers at hand, which can then be deployed to all of the servers identified as having the problem.

The most difficult part in auditing is figuring out what to audit. This varies depending on the type of applications being run, the source code management and deployment practices, source code repository type, programming languages used, operating systems, and even management practices and business requirements. In an environment that can tolerate a high amount of

downtime, such as a site that doesn't generate revenue, the auditing needs will clearly be less demanding that for a site that generates tens of thousands of dollars per hour. Such an environment will require a highly controlled and automated method for determining where problems exist.

In a fully virtualized environment, where all applications run off of some kind of virtualization platform, it makes less sense to have a heavy auditing framework or invest a lot of time figuring out where a problem exists, unless it is a problem that reoccurs extremely frequently, as long as there is a provisioning framework to reinstall a virtual server and redeploy all of the application code to it. It simply doesn't make sense to figure out what is amiss with a particular server that may be out of whack when it takes only 8-15 minutes to completely rebuild a server and redeploy to it.

However, with a fixed infrastructure or an infrastructure where there is no virtualization for the server, auditing is even more important. In this case, there's no way to roll back changes automatically using point-in-time snapshots or by kicking off a process to rebuild the entire server, complete with applications, from scratch.

■ **Note** Some configuration management frameworks will keep a history of every change that was made in a database and allow for point-in-time rollbacks of changes, even if there is no virtualization present.

In comparison with a virtualized environment, it's critical to determine what files, configurations and applications should be audited because of the lesser margin for error in a fixed environment. There are solutions, such as source code management repositories, that can solve a lot of problems, but they won't solve issues that result from system-wide configuration changes, such as applying a kernel patch or updating firmware and device drivers on a network card or disk controller. Such changes often have unexpected results, and then it's left to the engineer to determine which cards have the problematic attributes and then address each system one by one.

An auditing framework can provide the following benefits:

- Finds problems you didn't know existed
- Improves performance of systems and applications
- Helps the Web infrastructure recover from a problem that would otherwise mean hours or even days of downtime
- Locates problems when the web infrastructure experiences an issue that logs or monitoring systems aren't revealing

Automating Deployments Using Configuration Management Systems

One of the easiest ways to deploy code to a production environment is to automate the deployment using your configuration management and provisioning systems. This lets you, for example, easily resolve dependencies, such as shared libraries, or other binaries that code deployments may require to operate properly.

Most configuration management systems basically abstract physical and logical software resources into configuration files, which are then edited in a descriptive, domain-specific language. The desired changes are then applied via some kind of engine to achieve the desired results on the operating system or applications.

These configuration management frameworks can interface with deployment tools that actually place software binaries onto the application or web servers. This can be very useful for both development and operations teams because of the power of using a common language. Instead of software developers dealing with source code manifest files and operations relying on a release engineer or software developer to determine which version of a file is required for a particular environment or type of application server, the configuration management file can simply define which version of an application is required and where it goes, and all of the dependency management and installation will be handled in the background, without any manual intervention. Don't be fooled, though. This isn't an easy utopia to be achieved simply by purchasing, building, or implementing an existing configuration management system and dependency-resolution framework. It requires manual testing and engineering on the back end by operations and development engineers skilled in the practices of automation.

Summary

This chapter discussed how tracking certain metrics can help shape automation policy and make more effective use of automation efforts. What's especially important in the whole process is for all stakeholders to work out ahead of time what the automation will encompass. If at all possible, the web infrastructure should be built with automation in mind, by leveraging virtualization and configuration management systems. If you're dealing with an existing Web infrastructure and have to add automation later, this adds some complexity to the process. When you're setting up your automation, here are some practices to consider:

- Provisioning a new virtual server for developer use
- Kicking off a load test from QA
- Developing self-service operations practices, such as performing a Web application security scan, and making them available to QA, development, and so forth
- Kicking off a build-and-deploy process, which either operations or development can perform
- Tying together traditionally disparate systems, such as pulling web or application servers from load-balancer rotation automatically via API calls

Keeping an eye out for these kinds of potential optimizations illustrate the spirit of this chapter and how you might go about your automation efforts.

CHAPTER

6

Production Launches

The launch is usually the easiest part of the entire process of getting new websites and applications into production. Most of the testing should have already been done, so many bugs and errors that surface only when random production traffic hits the site or application will most likely have already been discovered. The teams involved will simply be following a script determined during reviews at the prelaunch phase, and the process is mostly a matter of executing steps on a checklist or kicking off automated scripts to perform the actual launch and reveal the new website or application to the public.

It's especially important for the development and operations teams to be working together during the launch because if problems do arise, they will have to be quickly identified and evaluated by these teams to determine to what difficulty any errors or bugs present to the business and whether they need to be resolved immediately or at a later time, or whether the launch should be rolled back entirely.

The launch typically involves developers and operations engineers sitting side by side, executing the scripts or checklist agreed upon during the prelaunch phase. If automated systems are being properly leveraged, the entire launch of the new site or application should take no longer than a couple of hours. The more complex the new application, the more time it will likely require to release, and there are many factors that affect release timelines, such as number of locations being released to, content delivery network propagation, and the degree of automation the engineering teams have put into the launch process.

Understanding the Process

Releasing a new web application can be understood as one part process and two parts experience—and having a good contingency plan. The devil is always in the details and last-minute changes can completely derail the launching of a new site or application, which is why it is imperative to have a well-thought-out procedure for the launch. Identifying some of the common problems in how the development and operations teams interact with management is a crucial part of this picture. Before a successful site or application can be built, the way that the teams operate within the context of the business and its management, and the common problems that hamper engineering efficiency, must be dealt with before the first design specification of a new web project is created. The human element of engineering must be considered before any new web development project is undertaken, because any new project will have many unknowns and unexpected results and problems as it is being created.

The essential difference between launching a completely new web application or website and updating an existing one is that there is no historical data about how the application will perform, what bugs will arise, or how much support effort will be required to maintain that application. There is only so much debugging that can be done prior to releasing a new application into production, and many errors, bugs, and unexpected behaviors will appear only after the application has been launched and is subject to production traffic.

Regardless of how much use a new web application is expected or intended to receive, testing the application in a production environment before fully rolling it out is crucial. Until a website is live, software developers, QA engineers, and operations engineers can only have a partial picture of how the application will *actually* perform in production.

Conceptual Development of a Website: *Concepting*

Among web development projects that will be used heavily by consumers or end users, the ones that succeed are those that are extremely well thought out, and the ones that turn out best typically employ the services of an industrial designer or engineer. An industrial engineer's responsibility is to take business goals, technical specifications, and other relevant factors and turn them into a model or prototype and present it to the architect or visionary whose responsibility it is to create something new. Coming up with something truly innovative that isn't a copy of something else is incredibly difficult, and when it comes to websites, much of what we do has been done many times before.

When conceiving a new version of a website, some things must be taken into account before even starting on the project, such as the size and scope, the technical complexity, cost, and available infrastructure. People who have not owned, managed, or been heavily involved (as an engineer) in a production website, or a site of some high technical value, may not have enough understanding of the infrastructure required to support a high-traffic site, such as those that support millions of visitors per day or more. Such sites require a large investment in rack space, cooling, and power. However, most startups don't need this type of investment right off the bat. Growth usually happens over time. Still, considering just how many users are expected to visit the website initially at any given moment will help in the concepting phase.

The Foundational Questions of Concepting

There are a number of essential questions that need to be answered during the concepting phase; among the most important are perhaps "what problem are we trying to solve?" or "how will this make someone's life easier?" With an application, you have to ask as well how it will be used, why someone will use it, and how many people will use it. It's easy to get hung up on the "vision" of a new website, such as a social network for a specific group, or the ultimate social network aggregator that includes and binds all social networks together. But the focus always should always come back to the foundational questions.

There are millions of websites, mobile applications, and software packages out there that already address millions of specific issues and problems. Why then would someone decide to build something new if so many things have been done before? Not all sites are done well. Many are difficult to use, slow, unattractive, and confusing, with few or no alternatives. Any of these reasons might be enough for building a new website, and the best reason may be simply that you have an idea for something entirely new, or a way to improve on what currently exists. MySpace is a classic example of a "better" website being created to meet a need. MySpace was

created because on Friendster there was no way for bands to build their own pages and share information, so Tom Andersen and his colleagues set out to build a social network where musicians and bands could create their own profile pages. At its core, MySpace was no different from Friendster, but it wanted to improve a few things. And today, though MySpace still exists, Facebook is the leading social network site. I remember when Facebook was available only to those who had a *.edu* domain name, and it wasn't even considered on the map in comparison with MySpace's popularity. Friendster, MySpace, and Facebook all essentially do the same thing—allow people to share and connect with friends. One improved upon the other, and was then improved on itself, and each had increasingly more features and innovations. What fostered Facebook's growth and success were a consistent, clean design, and a user interface that focused primarily on what friends were doing rather than customizing pages and rapidly adding as many friends as possible. At some point, the founders of these social networks would have asked themselves the foundational concepting questions for building a website:

1. What problem am I going to solve?
2. How will I make someone's life easier?
3. Why will someone use it?
4. Who are my competitors?
5. How can I do things better than my competitor?
6. What will be unique about my website?
7. What is my budget?

