Extracted from:

Manage It!

Your Guide to Modern, Pragmatic Project Management

Manage It!

Your Guide to Modern, Pragmatic Project Management

Johanna Rothman

The Pragmatic Bookshelf
Raleigh, North Carolina Dallas, Texas

Chapter 4

Scheduling the Project

Planning and scheduling are two separate activities. In Chapter 2, *Planning the Project*, on page 35, you started the project planning. Here, you'll think about scheduling and estimating the project. As you organize the schedule—and when you reestimate the work—you might have to modify the plan. That's fine. Your original plan is just good enough to start. Expect to refine the plan as you schedule (and reschedule). And, don't be afraid to refine the schedule as you replan.

4.1 Pragmatic Approaches to Project Scheduling

You planned just enough to start the project already. You need to schedule enough to start the project. There's no point to scheduling the whole darn project when you know the project is going to evolve. If you're working with a customer who wants to see a project schedule before they will sign a contract, be clear that the initial schedule is your best first guess. It will change. And, have that customer read about accuracy and precision of schedules in the tip *Estimates Need Accuracy, Not Precision*, on page 88.

A few years ago, I had a conversation with a project manager at a conference. I said it took me anywhere from about half a day to a couple of days to get started on a project schedule, and I wanted to shorten the two days down to half a day.

The other project manager stood there with her mouth open. She absolutely didn't believe that I could schedule a project in half a day. I explained that I didn't try to schedule everything, just the next week or so, and then I would build up the major milestones and the rolling-wave schedule (see Appendix B, on page 345) over the next few weeks.

"How do you know the end date?" she asked.

"I don't, at least not precisely. But if I tried to plan forward to see where the end date would be that early in the project without any data, I'd be wrong. Why take the time to schedule in detail when you know you'll be wrong?"

She said, "Gee, I never thought of it that way."

There are many ways to schedule a project. I think top-down, so I create a first draft of the plan because that helps me create a first-draft schedule. Other project managers start with a schedule draft first. Do what feels most comfortable to you and appropriate for the project. But don't neglect either the plan or the schedule. Every project needs both.

Tip: Projects Require Both Plans and Schedules

As the pragmatic project manager, your job is to start the project with just enough planning and to continue to plan (and replan) as you proceed. Whatever you do, don't ignore either the plan, especially the release criteria, or the schedule. You might not need a fancy schmancy Gantt chart for a schedule; yellow stickies on the wall are fine for many projects. But you do need to both plan and schedule.

Your schedule will bear some resemblance to the life cycle you choose. But remember, a life cycle is a *model* of how the project could look. When it's time to create the schedule, use the life cycle as a guideline, and make sure you've addressed the risks inherent in your life cycle, no matter how you need to do that. You might add timeboxed iterations and increments to phase-gate project schedules—as well as planning to replan—because those actions made sense for the particular project. Remember that a life cycle is a guide, not a straightjacket.

Scheduling and estimating are two different activities. Scheduling is ordering and showing the interdependencies of tasks. Estimating is guessing how many effort-hours a particular task will take. They are linked, because how you organize the schedule might depend on a given task's estimate of the effort-hours and specific people required.

I wish I could wave my magic wand and say, "Here is the One Right Way to schedule and estimate your project." But I can't. We generally need to estimate things we've never done before.

Estimation of the unknown is still an art. On the other hand, if you know what life cycle you're using, organizing the schedule is easier.

Tip: Timebox Initial Planning

Spend as little time as possible on up-front planning, especially if your project team is already assigned to your project. Take just enough time to plan so your project team can start. Timebox the charter to one hour. Timebox the project plan to another hour. Timebox the first draft of the schedule to an hour. The timeboxing will focus everyone on the few vital pieces they need to start. Once people know what they have to work on for the next week or two, you can return to the plans and schedule and see what else you need to write.

