Extracted from:

Pragmatic Project Automation

How to Build, Deploy, and Monitor Java Applications

Chapter 1

Introduction

This is the book your computer didn't want published. Until now, your computer has had a life of leisure: reading email, displaying web pages, and maybe even compiling Java code. Meanwhile you've been on the treadmill doing repetitive, mundane, and downright boring tasks that take away time from delivering valuable software and seeing your family.

Simply put, this book tells you how to put this thing called a computer to work doing some of that mundane (but important) project stuff. That means you'll have more time and energy to do the really exciting—and challenging—stuff, such as writing quality code. In other words, we'll task computers to do what they're good at, leaving us to do what we do well.

But aside from the obvious efficiency gains, automation also makes our project's procedures consistent and repeatable so that we spend less time debugging problems. How does this play out in real life? Let's start with a story....

1.1 Look Ma, No Hands!

Today we find Fred, our favorite programmer, working on his company's flagship product, the document management system, or *DMS* for short. OK, so "document management system" might be what Fred calls it on his resumé. It's really just a collection of HTML files that can be indexed and then searched. Fred chuckles as he thinks of how much venture capital (VC) money his company could have raised in 1998 just for promoting something by that name.

But it's 2004, and a cool product name and a web site just don't cut it. These days you actually have to demonstrate working software to loosen the VC purse strings. Speaking of which, Fred is in charge of preparing a demo for the venture capitalists tomorrow at noon. There's just one problem: By that time tomorrow Fred will be a few state lines away from the office. In fact, his RV is out in the parking lot right now, gassed up for a trip to the yearly family reunion in Kansas. Just as soon as he adds this last feature, Fred and his family will hit the road.

It Works on My Machine

Fred can already taste the barbecue sauce as he finishes up the last bit of code. He presses the Compile button on his favorite IDE. No errors. Then he runs all his local unit tests, and they pass. So far, so good. Now for the grand finale. Fred checks out the latest version of the rest of the project from the version control system to set up for an integration test. Then he touches off a build by running the project's build script.

WooHoo! The build succeeded. Fred is reminded once again that he's the world's greatest programmer. So he commits his changes, grabs his lunch pail, and races for the elevator. In the morning, all his team needs to do to deploy the demo is run the deployment script. They may even have time for a game of foosball before the venture capitalists show up at noon. Life is good as Fred, the missus, and all the rugrats crawl into the Winnebago and drive out of town.

Somewhere Out on I-70...

Fred has the pedal to the metal as the RV lumbers down I-70 in the dead of night. Just as the kids have dozed off, Fred is startled back into reality by a beep of his cell phone. It's a text message sent from the scheduled build process on the build machine back at the office, hundreds of miles in Fred's rearview mirror. When it woke up and tried to run a build, it failed. Fred grimaces as he reads the error message. In his haste he forgot to check in a new source file.

Fred leaves a voice mail for his faithful teammate Barney, letting him know that he'll need to check in the file before the demo. And then Fred goes back to counting mile markers.

The Next Morning

Barney strolls into the office a tad late the next morning. The whole team had worked hard preparing for the demo all week, so last night they celebrated by downing some brews at the bowling lanes. Checking voice mail is the last thing on what's left of Barney's mind. He'll return phone calls after the demo.

But he can't help but notice the boiling red bubbles in one of the Lava Lamps that the team uses to indicate the build status.[1] Oh no! The scheduled build has failed. When they left work last night, the green lamp was bubbling. "What could have happened?" Barney wonders as he checks the build status web page. It tells him that since the last successful build, one person has checked in code...Fred! The error message says he forgot to check in a file.

Back on Solid Ground

Perhaps it's time for Barney to check voice mail. He listens as Fred sheepishly explains that a local file on his machine needs to be checked in for the build to work. Having checked in the missing file, Barney wants some confidence that everything is in place for the demo. So he forces an independent build on the build machine. He also cranks up the frequency of scheduled builds so that Fred can't get so far away next time before finding out the build failed.

Everything compiles, and the tests pass on the build machine. Barney then runs a script that automatically creates a release branch containing the current versions of all files in version control, builds and tests the release branch, creates a distribution file, and deploys it into the demo web server.

After running the deployment script, Barney clicks through a few pages of the demo to make sure it looks right. Then he takes an early lunch before folks show up for the demo.

[1] Don't worry, you'll learn how to light up your own Lava Lamps in Section 6.2, *Getting Feedback from Visual Devices*, on page 128.

Then, Right Before the Demo...

Barney's pager goes off just as he's finishing his brontosaurus burger. The demo site has crashed. How does he know this? Well, Barney has been burned by demos crashing before. And when he has an itch, he finds some way to scratch it.

Before going to lunch, Barney hooked up a simple monitor to the demo web page. It automatically inspects the site every couple of minutes looking for an error message. If it finds one, it notifies Barney by sending him a text page. Fred gets the same text message on his cell phone, but he's up to his elbows in barbecued spareribs.