These questions apply not only to social networks but also to any site that's being created, and now to all the mobile devices and other interconnected devices as well. Of course, the Internet and the web have limitations. Ultimately, the Web is simply a free flow of communications with visual and audio interfaces. The Web is a form of human expression, at its most fundamental level; and yes, it can therefore serve lofty goals such as solving world hunger, but doing so would likely prove difficult to accomplish for any one organization. When concepting, it's important to keep the scope of problem-solving in mind and be realistic about resources and budget constraints. Just creating a website to try and collect donations and educate people about world hunger would be a massive undertaking and would require quite a great deal of funding in order to get off the ground successfully, but it is possible if the vision is thoroughly thought through.

Cost/Benefit Analysis

Before any code is even written, the business side of an organization and the engineering teams should meet and figure out how much the application is going to cost to produce and maintain, and if it makes sense for the business to go forward with it. Many organizations do not involve the development and operations teams in the planning stages, and directives are handed down from upper management without any consultation with engineering teams. In fact, engineering teams may be telling the business that certain purchases of hardware, software, or hiring of staff will be necessary to support existing web properties, and these might affect the business's plans and budget for new projects for the next year or two. Business and engineering teams must meet during the planning phases to balance the costs of supporting existing projects and building new projects that should generate revenue. By involving engineering from the beginning,

management can gain more insight into how existing applications and infrastructure might be better leveraged to support new software development initiatives.

It may seem somewhat elementary to suggest that a business should perform a cost/benefit analysis before setting out to build a new website or application, but many businesses don't take their engineering teams' input into account when performing this process, which can mean the difference between choosing a web application that might take minimal effort to build and offer a high return rather than a massive undertaking that may be unlikely to produce real results in terms of generating revenue or growing the business.

Once this meeting of the minds between business management and engineering and operations has been initiated, both sides will have a better picture of what to expect, and can collaborate in planning for the desired outcome, a necessary step for making better decisions about building websites and applications that will support business initiatives.

Special Projects Team

Having a special projects team is an excellent way to be prepared to build and onboard new websites and applications. Onboarding a new website with a special projects team is akin to a having a hydraulic forklift to lift heavy objects as opposed to trying to lift a heavy object using a team of three people. The necessary expertise of the special projects team will vary from organization to organization, depending on the nature of the business and the various platforms and technologies used. Generally, a special projects team dedicated to building and launching new web projects would consist of positions typical of any large-scale website, like the following:

- Front-end engineers (developers proficient in HTML, JavaScript and CSS)
- Data architect
- Application architect
- Database developer
- Operations engineer
- QA engineer
- Network engineer
- Senior application developers
- Mobile engineers
- Integration engineer
- Performance engineer
- Security engineer
- Project manager with software development experience

The primary difference between the engineers on the special projects team and those in the rest of the organization is be that they are experienced in building *new* applications from scratch, which requires a completely different skill set and mentality compared with engineers who maintain other people's applications. On the special team, the engineers will likely have experience in more than one of the other roles on the team, giving them a certain breadth that is less common with engineers who have been operating in a single role throughout their careers.

Most importantly, members of the special projects team should be rotated, so that exposure to different technologies and new initiatives is evenly distributed, and not all of the responsibility for understanding a new technology is not always placed on the same team, although many veterans may be a foundation stone of the group and remain on the team frequently.

Marketing

Building a website, even a complicated one with many features and lots of functionality, is trivial compared to marketing that website. It is incredibly difficult to start a business, any business, and one of the hardest parts is marketing. Marketing is the action of promoting products and services in hopes that people will buy these products and services. Marketing budgets are seldom looked at in the conception phase because startup founders are so in love with their *idea* that they don't focus on the foundational questions related to the new site. I am by no means a marketing expert, but I'm a fan of reading textbooks on a given subject I want to learn more about, and attending relevant meetings, such as, in this case, the American Marketing Association meetings.

There are a few things I've learned about marketing a new website or application, which will hopefully save you time, effort, and money before you venture down the road of investing thousands or hundreds of thousands of dollars in advertising.

First and foremost, have a good product or service to offer. Second, have a good website. If your website is your only product or service, you've got the cart before the horse and should rethink your concept. You need to consider:

- Credibility
- Reach
- Quality of placements
- Marketing design of the site

The questions that come into play are: Has the website been designed properly for marketing? Is it easy for people to talk about, discuss and share information within the site or, at a bare minimum, to quickly and easily allow visitors to direct others to the website?

These are critical questions that will help reduce the amount of money spent on the advertising, staff, and effort needed to drive people to the website. Once people have been driven to the site, it becomes a matter of figuring out how to convert a visitor to a sale. Only large websites like Facebook and Google can depend on large numbers of users advocating large rounds of funding until they figure out how to generate revenue. In reality, most companies do not have this luxury.

Design Elements of a Launch

The design of a website is more than just aesthetics. Some sites are more attractive and some are more functional, but it's the combination of art and science that creates websites that you'll remember and return instead of one you just use when necessary, such as an online banking site. One is an experience, and the other is a tool to be used to meet some kind of end, such as transferring money. Sites that combine both art and engineering are the ones that create an experience for the user when he simply visits the website. The factor that makes the difference is the artistic inspiration that permeates the site.

Artistic inspiration might make users feel as if they are underwater, or provide a feeling of expansiveness using white space, 3D technology, and animated zooming menus, for example. Of course you have to think about:

- Who will use the website?
- How will the website be used?
- How many people will use the website?

In creating a concept, the most valuable part of a new web development endeavor is the discussion phase. Before any specifications are laid out for an industrial designer, before any models are drafted, people should get together and discuss what it is they are trying to build and for what purpose. This addresses some common problems encountered when setting out to build something. The discussion phase helps take ideas that exist in someone's mind and brings them into reality a bit with questions such as "what if" and "how will it."

For example, suppose you were going to build a local social networking site to enable a small neighborhood to discuss common issues that come up, which would help create a focused agenda that would make local government meetings more productive. Before actually getting started, it would be prudent for the engineering team building this new application to sit down and discuss how the site should work before they start laying out any technical or business specifications.

The meeting might consist of an industrial engineer, a web developer, an operations engineer, a business person who works on funding, and a city council member, who will all sit down and discuss what they would like to get out of this project that, for the sake of discussion, we'll call NeighborhoodNet. The primary constraint to look at first would be to consider high-level goals against the available budget for building such a project.

Suppose the available budget for this project was $500,000, which is a large budget to work with for a project of this size. Now suppose that one of the high-level goals was to build a database of local residents to be invited to join NeighborhoodNet once it's launched, but it turns out that collecting the information about all of the local residents will cost $100,000, one-fifth of the total project budget. Chances are good that this goal will be discarded early on in the discussions, and better earlier than after work has already begun. The point is to bring up all of the major goals of the project and consider what is feasible within the budget before drawing up any specifications. Michelangelo said in creating the David statue that he simply removed chips of stone until what was left was David, and in planning the building of a large or even a small website, the practice is the same.

Here are some samples of what the discussion of NeighborhoodNet might cover before getting to the specifications stage. These might initially be expressed as simple wish-list items that will eventually be evaluated to determine whether or not it is doable.

NeighborhoodNet Discussion Part 1, High Level Goals:

- I want the website to have the ability to identify the exact location of potholes in the roads of my community.
- I want a neighborhood watch mobile application that will integrate with the site and alert others to suspicious people in the neighborhood.
- I want the website to be inspired by a mosaic of tiles because of the well-known mosaic in the town center, which is a symbol of the neighborhood and gives it its character.

High-level goals should not be cluttered with minute details. Minute details can completely derail a web development project and can slow down or completely keep a project from attaining its high-level goals. Of course, you do want to keep track of the desired details because they will help shape the vision for the project during the planning and concepting phases, when the finished product is little more than a hazy picture to imagine. The details might look something like the following.

NeighborhoodNet Discussion Part 2, Design Details:

- Buttons on the website should have rounded corners
- All submitted content should be sent to a moderation queue for administrators to approve or disapprove.
- There should be JavaScript tooltips to show first-time visitors how to use the website, without requiring them to read a lot of documentation to learn to navigate the site.

Inspiration and Vision

When building a website, it's often difficult to figure out how and what will inspire the design and layout. For example, my friend Javier Mozo, an industrial designer, and I would discuss the creation of a new website. I might look at the Maybach Mercedes Benz and say to Javier that I want the site to somehow integrate the interior and skylight of a Maybach Mercedes Benz vehicle. Being a web engineer, I might not have the aesthetic and design ability to draft a model of a site that looks like a Maybach interior. I might have the vision in my mind but not the ability to bring the model into reality. The benefit of working with an industrial designer is that that is exactly the job of an industrial designer—to create a design from someone's vision that can actually become real. The point is that I can take something from nature, entertainment, astronomy, emotion, or almost anything, and use it to inspire another artistic form. This is artistic inspiration at its best, and it's not exclusive to creating a website. The important thing is that having a vision, and someone with the artistic and design ability to carry out that vision, when creating website will make it easier to come up with a pleasing aesthetic design and layout.

Building

Building the website should be the relatively straightforward once all of the planning has been done and a model completed. It is the initial efforts: the concepting, design, discussion, planning ,and assembling the proper team that are the most time-consuming, so if all of these steps have been done appropriately, and provided what is being built does not include some newly invented technology, it should be fairly straightforward to build the site. If the site doesn't use many third-party APIs or closed source software packages, then building the actual site according to the blueprints shouldn't take too much time.