4.2 Select from These Scheduling Techniques

I select from among these scheduling techniques when laying out the project: top-down, bottom-up, and inside out, Hudson Bay Start, and a short iteration.

Top-Down Scheduling

Top-down scheduling generally starts with milestones. Serial life cycles tend to start with top-down scheduling, because the phases are so clear. (Hint: if you must use a serial life cycle, make sure you use deliverable-based planning as a technique to generate your milestones, as discussed in Section 4.3, *Deliverable-Based Planning*, on page 77.)

Organize the project schedule into phases, iterations, or chunks. Lay them out on a whiteboard or on stickies on a wall. Dwayne Phillips recommends cards on a wall as another low-tech scheduling technique. When you schedule with cards on the wall, each person creates cards with the tasks they think they need to do. Then link the cards with string [Phi04]. This technique is particularly helpful if you don't know where to start.

The team starts organizing the schedule from the highest-level milestones and develops the tasks to support those milestones. As one or more team members understand more about what each milestone means, they break the milestone down into its component tasks.

The smaller the task at the bottom level, the easier it is to estimate how long the task will take.

Bottom-up Scheduling

Bottom-up scheduling starts with specific tasks. If you're using an incremental life cycle, it might make sense to start with bottom-up scheduling. "We know we need to do this feature first, then do those features, and then have a go/no-go decision. . . ."

The project team members, working alone or in cross-functional teams, develop the milestones from the tasks. As the project manager, you can ask questions about how things fit together. (The more technical you are, the more you can help. If you don't have domain expertise in the product, don't interfere.)

Inside-out Scheduling

Inside-out scheduling works best with people who think they need to be completely adaptable. At one of my project management workshops, one PM said, "First I make a mind-map [BB96] of everything I know about the project. I might know some go/no-go review points. I might know about certain features. But I don't know about everything at the same level, so I want to see everything before I start scheduling."

Your mind-map might be crystal clear to you. But it might not be clear to others on the project. Mind-maps communicate much more to those present when it was created than to those who are just shown the results later. If you and your project team are using inside-out scheduling, make sure the team works together to generate the tasks and milestones.

Hudson Bay Start

Imagine you're managing a project that's completely new to you and the entire project team. You have no idea whether the environment you have will support the tools. You don't know how to estimate the project. Consider a short iteration, such as a *Hudson Bay Start*.

The Hudson Bay Start approach was originated by the Hudson Bay Company in the 1600–1700s in northeastern Canada. The Hudson Bay Company outfitted fur traders. To make sure the traders hadn't forgotten anything they needed, they left Hudson Bay and camped just a few miles away. By making camp just a few miles away, the traders ensured

they hadn't forgotten any tools or supplies—before they abandoned civilization. With just a short start to their journey, they had a better idea about their ability to endure the winter.

A Hudson Bay Start is a technique that allows the project team to push something through the project's environment. You want this to be as small a thing as possible. (A "Hello World" program might be just fine.) The idea is for the project team to see what it would be like to start working in this environment with this product domain.

If you and the team can't figure out what it would take to estimate any piece, timebox a Hudson Bay Start. Start something you can complete in four hours or less. (This thing doesn't have to be real functionality.) After the team has created *something*, debrief the activity. The team will know more about how to estimate the tasks needed. If the team knows only a little more, start with a short iteration, and then decide what to do.

A Hudson Bay Start helps in several ways. First, the team gains some confidence that they can accomplish *something*. Finishing something helps them gain some insight when it comes to estimating. In addition, the team has a little insight into how to organize some tasks. "Oh, if we want to do those features in parallel, we're going to have to make another branch and merge back in. Yikes, that means staging integration. That will take longer than working on the mainline."

When you hear conversations like this, where people articulate the risks, then you can capture them in a *parking lot* (see Appendix B, on page 345) to deal with later or as you schedule.