This time it looks like somebody shut down the database on the demo machine. Thankfully, there's time to straighten that out before the big demo.

A Happy Ending

Today we find Fred, Wilma, Barney, and the whole crew down at the bowling lanes high-fiving over the huge success of last week's demo. They all laugh at themselves for being in the stone age of automation for so long. "1998 called," Fred jokes. "It wants all its manual, repetitive, boring work back."

Sure, Fred learned his lesson about missing files—but more important, he and his team learned to appreciate all the automation that's watching their backs. It was automation that reduced the risk of a failed demo by notifying them early when problems popped up, wherever they were. It was automation (and version control) that saved them time by giving them a consistent and repeatable way to build and deploy their code. They'll prepare for a lot more demos and (if things go well) production releases after this. Automation will pay for itself many times over. That's what this book is all about.

1.2 Types of Automation

In a short amount of time, Fred and his team experienced the three primary types of automation shown in Figure 1.1 on the next page. Let's look at each of those in detail.

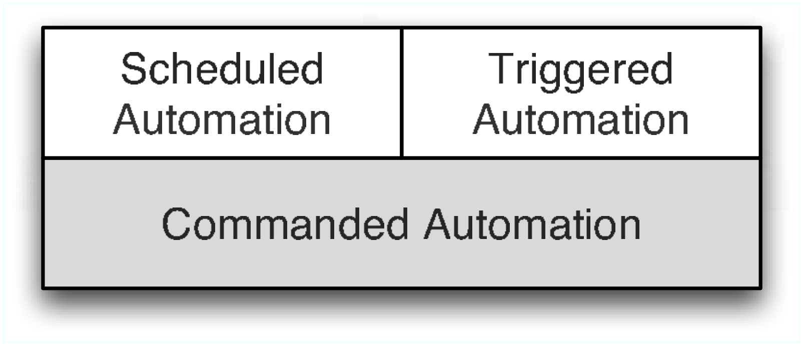

Figure 1.1: TYPES OF AUTOMATION

- *Commanded automation.* This happens anytime you run a command and the computer performs a set of tasks in a consistent and repeatable manner. For example, Fred ran a build script, and it attempted to generate a build just as it would on any machine. The computer remembered exactly how to do all the build steps for Fred, and everyone else on the project. Likewise, Barney ran a script that carried out the lock-step instructions for deploying the application consistently.
- *Scheduled automation.* Once you can get automation by running a command, then you can put that command on a schedule so that nobody *has* to run it manually. Fred forgot to check in a file, but even though he was miles away the scheduled build ran on time and notified him of the problem.
- *Triggered automation.* Commands can also be automatically run when some important event happens. For example, every time a file is checked in to version control a formatting script could be automatically run. Triggered automation is frequently associated with a scheduled task. For example, Barney wanted to reduce the risk of the demo site not being ready, but he didn't have time to continuously check the site. So he ran a monitor that periodically watched the site for an error event that then triggered his pager.

Because the team made effective use of all three types of automation while preparing for the demo, they got feedback at each of the stages: building, deploying, and monitoring their software. Imagine how stressful it might have been for the team otherwise.

1.3 Questions About Automation

Before diving into automation, it's only natural to have questions. Let's look at some common ones.

What Do I Need to Get Started?

The automation techniques used on Fred's project were fairly simple and inexpensive, but they didn't come for free. The team needed a few basic things in place before they could capitalize on automation.

- *Version control*. A central repository for all the files in their project gave the team a place to synchronize all their work. This in turn gave the build machine a single source from which the project could be built. Using version control also allowed Barney to create a snapshot of all the files used to build the demo so that the same demo can be reproduced at any time in the future. Version control is covered in detail in [TH03].
- *Automated tests*. Running automated tests—tests that check their own results—gave the team confidence in their code base. Fred ran automated tests on his local machine before checking in code to version control. The tests also ran as part of the scheduled build on the build machine to check that all the project code worked in harmony. Barney then ran the same automated tests to verify that the code in the release branch was ready for distribution. At each step in the project life cycle, from writing code to deploying a new release, the automated tests were run to gain confidence before moving on. Indeed, automated tests are the underpinning of effective project automation. Writing good automated tests is covered in detail in [HT03].

- *Scripting.* The team needed to write a few shell scripts (or batch files) to train the computer how to automate procedures. And while you can use programming languages such as Java for automation, a simple shell script is quicker to write, simpler to debug, and doesn't require a build process. Throughout this book we'll look at several scripting examples that make it easy for beginners to follow along.

- *Communication devices.* Automation helped the team communicate and get feedback even while they were on the go. Email and web pages are standard communication tools on software projects, but all too often they get ignored. It was a Lava Lamp that captured Barney's attention. Cell phones and text pagers let you get notifications on the road (or at the beach). Thankfully, we're surrounded by such communication devices these days, and in this book we'll put them to good use.