When Things Don't Go According to Plan

Unfortunately, sometimes things don't go according to plan and the timeline for a project is significantly delayed. Typically this occurs not because the engineering team doesn't have the know-how to get something done or because the money ran out, but because the expectations were not clearly communicated in the very beginning.

Generally this means that the defined scope of the project was not clearly communicated to the customer or manager who commissioned the site. This usually doesn't happen if there are good specifications laid out in advance, before starting any development work. The easiest way to address this problem is to communicate that websites are software, and software has a life cycle. Like any living thing, a website is composed of varying parts, software in this case, which might include a database, one or more applications working together, third-party APIs and feeds for data, and other kinds of integrations, and all of this is, at some level, code that is powering a dynamic website. Code is continuously being updated, and, therefore, a site is an ever-changing thing.

Research and Development Websites

Research and development sites are building something that has never been done before, and it is uncharted territory in terms of available technology. These kinds of projects aren't intended for production launches; they are mainly intended for creating some kind of new technology where there isn't a solution readily available. Any solution that's conceived is considered somewhat experimental, so no one has really explored the field yet. These kinds of projects, although they may have a timeline, typically never have a real or realistic deadline attached to them. Don't try to engineer something new unless it's absolutely necessary, or the time and resources are available to do so without interfering with regular business goals.

These kinds of projects involve the cooperation of many teams and groups to create a technology that is entirely new in order to solve some common problem or solution that benefits many people. Such cooperation usually can't be achieved without either a mutually beneficial or financial incentive, most of the time both. For example, the OAuth protocol for transferring login credentials between websites (for when a user wants to log into a website with his Google or Facebook account) was developed out of a common need. OAuth, in its infancy, was a research and development project, and developed over time through the efforts of large companies with many dollars to spend on development and a vested interest in seeing the protocol come to fruition.

Testing

The actual programming of the web project before it is launched into production will follow the standard software development life cycle rules. The difference in launching a new application into production is that there will be a much more rigorous emphasis on testing the applications and their individual components, according to the guidelines covered in Chapter 3. Here, though, performance of these applications is generally unknown, and there is no usage data around the application yet, so performance requirements are only rough estimates. Historical performance data can be taken into account from similar web applications to help refine these estimates. The main idea is that when developing a new application, setting rough performance requirements for the application is something that the special projects team must take care not to do. For example, the special projects team may say that a new web application that will provide users with pricing information for foreclosures on the real estate market needs to be able to support 500 end-user requests per second. Pulling this metric out of thin air and aiming for it arbitrarily without any business reason is a practice that could potentially waste time and valuable resources. If the business has a similar application that only requires serving 100 requests per second, it is unlikely that the new application will need to support 500 requests per second. Moreover, it may

be magnitudes more difficult to support an application that performs at 500 requests per second, and require twice the hardware investment to support this goal.

End User Testing

End user testing is something that should happen with a focus group consisting of different kinds of users and the target demographic. The focus group should be given the opportunity to test the site and provide feedback. Regardless of the tools used to measure a user's interaction with a website, users should be involved early on to begin turning wheels and knobs and providing feedback to the web experience team about how the navigation and layout of the website works and how easy and intuitive it is to use. This is something that should occur early on because once the primary navigation of a site has been set it becomes difficult to change. The best idea is to have at least three different versions of a user interface that users can test early, way before the launch cycle.

Performance Testing

Performance testing is a crucial part of launching new websites and new code. Thorough performance-testing initiatives can mean the difference between a successful and an unsuccessful launch.

Performance testing is especially critical when launching new sites and applications because there is no historical data about how the application performs. New technologies in application frameworks, platforms, and hardware may also be coming into play. Hardware changes at a very rapid rate, and newly released hardware might have drastically different performance results for an application from hardware that was ordered even six months earlier.

Performance testing should happen early on, and all components of the new product should be tested before starting development work. If new hardware performs at twice the capacity as hardware on which legacy systems reside, this will place less demand on the application architecture to yield the same performance results as that of an application developed in the past.

As soon as there is an accessible, functional web application, that new application should be exposed to end users by metering a small percentage of bandwidth to the new application (if it is replacing an old application) to see how the new code performs. This "testing in production" approach gives valuable insights into how the application will perform once production traffic is thrown at it. You can also simulate production traffic against the web application by parsing the history of the web logs and throwing it against the new application to see how it might perform in production. However, this is still a synthetic test and will not yield the same results as testing the web application with real traffic from real browsers or client applications from users on the public Internet. By metering the amount of traffic that hits the new application, or revealing the new application to a small subset of users on an existing website, you can get valuable insights into how the application will perform once it is fully launched and receiving production traffic.

Local Performance Testing

A web developer should set up an instance of the web application on a dedicated server that resembles the hardware and configuration of the environment where the new website, and the

applications and databases or data stores that comprise the site, will reside. It's not feasible to set up a production-like environment for every web developer. However, it is important that they have the best available resources that *closely resemble* the production environment they'll be deploying the finished product to. This might mean that the web developer has a workstation that is in a tower form factor, but has about the same processing power as the server that will be running the production website has. This ensures that the environment the application is being developed in is as close as possible to the production environment it will eventually be running on.

Another route to ensuring that the performance of the website or application is as close to the customer-facing environment as possible is to simply develop on a production-like test environment. This depends on how well the snapshotting schedule is going and how many pre-production or staging environments exist, but it will save a lot of time because you often can't tell from a local developer workstation what the true performance of the web application will be once it gets to a production environment.

Local testing can be done simply by using some automated tools or browser plugins. It is always best to use a real web browser to test web application performance because it will give a more realistic view of website performance. Most websites are dynamic, and automated tools such as Jmeter or Apache Bench, known as synthetic tests, do not render dynamic content such as JavaScript and CSS and may throw off website's response times. There is a tool called HammerHead that will reload a web page in Firefox over and over again and clear the cache, giving the web developer insights on how long it takes a web page to load. Firebug is another utility that provides handy information on how long it takes the web browser to render a web page, including all of the dynamic content.

If on a local test there's more than a one- to three-second page-load time, there might be something wrong with a non-graphics-intensive site. Most Internet users are notoriously impatient and are not used to waiting for things, especially with the proliferation and availability of broadband Internet. The dial-up days are over, and users don't understand that the database has to do some heavy lifting in order to render a web page. So when testing the web application, if it doesn't render within one to three seconds, you might need to do some trimming in terms of the amount of static content that is getting loaded or the amount of front-end work that needs to be done.

Caching

One potentially bad decision many companies make is to sign up with a content delivery network (CDN). A CDN typically acts as a reverse proxy for web content, so the CDN company configures web servers all over the country, much like a web performance monitoring company. Instead of having servers with a web browser checking how fast your website is loading on a periodic basis, the CDN actually stores a copy of your website servers all over the nation or world. The reason for having a CDN is that if your web server is in Toronto, Canada, for example, and someone accesses the site from Wichita, Kansas, the round trip time is the time it takes to transfer the data to load the site from Toronto to Wichita. In contrast, a CDN caches the content locally using a reverse proxy on a web server so that when someone accesses a site from Wichita, the response comes from a CDN company server in Wichita, rather than the origin server in Toronto, which significantly reduces response times.

Many CDN companies are now adding web performance best practices and baking them into their services, such a compression, minifying, or compressing JavaScript, HTML, and CSS content, and even adding a layer of web application security. These are excellent services to

have on a production website, and make its performance better, but engineers have to be wary about relying on these "canned" services to solve the problems and shortcomings in performance. Caching is an excellent way to accelerate and improve a site's performance, but it's not an effective way to fix performance problems that should be solved on the developer's local workstation.

Code Analysis

When new applications are being developed, the code moves at an extremely rapid rate and changes frequently. Even with senior development staff, there will be many more bugs and errors in new applications than in those that have been vetted in production over a period of time. Because of this, the source code should be reviewed by a third party. A QA engineer (a software developer whose primary function is to review the functionality of applications but who also has an understanding of applications at a source-code level), who is not involved in actually writing the software will produce better results in code quality even with the most aggressive of timelines for launching new web applications.

This kind of process and feedback loop will also help operations encounter as many issues as possible before the site or application is launched into production for testing. Operations engineers should be able to assist in determining where in the code problems lie and in communicating this to both the web developers and QA engineers, or even resolving bugs themselves and checking them in, depending on the experience and role structure of the operations engineer.

Production Testing

The performance of the application should meet its projected usage profile. If no more than 100 people are going to be using this new service at any given moment, it doesn't make sense to try and hit a performance goal of 300 simultaneous connections per second. Such arbitrary performance metrics are a waste of time and can introduce serious delays in the quality assurance portion of a new website launch.

Although as engineers and engineering managers, we want to get the most bang for the buck out of an application, this should not become the overriding focus of the team at the expense of end user response time. The only way to measure true end user response time is to set up servers all in geographic regions all over a country or the world and access a web page on a periodic schedule, such as every 15 or 30 minutes. This is known as real browser performance testing and it is a long-term approach to monitoring how a web application will perform. This is one of the most effective ways of understanding how a website or application will perform. Typically these processes are performed by a third-party company that run tests on behalf of its customers throughout the designated are. Keynote and Gomez are two well-known companies that perform this service. For most businesses, setting up this type of infrastructure would be too costly and a massive waste of resources for very little return. Thus, it makes most sense to leverage these services from companies whose core competency is to provide web performance monitoring and testing services.