Start with a Short Iteration

Use a short timeboxed iteration when the team understands the environment but isn't sure how to estimate the tasks. A short iteration helps people see how much they can accomplish in one or two weeks, so their follow-on estimates are more accurate. You can use a short iteration after a Hudson Bay Start, once the team understands how to use the environment.

Timebox a short iteration (no more than two weeks—one week is even better), and see what the team can accomplish in that time. By the end of the iteration, the team and you will have a better idea about the requirements, the risks, and what they don't know.

If you combine a short iteration with a short retrospective, the entire team will learn more about what it takes to schedule this project.

4.3 Start Scheduling with a Low-Tech Tool

Back in the Stone Age, when I started managing projects, we didn't have electronic scheduling tools. We had blackboards, paper, and flowchart templates. I used a blackboard to lay out the schedule for projects. Blackboards worked well—if I made a mistake, I erased the sequence and inserted it where I needed it.

But blackboards can become messy if you have to erase and rewrite information. Even when I moved to whiteboards in the Neolithic Age, the whiteboard can be hard to see—sometimes the old information is still visible under the new drawings.

When yellow stickies came out in the Modern Age, I moved to yellow stickies.[1] It's easy to write a task on a sticky, put the sticky up on the wall, and discuss with the rest of the project team—sometimes quite loudly—the sequence of tasks or who will do them or what the risks are. And, if the task is in the wrong place—because the team sees another way to organize the project—it's easy to move the sticky from one place to another.

Yellow stickies involve the whole team in scheduling. The team will explain the risk as they proceed, providing you with valuable information you can use for steering the project.

High-Tech vs. Low-Tech Scheduling
by Sandy, seasoned project manager

I've been managing projects for about fifteen years. I started when we had scheduling tools, and I became an expert at the best-known tool. Sure, it had problems originally rolling up subprojects, but I knew how to get around that. And, we had a little problem with trying to track the details, but I got good at figuring out how to outwit the tool. I had a little problem with earned value calculations, but we moved to implementing by feature, and that helped (see Section 11.2, *Earned Value for Software Projects Makes Little Sense*, on page 220).

1. For those of you who are wondering why I didn't move to an electronic scheduling tool, the answer is easy—one didn't exist for the operating systems I was using. Since it didn't exist, I couldn't use it.

> Then I started managing a really large program a couple of years ago, including about 300 people in six sites. I'm no dummy, so I brought all the project managers for an initial planning meeting. I had my computer hooked up to the projector, and we started developing the schedule. Everyone was yelling at me, trying to make me see where tasks belonged. I was a little stressed but was getting there. Then the power died.
>
> Bob, one of the subproject managers, said, "Don't go anywhere. I'll be right back, and we can continue." He came back in about five minutes with pads of yellow stickies and pens. He explained how we would schedule and then everyone started writing their stickies. In about ten minutes, we started posting the stickies on the wall and discussed what each one meant and where we had issues.
>
> We had an initial schedule in less than an hour. We took pictures of it, in case the power stayed off and my computer ran out of juice.
>
> At the end of the meeting, every subproject manager congratulated me on how quickly we developed a schedule. Me! I gave all the credit to Bob. That schedule was good for a couple of months, and when we had to update it, we gathered the subproject managers together and did the same thing.
>
> I was amazed by how well it worked. I still use scheduling tools, but I always start with low-tech scheduling, and if we need a major replan, I use low-tech scheduling now.

Many project managers prefer to start scheduling with an electronic scheduling tool. If you need to lay out many tasks at once and you think the sequence of those tasks are not going to change, maybe an electronic scheduling tool works at the beginning of the project. But it doesn't involve the entire team in the scheduling activity. Using a tool to generate a schedule shortcuts the discussion and doesn't expose silent dependencies and risks.

The project manager can type only one task at a time—and only the project manager can create tasks. The scheduling tool can show you only one page of information at a time, and the team might lose context if they can't see the whole schedule.