Why Should I Automate Something?

Frankly, you've got better things to do than piece together builds, follow checklists full of release commands, copy files around on servers, and monitor running programs. So automation will give you back something you don't have enough of: time. And with the global competition for development work heating up, you have to be as productive as possible.

Better yet, automation will give you confidence because automated procedures are accurate, consistent, and repeatable. People just aren't as good at repetitive tasks as machines. You run the risk of doing it differently the one time it matters, doing it on one machine but not another, or doing it just plain wrong. But the computer can do these tasks for you the same way, time after time, without bothering you. You don't have to fear something bad happening when you hit the Enter button.

Automation also reduces the need for documentation. Rather than explaining to a new team member all the steps that go into making a build or generating a release, you just show her how to run a script. And if she's interested, the script has all the details.

Automation changes the way you work. Not only does it make your job easier, it also enables you to perform critical project procedures as often as you should.

When Do I Automate Something?

The simple answer is that you should apply automation whenever you've grown tired of doing something manually. Some folks have higher boredom thresholds than others. As a rule of thumb, manual procedures that will be run more than twice should be automated. Odds are the third time won't be the last.

Errors follow naturally from boredom, so if a repeated manual procedure needs to be accurate and consistent, then it's time for automation.

But remember, this book is about being pragmatic. Never spend more time developing an automated solution than the time the solution will ultimately save.

When Should Automation Run?

The frequency of automation varies with the procedure being automated. For example, the build process is commanded automation that runs whenever we want to create a build. Scheduled builds, on the other hand, should run as often as necessary to give us timely feedback about the health of our software. The scheduled build we'll set up will run many times a day.

Releasing and deploying applications will occur on a less frequent basis, in phase with the project's release cycle. When we have enough new features or bug fixes, we run a command to generate a release and possibly another command to deploy new software to a server.

Monitoring can happen in real time such as when an event is triggered or in a polling loop with a configurable interval.

In the road map that follows, each procedure we automate includes a suggestion of its frequency.

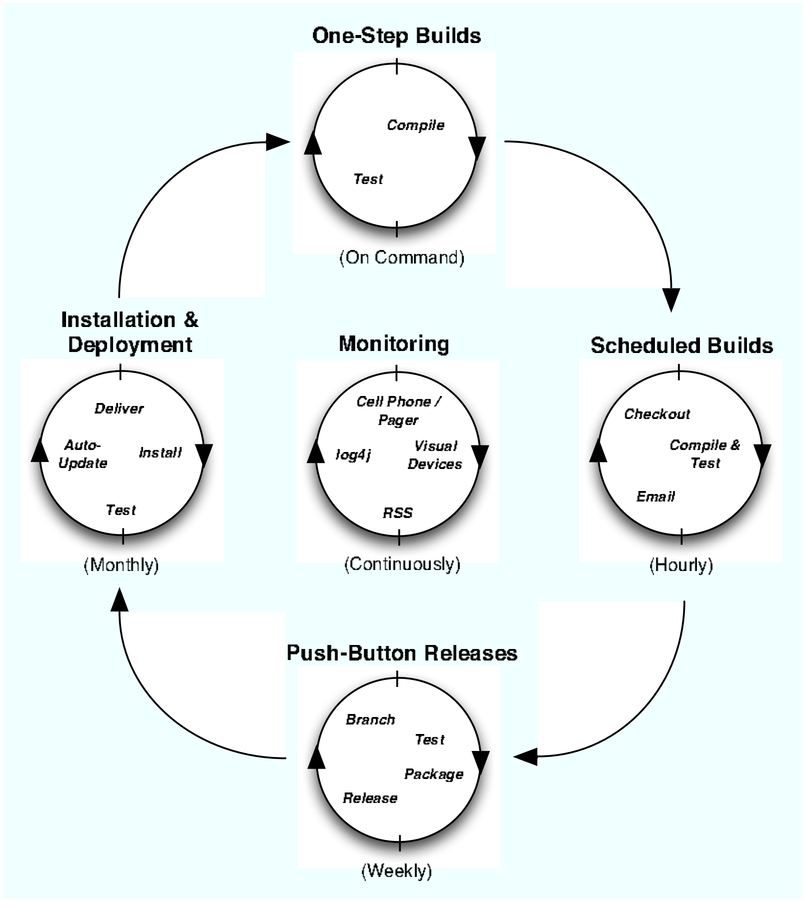

Figure 1.2: AUTOMATION ROAD MAP

1.4 Road Map

Figure 1.2 shows the procedures we'll visit. We'll start with one-step builds that can be run by everyone on your team. Then we'll put the build on a schedule so we always have fresh software. When it's ready to be released, we'll push a button to cut a new distribution. Finally, we'll make that distribution available to our customers through an automated installation process. Throughout this cycle we'll set up monitors that alert us to problems that require our attention.