Production testing doesn't necessarily mean launching a new product into production in order to see how it performs, which could wind up being incredibly bad for your brand and reputation if it doesn't go well. When you're introducing a new feature or making a change to a major part of an existing website, it's best to take a small percentage of traffic and present

the new feature or section of the site to those users. Once the application has been fully tested internally, it is beneficial to understand how production user traffic will cause the new application, site, or feature to perform. This practice must be taken with a grain of salt, though, because the feature is only being introduced to a small portion of users and doesn't represent the full load. However, it's useful as it provides the following data:

- Errors and behaviors that would otherwise not appear except under production load
- Popularity data: how many users are attracted and wind up using the new feature
- Performance metrics

This kind of practice can be achieved by funneling traffic using a load balancer based on a URL and a percentage of traffic. For example, the web or application servers that contain the new code might have a URL of /beta/player, and might be on a server farm on the 10.10.100.0 network. Most load balancers can be configured such that only a certain percentage of traffic or sessions will be served by application or web servers that contain the new application or module.

Once some performance and log data has been collected in a controlled production test setting, it can then be reviewed and compared with internal and synthetic testing signatures.

If this is not just a new feature on an existing website but rather a startup that has never launched before, testing is even more critical. What many startups or brand-new websites do is put up an invite page and ask a select number of users to try out their service. The point here is that these users are opting in and are well aware that they are serving as beta testers, to assist in determining what changes need to be made before an official public launch. The procedure of doing extensive internal testing and a limited amount of production testing, and comparing the results of the production usage and error data with the internal testing data are the same, except the method of limiting access to the new website is controlled by an opt-in list rather than automatically using a load balancer or a tagging system that identifies users based on some criteria, such as age.

Security Testing

Web applications, like any other software, are subject to attack and have weaknesses. Therefore, security testing is a crucial aspect of launching new websites. This is often an overlooked aspect of building and launching websites, but it is a valuable part of the process and can result in finding unexpected errors, unexpected functionality, and user experience issues *in addition* to finding exploitable aspects of the site that could lead to the system being compromised. Making security testing a part of the standard production launch process will prove to be incredibly beneficial and yield more information about the application than standard stress testing and user traffic monitoring can yield alone. There are many good books on the subject of "fuzzing" web applications, or applying random behaviors to software to see if it can handle these behaviors in a graceful manner or if it comes crashing down like a house of cards, and this process should not be overlooked.

Security testing should be a critical part of any new product launch and it shouldn't be an afterthought. It should begin as soon as there is an application that can be tested and continue throughout the development process and after the launch. Penetration testing efforts will be best led be a security engineer who specializes in detecting exploits and vulnerabilities in web

applications, and who tries to uncover security issues before the new application is accessed by end users. The Metasploit framework or WebScarab project are good examples of software that might be used in the penetration testing process.

Web App Scanners

Plenty of commercial or open source web application vulnerability scanners are commercially available, and rather than point to any particular ones as best, which may well be outdated by the time this book is in print, I would simply like to advocate their use. These vulnerability scanners can analyze the structure of a website or web application by crawling it much as a search engine would, and then applying various common exploits to the site. This not only finds whether common exploits are present in a newly developed site or app, but it also creates scenarios and usage patterns for the application that will often yield unexpected behavior that can reveal to software engineering teams areas they may need to improve in their code. The scanner is thus not only a security utility, but a quality assurance tool as well. No new software should be shipped or made available for public consumption until a vulnerability scan has been performed on it.

Integrating into QA Processes

Ideally, the vulnerability scanning should be automated and integrated into the QA process. Your quality assurance test suite should kick off a vulnerability scan after a new release of code to the web environment. This enforces a culture of security testing, instead of a practice that occurs only when there is a suspected breach of security. Moreover, security testing shouldn't be the responsibility of a single engineer in the organization, and the reasons and benefits of performing security tests should be understood throughout the company so that it becomes commonplace. By adding security testing into the automated, standard quality assurance processes makes it as familiar to all engineering teams as checking log files or server performance metrics.

Stress Testing with Load

One surefire way to get a flavor of how the new website will perform is to open it up to a performance monitoring service, and then perform a distributed load test against it. There are many different types of load tests, but they primarily fall into two categories: synthetic load tests and distributed load tests. A synthetic load test typically involves a server or cluster of servers in a single datacenter or geographic location, and they are blasted at the same time to see how the website or application performs under stress. In a distributed load test, there are multiple servers with different browser configurations distributed throughout a nation or the world, and recorded or realistic traffic is then "played" across a new web application. There are many ways to collect these recordings, but typically the access logs from a web server are used to determine which URLs of the application should be requested from the browser. This happens all at once from multiple servers so it simulates real user activity from multiple networks, browsers, and operating systems from different geographic regions. You never know ahead of time how an application will truly function in production. Real user behaviors and configurations when accessing a newly created web application are almost impossible to predict. However, the closer you can come to simulating real users the more accurate a stress test will be, and a stress test gives a lot of information on how the web application will perform on the top end rather than just when sitting idle.

Think about getting a physical at the doctor. While you're sitting down in a doctor's office, completely relaxed, your blood pressure might be low, your breathing normal, and your heart performance good. But if the doctor hooks you up to a treadmill and takes the same measurements, the results are likely to be completely different and may suggest you refrain from hiking next weekend and instead stay home and take it easy. It's the same for website performance testing; without knowing how the site will perform when the going gets tough, there will be no ability to predict the site's behavior after launch.

Stress testing is valuable because:

- It can surface errors that would otherwise never occur under normal load.
- It helps in capacity planning efforts.
- It sets a baseline by which to measure historical performance trends.

Logs

Viewing application and web server logs after performing stress tests will provide a wealth of valuable information, especially regarding errors that result from the stress test. Knowing about these errors will prevent much heartache during launch time. The act of stress testing can cause problems with underlying web applications and data stores to surface, or they may fail entirely. Open source tools like Logstash allow web developers and operations engineers to take a look at the errors and correlate error messages to events such as a spike in page-response time.

Why is this so critical? When an established company launches a new product, feature, website, or application, the whole world is usually watching. In such situations, the success or failure of the new product can strongly contribute to the company's success or failure. Take MobileMe, for example: When MobileMe was first launched, the service barely worked for the first couple of weeks and teams within Apple didn't leave the office during that time, trying to solve the problems. In fact, during the upcoming months the service had many failures and, as a result, got a lot of bad press. This didn't build a lot of confidence in the service and although Apple was doing very well, it may have had an impact on users' confidence in the service and in Apple's online services in general.

For some businesses, a bad launch may not make too much of a difference. For a startup making its first public launch, it could mean the end of the company. That's why it's so important to understand performance both during idle times and at maximum capacity. Production usage is so random and varied that it is incredibly hard to predict, so taking the time to gather as much information as possible by "kicking the tires", and even "taking a sledgehammer" to a website will help to ensure that a public launch is successful, and that users accessing a service for the first time will come back for a second visit.

Prelaunch

The prelaunch phase should be used mainly for planning and coordinating how the various teams will work together to fully deploy the new software application. Having a checklist that details what needs to happen in sequence to ensure a successful launch can make the difference between failure and success. The public relations and marketing teams may have already scheduled press releases and other notices to the media about the launch of the new product,

and there may be a great number of eyes on the application on launch day. It may be a better strategy to distribute press releases only after deploying successfully. Regardless of the marketing strategy, however, having a detailed, step-by-step plan for launching the application into production is crucial. The plan is also dependent on the way the website or application has been architected. Some applications will need to be deployed and started up sequentially; others may have a more asynchronous design where this is not a requirement. A launch plan that is well thought out and details the coordination between the various members of the team launching the new product contributes significantly to the success of the launch and could mean the difference between the success of the application or its doom.

Having a backup plan in place will also help mitigate the risk associated with launching a brand-new website or application. It can potentially save the company from the criticism and disappointment of users in the event of unexpected server crashes or application failures when the new site or application is hit with production traffic for the first time.

The Dark Side of Launching: Common Pitfalls with Personnel

One of the most commonly ignored aspects of the launch process is the mental and physical health of the engineering staff. This is as critical a part of the release process as having code review and testing, and should to be mentioned because when it is ignored, launching new projects and products *will* fail.

Don't underestimate the stress involved in launching a new website or application. When a new product is to be launched where tens, hundreds, or millions of dollars have been invested, it becomes critical to ensure that the *people* responsible for the application, who are taking the project all the way from idea to end user, are in good shape. It is an incredibly stressful process and can take long hours for major releases (even if things are structured to be done in small releases) and the full involvement of all stakeholders, including investors, business management, development, and operations, is necessary. The staff needs to be treated from a scientific and engineering stance. If the people developing and administering a new web application are performing poorly, the launch is guaranteed to have more errors and end up costing the business more money, or failing entirely.

Lack of Recognition

People want to know that they are putting in long hours and giving up a significant portion of their personal lives for managers who appreciate their efforts. Nobody is inspired to put forth his or her best effort for management that sees the project as just another way to make revenue and meet a deadline. That kind of domineering mentality is the perfect way to make people want to give up on a project. People who know that their individual efforts are recognized are more likely to put forth the sustained effort required in launching a production website.

Lack of Staff

When undertaking a new software project, management should be prepared to staff accordingly. Short term projects, such as launching a new module of a website or redesigning some aspect of an existing application, are best approached with a team focused and dedicated to that task and

that task alone. A better product is produced when some development and operations teams work only on maintaining the existing production websites and services, while others are dedicated to building new sites and applications to support the growth of the business and other business initiatives.

For this reason, it is imperative that management correctly assesses the available engineering staff to ensure there are enough to take on a new project. Without taking current staff capacity and abilities into account, people can easily become spread so thin they can't accomplish either normal maintenance or building new products, and quality and output might be compromised as a result.

Keeping track of the number of simultaneous projects, the next major release of a new app or website, and the available staff, and allocating those people to projects appropriately, will result in higher output and a better quality product.

Lack of Sleep

Lack of sleep is perhaps the greatest threat to productivity when launching a new website or application. Studies show that lack of sleep can not only affect an employee's output, it can also lead to accidents and affect judgment[1]. Too great a load placed on engineering staff will almost certainly affect the amount of sleep they get. The best engineers feel very committed to the projects they are working on and almost internalize those projects, and they will work on them almost nonstop. There are some engineers who need to be ordered to go home because they become committed to their work. These engineers are the high producers and especially need to be told to go home and rest because they won't do so on their own accord.

In the world of Internet and web companies, there is a late-night culture in which engineers are expected to work very long hours, and in some extreme cases almost live at work, and event to log on from home after spending 8–14 hours at the office. This may be necessary in special circumstances, such as beating a competitor to launch on a product, but making this a cultural expectation within an organization is a catastrophic failure waiting to happen because of the toll it takes on the engineering staff.

Ensuring that teams are properly staffed is the first step toward resolving the problem of a tired workforce. If economic conditions don't warrant hiring new personnel or contractors, the organization should take into consideration which projects have the highest priority and put their efforts into those endeavors.

Management should make sure that engineering staff isn't spending unnecessary time on work-related projects. Many engineers will spend time on work-related projects because they feel that if they do not, then they are neglecting their responsibilities, even though they may well be neglecting their personal responsibilities outside of work. This applies especially with operations engineering personnel. It is management's responsibility to drive home the cultural aspect to their engineers that staying late unnecessarily actually harms the business rather than helping it.

Cultures that reward the overachiever, the person who is always working and never takes a break, can actually undermine the accomplishments of those who go above and beyond in a more disciplined way. That kind of culture results in people who get burnt out rather than produce over the long term, and who almost invariably add to the problems and errors in a launch.

---

[1] http://healthysleep.med.harvard.edu/healthy/matters/consequences/sleep-performance-and-public-safety

Successful Launches: Preventing Burnout

Preparing for the launch is one of the most important parts of a project and it involves consideration of human abilities. For a company's efforts to succeed, it must take into account people's physical and mental capabilities during the process of a new product launch. Unfortunately, some companies lose a large amount of their staff because they treated them poorly during the launch by overworking them and not maintaining morale. Management often wants nothing more than to meet the deadline and get the new product launched, while engineering teams want to spend some time with their families and perhaps have time for something other than work. The two goals are often in direct conflict, but they don't have to be.

Dedicated Teams

The most important part of preventing burnout on a project is to have enough staff assigned to a project, and to allow them to focus on just that one project. It's not really possible to multitask—it just dilutes focus and is inefficient, and trying to make people develop multiple websites or major web products in parallel can lead to burnout and low morale. Therefore, it's best to have a dedicated team for each product or site. In a large organization, it might be best to rotate software engineering and operations engineering teams with sub teams whose focus is to primarily build and launch new products. There could be a maintenance staff whose job it is to permanently support websites and applications that are already in service, as opposed to those that are being developed, and a separate, dedicated team for creating and building new sites and applications. A mix of the two might actually give the best results because those busy maintaining existing applications often know best how new features and sites will fit together with what is already in production service. This way, those building the new site or product aren't out of touch with what already exists, and products and services that aren't generating revenue or positively benefitting the company can be deprecated and decommissioned, and those resources devoted to new efforts.

Rotation of Special Projects Teams

The developers and operations engineers on the special projects teams should be periodically rotated out, on a monthly, bimonthly, or semiannual basis, depending on the length and scope of the new development project. This helps the engineers on the special projects teams keep their skills sharp by continuously solving problems and then having new problems to tackle. Approximately 50 percent of the special projects team should remain on a project to avoid so much loss of knowledge that the new project begins to fail once it has been completed. This 50 percent might be team leads and will represent the "base" of the special projects team.

The other 50 percent should be integrated into the general population of web developers and operations engineers who are maintaining the core properties of the business. This is to ensure that engineers who don't have hands-on experience with the new sites and applications work directly with those that do, which is the most effective method for transferring knowledge and producing good documentation.

Planning for the Worst

Project managers should build in time for major catastrophes on a web development project. For example, if a key stakeholder, such as the director of software engineering has a medical issue

that takes him down for a few months, it can derail the projected launch dates of a new product. This may impact business deals and contracts as well as legal obligations that are in place, and cause tension all the way down the line. If extra time has been built into the development process, it's more likely the company can avoid severe consequences. Of course, it doesn't even take a massive catastrophe to impact the timeline of a project; many small delays can add up to completely derail the timeline. Project and product managers and engineering staff all need to be on the same page about what is achievable in a given amount of time. Three months is an ideal timeframe for launching a new web application, and six months for an entirely new website. These are loose projections and will vary depending on budget and personnel resources, but for the most part people tend to lose interest or a competitor will beat you to the punch if it takes any longer. No matter whether they're successful or not, they've still stolen your thunder. Avoiding an over-aggressive schedule will benefit the whole development team and ancillary groups as well.

Maintaining health and morale during the processes of launching a new software product into production will actually have an accelerating effect, and will benefit the entire company and organization in the long term. By setting reasonable timelines and not overworking people, teams will be able to more effectively meet their goals and feel better physically and mentally, and this will help the company retain its best talent over time, which is more important than getting *one* website out the door which *may or may not* produce any revenue or benefit to the company.

Keeping People Energized During a Production Launch

I worked at a company where there were a significant number of deaths during the few years I was there. Whether it was related to overwork is a subject of much curiosity and debate amongst myself and former colleagues. In any case, though, not making employee health a top priority during production launches will almost certainly negatively affect the overall success of the launch.

Building software and launching a new website, redesigning an entire site, or adding complicated functionality to a site are all much like running a triathlon. Expending a lot of mental energy can make someone feel as equally drained as expending physical energy, if not more so, and may even take longer to recover from. Moreover, discontent among software and operations engineers can be contagious when people are being mistreated and can have a domino effect in the work environment. If this happens while a company is trying to get its latest and greatest product launched, it could spell catastrophe for the effort, with only a few dedicated individuals shouldering the bulk of the project.

Success Metrics

A lot of planning, discussion, and research needs to take place before launching new software for production use, and a gradual approach is often taken so as to not risk the company's brand too much. If everything has been executed properly, the number of bugs in the production code will be minimal; the morale of the teams that worked toward and built the project will be high through to the end; and most importantly, there will be a good response from the end users.

End users are the main success metric by which to qualify and quantify a new launch of code into production, and the number of hits is the first indicator of success, though there may be a delayed response until the feature or website shows up in the media, or until new users discover the product and begin talking about it, thereby causing more people to take a look and try it out.

You'll also want to take a look at metrics like how many users explored other parts of the site after the new feature launched. When you add new functionality to an existing site, it can often encourage users to take a look at and try out other sections of the site they haven't tried before.

Complaints and feedback are others ways to measure the success of a production launch. If users are having difficulty using a new feature or new site, and some kind of feedback mechanism on the site, such as a trouble ticket system or customer service e-mail address, lets them ask for help, there will be increasing questions about how to do certain things on the site. A large increase in such inquiries may indicate that certain user experience and design considerations are causing difficulties. Keeping track of the various metrics in a centralized dashboard can provide visibility into the success of a launch. Quantifying success metrics of a production web launch is indeed difficult, and when the systems of measurement are all disparate, it becomes even more difficult to centralize metrics. Doing so, however, will have a tremendous effect in showing how successful a launch was, and will continue to do so over time. There is a certain cadence that comes to launching new websites and features that involve not only having good technology, such as automation, source code management, and deployment mechanisms, but also the way and attitudes of how teams plan, test and implement new launches. There is a greater sense of urgency, and a greater chance of unexpected events occurring when launching software into production, and thus a launch needs a different approach from just maintaining existing code or monitoring an already running production site.

Making a Launch a Success

In short, launches are not often easy. However, the following five steps should help guide you through the process:

1. Continue to remind people how valued their efforts are at the company and on the particular initiatives. This should come not just from direct managers but from every level of the company from the top down, including executives. An annual pat on the back from the CTO or CEO isn't enough to maintain morale.
2. If you can, have an annual company sponsored health checkup during a particularly trying week.
3. Set realistic goals for project management.
4. Reduce the number of meetings you have. Daily meetings are a waste of everyone's time and they decrease both productivity and morale ("not another pointless meeting").
5. Provide outlets that encourage people to get away from the computer and encourage physical exercise, as it boosts people's mood. Happy software developers mean happy code and launch times.

Summary

Launching new web apps and web sites requires all hands to be involved and actively working together for an extended period of time. Having a proper plan and distributing work and

allocating enough people to the project will result in a better product, with deadlines being met. Although many elements of designing and testing the software to support the website are the same as in maintaining an existing application or extending its functionality, a greater emphasis is placed on these aspects when performing a release for a new website or app.

Many of the human aspects of a release are overlooked in IT organizations, and they are in fact among the most critical factors in ensuring a successful launch. What this means is that managers and management must take into account the welfare and health of the employees, even at the expense of deadlines or goals. Every business needs to operate, meet deadlines, and turn a profit, but what is sometimes misconceived is the idea that organizations need to overwork people in order to achieve these goals, when really the contrary is true. Making sure there are enough people to work on a new project, and making sure it will be adequately maintained once it is launched should be integral parts of the planning phase of a launch, and this can be estimated by looking at the number of hours people are working on existing projects, how much maintenance the projects require, and how that might affect staff being available to develop future projects. Timely consideration of all these issues will make future launches go all the more smoothly.

CHAPTER

7

Mobile Web Integration

Knowing the habits of mobile users of a website is tantamount to developing a mobile integration strategy for that site. If you don't have a lot of mobile Web traffic in the first place, this is not something you have to worry about, especially if you have a small business site with fewer than 5,000 unique visitors per month. At this level, it's unlikely that the site is the main focus for conducting business. When the number of page views reaches tens of thousands, hundreds of thousands, or millions per month, however, it's crucial to develop a mobile integration strategy, and this means you need a good idea of what your users want out of a mobile experience.

In this chapter we'll take a look at how mobile users interact with an existing website, and how to better track that behavior. You'll want to have a good understanding of what mobile users tend to do on a site when you're thinking about how to integrate a mobile app or website into an existing site. Of course, there are many different ways to do this. Some pre-canned solutions provide a quick, simple solution for getting a mobile Web site up and running. More advanced solutions make a site available via an API. Such solutions are more complex, but they are generally far more flexible and give you many more options.

Different Devices, Different Experiences

The number and variety of mobile devices never stops increasing—and that can be a problem for a mobile website: You need to ensure a consistent user experience across all devices. Some mobile operating system manufacturers keep properties across all of their mobile devices consistent so that, for example, the device's screen size never changes. This makes it easy to achieve a consistent user experience for a mobile site, regardless of the device or operating system. There may be only one or two different screen sizes and a limited set of options for changing things. Other mobile operating platforms may have thousands of devices and screen sizes with a high level of configurability, making consistency far more difficult to achieve.

Knowing which devices to target is tricky. Companies don't always have the development and QA resources to test their mobile applications on every possible device. There are some services, such as Keynote and Gomez, that allow automated testing on real mobile devices using real carriers, but these are more for tracking how a mobile application is performing after it's in production rather than determining which devices to target.

What you need to do is keep track of which devices visitors are using to access your site with, and continuously add new functionality as new devices are rolled out. Keeping up with what your users are doing will help you stay engaged with the users that visit your mobile website.

Mobile Web Limitations and User Expectations

The limitations of the mobile Web environment versus the expectations of mobile users should be understood before venturing into creating a mobile version of a Web site or integrating a native application. Here's what you might have to contend with.

User Impatience

Patience is a virtue, just not one employed by mobile users, who are often more impatient than standard users. In general, mobile users expect an instant response, so any problems they experience with the design or interface of a mobile Web site or application may keep them from coming back.

It has been found many mobile web users will not wait more than 10 seconds for a page to load. Most mobile Web pages load as fast or equal to their desktop computers, which means that the engineering teams supporting a mobile application need to achieve speeds on par with a desktop experience while overcoming the latency associated with wireless data networks[1].

Here are some suggestions:

- Don't overload your mobile application with large images or huge uncompressed audio or video animations. Users might like them, but not at the expense of a responsive experience.
- Use a native application if you have many static assets, such as images and audio files or other non-streaming media. Keep in mind that, currently, mobile Web browser cache sizes average around 4 MB, not a lot of room for storing objects.
- Cache, cache, cache important data when the user has a good connection so that if connectivity issues or bad reception occur they will still be able to interact with the application with the most important functionality intact.

High Latency

Mobile applications run on mobile devices, which run on some kind of wireless network, whether it be WiFi or cellular data networks. This means that mobile applications have a disadvantage in that they are always dealing with a high latency network, which increases the response time of mobile web sites and apps. So that you do not annoy your users and to ensure they do not blame your application for slow network speeds, its best to keep the user informed when slow network problems arise.

---

[1]Gomez Inc, / Equation Research, Whitepaper, Lexington, MA, 2009: Why the Mobile Web is Disappointing End Users

Here are some suggestions for overcoming mobile network issues:

- Keep the user informed. Slow network speeds can make your mobile application look like it's performing poorly. The best defense is to set timeout values on the application, and then advise the user that the Internet connection may be impacting the experience. There's really no way to make the network go faster, but telling the user what's going on is one way of keeping him from blaming your site or application.
- Store what you can. With a native application, you have the advantage of storage space. Also, HTML5-compliant mobile web browsers that support local storage for data and WebSQL Database can keep data offline. Using images stored locally in the application and then refreshing them periodically when the device is connected to the network is one way to avoid poorly performing networks.

Understanding Usage Patterns

Mobile users tend to behave differently from desktop web users. They typically want to retrieve some information and be on their way, unless they are using a native application that is highly interactive, such as an ecommerce service or a game.

For many startups and small companies, it may not make sense to have a mobile presence whatsoever. Be sure to research to what degree a mobile website will be used. Mobile use is expanding at a rapid rate, but that doesn't mean it's necessary. The key is to examine your company's internal usage patterns as well as competitors if you can.

Gathering information about how mobile users are currently using a standard Web site will yield some information whether or not a mobile presence is required and if it would expand the company's web or online presence. Looking at data, such as access logs and Web analytics, and measuring the amount of mobile user traffic to the website will gauge if you need to invest in mobile and to what degree. For example, if 20% or more of users are accessing a web site via a mobile device, then it would be a strong case for launching a mobile version of the website. If you have features that need to leverage the power of mobile hardware, or very graphics intensive applications, such as games, then it would be prudent to consider accompanying a native application as well with the mobile web site.

Note You may not have an existing website to examine. In this case, you may be developing a mobile app independently of any website, in which case you'll want to use in-app analytics to determine mobile application usage to understand how to give users what they want.

When we talk about usage patterns here, what we mean are the ways users have accessed and interacted with a given website or mobile application over time. By examining these historical patterns, you can get some ideas about how users want to interact with your site and what features are important to them. This is something you definitely want to do before developing a new mobile application. Some questions to answer might be:

1) What are the most frequently accessed portions of your standard website?

Determining which parts of your site are most commonly accessed can help you design a mobile application or a mobile version of a website or native application that's truly useful to

your users. Jamming a mobile site with all of the features or functionality as the primary site typically diminishes ease of use and the overall user experience.

Historical data about the most popular features and sections of a website can give you a firm base for deciding which features and functionality to include on the mobile site. Moreover, a more tailored site or application won't confuse users, while still giving them most of the functionality they need when they are not at a desktop or don't want to open their laptop. It also helps to reduce the complexity of creating a mobile version of a site, and it decreases costs, effort, and overall painfulness of creating two vastly different user experiences.

2) What percentage of my existing user base is accessing the website using a mobile device?

It's crucial to know how many users are accessing a website using mobile devices before investing time, money, and effort in building out a mobile solution. When mobile traffic to an existing site reaches a good percentage of the total traffic, you'll want to have your effort well underway, though, so you should monitor the traffic regularly. What constitutes a good percentage of the total website traffic is going to be different for every company, based on its objectives and user base, but overall, there should be some indication that a substantial number of users are already accessing a site via their mobile devices.

3) How will having a mobile website drive sales or attract more customers, or is it simply a branding or image play for the company?

One of the first steps to take before developing a mobile website or application is to clarify your business objectives. Having insight into how users are currently accessing a standard website and their usage patterns will help in achieving business goals or further monetizing a site when rolling out a new mobile application that extends the features of a mobile website. If driving sales is an important consideration, you'll want to focus on areas that help users shop easily. If sales is not the primary goal but enhancing the company's image is, the focus might be making content available to users. Regardless of the goal, there should be a clear and definitive understanding of the features that will promote business objectives.

Some mobile websites and native applications might be published by a company simply so they have a mobile presence and may not be well thought out at all, acting as nothing more than a simple RSS reader for blog content and the like. Although this may be convenient for a reader, these applications are really nothing more than marketing devices. They are fine and may enhance a company's brand profile and make them appear current, but truly taking the time to understand how a mobile website could attract and retain more users and enable them to further engage with a primary site will result in greater user retention and satisfaction.

4) How will the mobile website make users' lives easier or more enriched?

Mobile phones were invented so people could keep in touch without having to be tethered to a particular place. In the 1980s and early 1990s, most people were stuck using pay phones or pagers when they were on the go. The cellular phone was a marvel of modern technology that allowed people to communicate even when they were nowhere near a regular phone. It was shortwave radio for the masses, and the mobile telephone industry exploded.

With smartphones, a computer was essentially fitted onto a device with the form factor of a mobile phone, complete with its own operating system and application development platforms. It's impossible to exaggerate the effect of this, in terms of the convenience it brings to people and companies alike. When setting out to develop a mobile website or native application, it helps to consider from the very beginning how it will improve someone's life. Will it be a fun game, will it help people stay safer, will it help them save money, will it improve their family life, or will it

simply allow users to interact with the desktop-based website from anywhere? It doesn't matter what the mobile website or application is for, as long as it's clear:

- What it intends to do that users will need or want.
- How it will interact with the primary website, if at all.

Keep in mind that mobile users are dealing with a reduced amount of screen real estate and don't want to zoom and scroll around much. Because there is less screen space than on a desktop, a simplified version of the website or application should be presented to mobile users to ensure interaction is easy and not confusing.

Native vs. Mobile Web

A hot debate is whether or not to use a mobile device's native programming language (think Objective-C for iOS or Java for Android) and distribution techniques (app stores, manual installation of a binary, and so forth) or to create a mobile website using a universal, cross-platform format that's available on most mobile devices and doesn't require the installation of an application binary via an app store or other method. Many mobile software platforms constrain users to a certain set of legal, business, or software requirements in order to use what is called native functionality for a given platform. This means that users are restricted in what they can do and what they can install in an application. The Web, on the other hand, doesn't have these limitations because of its inherently open nature. This is why building a mobile website is often favored.

Mobile web browsers, however, are the limitation when it comes to mobile Web applications. Although mobile browsers are advancing in terms of what they can functionally do, they can't and will most likely never get as close to the hardware as a native programming language or application platform. To truly harness the power of hardware, there needs to be fewer layers of abstraction between the application's code and the device on which it runs, which includes access to hardware features, such as GPS and the camera.

During the process of developing a sound mobile strategy, it's important to take into account the more advanced features that can benefit both users and businesses. For example, geolocation information that comes from using the GPS on a mobile device can be used to enrich both the desktop website experience and the mobile experience. A company could, for example, post a contest to get users to take pictures of themselves with the company's product on their mobile device, and then have a mobile application upload their pictures to their primary website. Using a mobile website or native application to enrich the standard desktop website experience is a common practice. Thinking from the start about how to take advantage of the advanced capabilities and mobility of a mobile device, and how people leverage these when they are out in the wild will prove advantageous.

Building a Consistent Experience

Building a consistent experience across all ways of accessing a website, whether through a mobile browser or a desktop browser, can work only if the design and interface of both are as similar as possible. Having completely independent systems that don't integrate with one another can confuse and frustrate users. There are many different ways of creating a consistent experience across standard desktop browsers and mobile and other devices. Let's take a look at three methods for doing so: conversion, native apps, and APIs.

Conversion

There are frameworks and services that can take an existing website and automatically generate a mobile version. This might be a good option for very simple sites, but doesn't work so well for large sites with a lot of content, specific design requirements, or advanced functionality. The most common way to do this is through a third-party service, as shown in Figure 7-1.

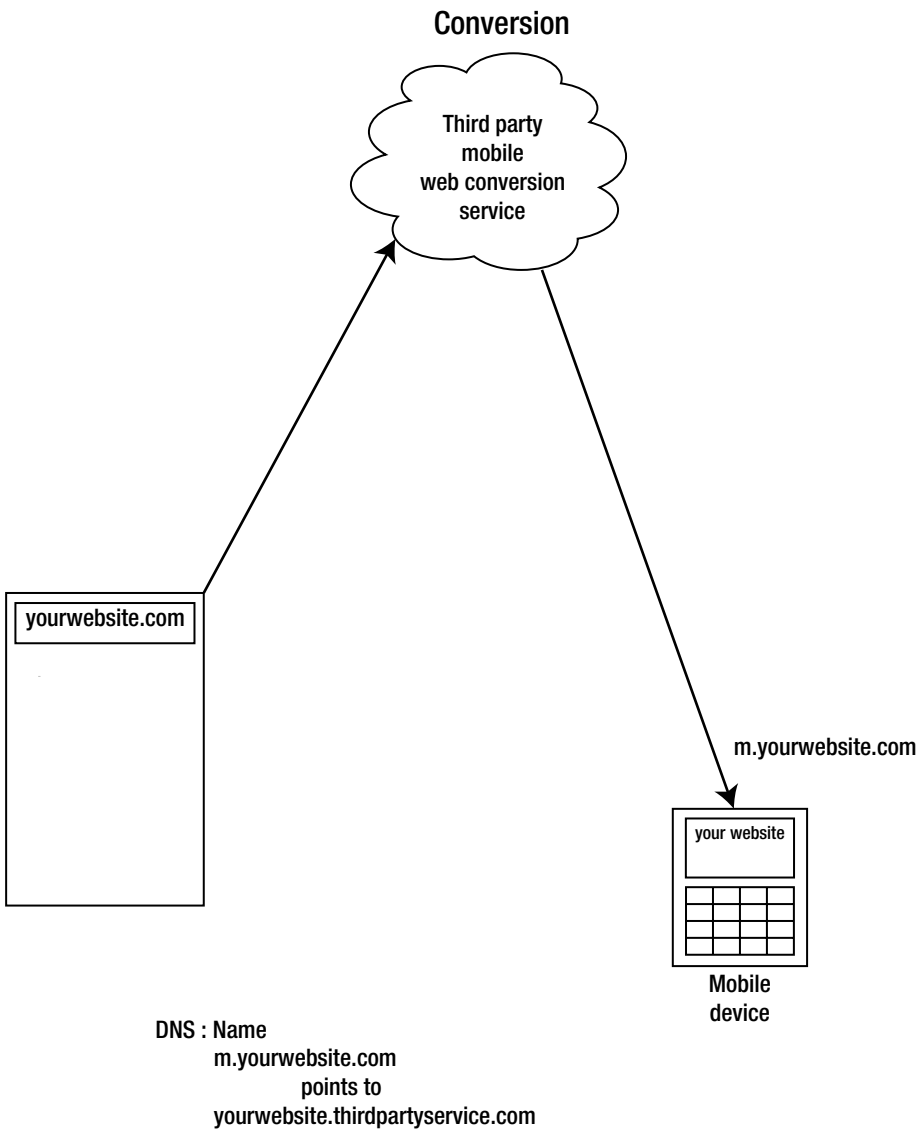


Figure 7-1. *Converting a standard website to a mobile version using a third-party service*

Using a Native App for Integration

This type of integration exposes services from a website via a private API, making those services, such as search functionality, available to a native application. This allows the mobile device to become a powerful extension of the Web site.

Suppose we have a real-time chat application, which takes input from the mobile device and sends that information back to the main chat server (such as ejabberd, the Erlang based XMPP chat server), via the XMPP protocol. Then the chat server holds the information until it is sure that the other user is connected via their mobile device to receive chats. If that user is not online then it will retransmit at a better time. This is an example of a good use case for a native mobile application. Creating this kind of solution using the Mobile Web is not as well suited for the task, although it is possible using mobile Web technology. In this example, we would rather use a native application because of its tighter integration with the chat server back end getting the benefits of a native application. This would be possible in a mobile Web application; however, it would not perform keeping a history of the chats as well as a native application, and would not have as good performance as some strong reasons not to use mobile Web in this case.

Integration via an API

With this type of integration, the rendering of a web page relies on an API, and the mobile version of that website ties into the API so that building a mobile version of the site doesn't require additional architecture to support it. Only the presentation layer of the website is different, but content for the page is the same or similar for both primary and mobile sites.

This is the most elegant way to integrate a mobile website with a primary website or web application, by creating a source API where all information and functionality is stored for interacting with the data set, and both the mobile and standard sites and native clients all interact with this API to build and render pages. In most cases, creating a subset API for the mobile web and native clients to interact with is the easiest solution for building a comprehensive desktop and mobile combination, but some companies build their primary desktop-based site based purely on an API, and the desktop site is merely a consumer of the same API as the mobile Web and native applications.

The reason this is the most elegant way of achieving both a mobile and a desktop version of a website is that having one API to serve both sites reduces the amount of work and complexity on the back end. Twitter.com is a perfect example of a company that builds its desktop-based website from the same API or data source as their mobile web and native applications. This means that upgrades and changes to functionality can be done in one place and, provided the clients are updated as well, have the same functionality and access to data as the primary Web site, which reduces the effort involved in keeping mobile, native, and desktop codebases in sync.

One of the methods for generating a mobile version of a Web site is to use an underlying API of the primary site to generate a mobile-formatted version of the existing website (see Figure 7-2). This ensures that the functionality of the existing site and the mobile version is consistent, and that there's no need to rewrite the entire application for mobile. The website might be powered entirely by an API, calling multiple methods to compose a given web page. This is a powerful method of breaking off the presentation layer of a site from the business-logic layer allowing for a single data source for rendering the regular website, as well as mobile websites and native applications. Witter is one example which uses this method.

With such an API, you can use any combination of methods to publish data to mobile consumers.

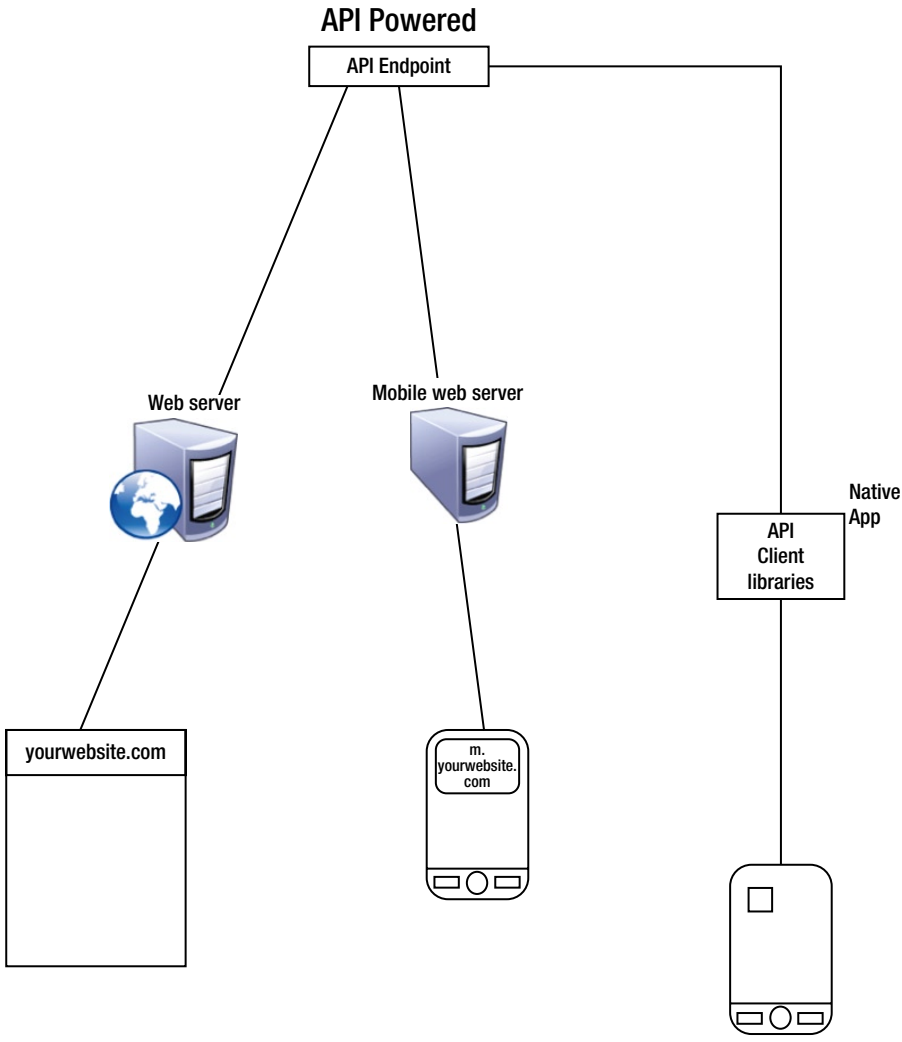


Figure 7-2. *Both mobile clients and the standard website accessing data via an API*

Tracking API Usage

When mobile websites are leveraging an API, there tends to be something of a disconnect between users accessing content and data through a mobile API, such as with a native mobile application, compared with users accessing content and data via a mobile website. In such situations, it helps to have some kind of analytics on the number of API calls or the types of data accessed. There are a number of third-party services that can provide this information, but these kinds of metrics can also be easily collected when using RESTful APIs by measuring the

number of HTTP requests to a certain endpoint. For example, if you want to track the number of times a user has registered, you could sort data based on the access method, such as from a native client, and then track that usage by putting the data into a graph or dashboard of some sort for business users. There are also software development kits such as Flurry (www.flurry.com/) that track how users are using applications by instrumenting the application rather than tracking access at the API layer. On the back end, API access and statistics should be monitored as well. You can take advantage of application management systems that do this, such as New Relic (www.newrelic.com) or Hyperic (www.hyperic.com). You'll find many different analytics platforms available for tracking user behavior within a mobile application or mobile website as well as on the back end API.

Summary

Once you've decided your company needs a mobile presence, you need to think about developing a strategy for integrating your company's primary website with a mobile website or mobile application. One of the most important steps is to take a good look at the people who visit your existing website. You will want to understand exactly what they are doing on your site, what parts they access most, what features they are using, and what kind of devices they favor. When you have a good picture of all of this, you can begin to consider what type of mobile experience will give users what they need without overwhelming the lesser capabilities of the mobile devices. Consistency between the primary and mobile web sites is paramount—not only does this make development and updating easier, but it also minimizes user confusion as they move between the primary and mobile sites.

Index

M

N

O

P, Q

R, S, T, U, V

W

X, Y, Z

Pro Website Development and Operations

Streamlining DevOps for Large-Scale Websites

Matthew Sacks

Apress®

Pro Website Development and Operations

ISBN-13 (pbk): 978-1-4302-3969-7

ISBN-13 (electronic): 978-1-4302-3970-3

President and Publisher: Paul Manning
Lead Editor: Ewan Buckingham
Technical Reviewer: Patrick Debois
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Katie Sullivan
Copy Editor: Sharon Terdeman
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code.

Contents

Foreword

We've all been there: Long lead times, onerous procedures, failed deployments, rollbacks, roll-forwards, middle-of-the night alerts, features that "worked on my machine," emergency updates, all overlaid with a culture of blame. This will sound familiar to almost anyone who has been involved in the roll-out or maintenance of a production software application.

There always had to be a better way.

The idea behind the DevOps movement is simple enough—bring everyone with his or her own unique skill set to the table, break down any barriers to success, and work together toward a common goal. The Agile movement laid the groundwork and has been widely successful in improving communication between business stakeholders and engineering. DevOps applies the same ideas to the traditionally isolated teams who perform development, QA, and operations.

The DevOps movement started in 2009 and began gathering steam in 2010. Those behind it must have be on to something, as it continues to grow rapidly in both recognition and attempts to put it into practice.

Growth has also been fueled by a perfect storm of technology factors in recent years that pushed developers and system administrators to work together more closely. As Patrick Debois describes: "Virtualization enabled ops to spin up new environments very quickly, and the cloud did away with the resource problem. The real differentiator came from two concepts: configuration management, and infrastructure as code."[1]

It is no surprise to me how big this movement has become. I've worked for years on improving application delivery processes and centralizing project knowledge and feedback, so when I first heard the term "DevOps" and the problem it was trying to address, it instantly made sense. It was one of those "a ha!" moments.

Reducing or eliminating the gap between development and operation is of incredible importance to businesses as they seek to not only deliver new applications and features to the market as quickly as possible, but also operationalize them at the stability and scale that the modern Internet demands.

Though this is a simple idea, it presents a huge cultural challenge for most organizations, particularly those that are large or have deeply entrenched processes. Moreover, even though the problems DevOps is solving are often clear enough, many struggle to understand what it really means to apply it to their organization. As was the case with Agile, there is some debate about the "right way" to do DevOps, and various groups have put forward conflicting ideas as they pursue their particular agendas. Nevertheless, certain basics are beyond dispute.

---

[1] Cutter IT Journal, Vol. 24, No. 8. August 2011.

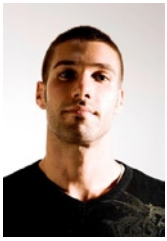
This book focuses on the fundamental concepts readers need to identify problems within and between their teams and the best approaches for addressing those problems. It recognizes that each situation is different. Rather than describing a particular methodology, it arms readers with the insight and practices to apply in their specific circumstances, and the knowledge of what challenges and outcomes they may face during the process.

The book also presents a number of useful experiences shared by professionals from well-known organizations that have already confronted—and overcome—these challenges themselves.

Dear readers, I hope you will be inspired to pursue these ideals within your own organizations.

Brett Porter
CTO, MaestroDev

About the Author



Matthew Sacks (http://www.matthewsacks.com) is a system administrator and programmer specializing in highly scalable Web sites and applications. He has also worked as a Java and Python programmer. He has presented at USENIX LISA and ApacheCon and is the founder of the USENIX Blog team.

About the Technical Reviewer

Patrick Debois has been working on closing the gap between development and operations for many years. In 2009 he organized the first devopsdays.org conference and, thanks to that conference, the world is forever stuck with the term "devops." He stays actively involved in the devops community by sharing a myriad of technical and cultural ideas and always encourages others to do the same—especially about employing devops ideas within traditional IT enterprises and shifting people to a collaborative mindset to achieve better result.

Acknowledgments

Thank you to my family and to my girlfriend. I could not have done this without your patience and support as I worked long hours into the night. Thanks as well to my mentors, Safdar Husain, Blake Swopes, and John Martin. Your patience in coaching me taught me to work slowly and think things through to become more efficient. I'd also like to thank USENIX and the Apache Software Foundation for allowing me to make a modest contribution to your organizations. And lastly, I'd like to thank you, the reader, for purchasing this book.

Thank you to my contributors: Tom Limoncelli, Rik Farrow, Paddy Hannon, Aslakey Hellesoy, Santiago Suarez Ordoñez, Brett Porter, and all who have helped this book become a reality.