If you're not using rolling-wave planning, then an electronic scheduling tool might be OK once you and the team create the initial project schedule. (You will lose the benefit of a Big Visible Chart or Information Radiator; see Chapter 11, *Creating and Using a Project Dashboard*, on page 214.) But starting with a tool says to the team, "I'm in charge of the schedule; you're not."

If the project team owns the schedule, they will stay committed to it. If you own the schedule, you're likely to micromanage the team, not manage the interdependencies of their tasks.

I hope I've convinced you to start with stickies or cards on the wall. If you're not sure how to do that, here are several techniques I've used for different projects.

Basic Sticky Scheduling

Gather the entire project team together in a room with a long wall or a long whiteboard. Hand everyone a pad of yellow stickies and a medium or bold black pen. (I prefer to use three-inch by five-inch stickies so they're big enough to read and a felt-tip black pen.) If you know you're using a serial, iterative, or incremental life cycle, post the major milestones on the wall so people can see the structure of the project. Ask everyone to write all their tasks down on a sticky, one task per sticky. As the team members write down tasks, they post them on the wall. (You can see examples of this in Figure 4.1, on the next page, as well as in Figure 4.2, on page 75.)

Assign one part of the wall as the parking lot (Appendix B, on page 345), the place where the team will collect questions and assumptions that you'll need to resolve as part of the scheduling. I use flip chart paper for the parking lot, so if I need to take the parking lot back to my office to resolve, it's easily transportable.

Now stand back, out of the way. The project team members will start collaborating about the sequence of events, any prerequisites, assumptions, and questions.

As developers start writing their tasks, they will have questions for requirements analysts, writers, and testers—who will have questions for the developers. The project team starts to bond in a cross-functional way before the project "starts." (In reality, the project has already started—see the sidebar on page 32—there just are no other artifacts at this time.) You can see what a short project might look like in Figure 4.1, on the next page.

Once the team has written down as much as they can and resolved the issues, it's time for you to be involved. Expect to see these issues in the schedule:

- The team has scheduled only the first few weeks of their work. They can't see much more detail than a few weeks out, so that's

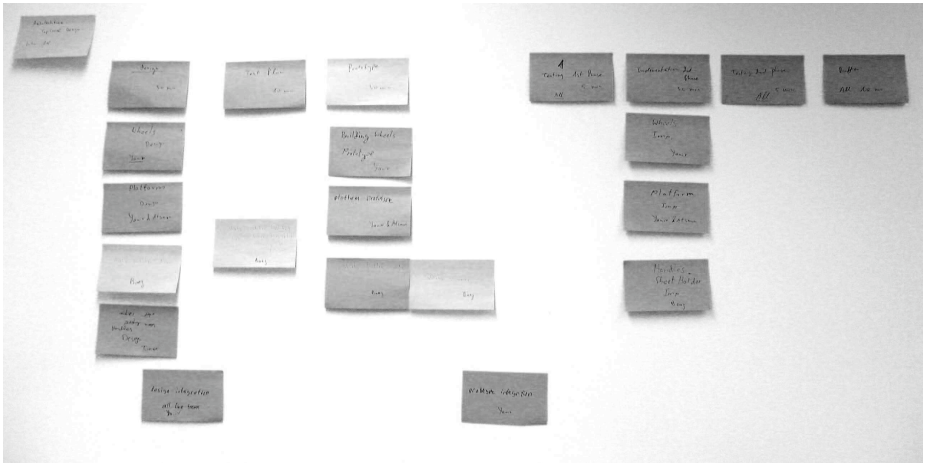


Figure 4.1: One project's yellow sticky schedule

all they've scheduled. That's OK, because you can use rolling-wave planning (Section 5.6, *Using Rolling-Wave Scheduling*, on page 97) to iterate on the schedule. And, it's OK because you don't want people to provide detail that isn't based in reality. More detail is a waste of time.

- You might see long sequences of serial tasks. Expect this in a serial life cycle. But if you're seeing this in an iterative or incremental life cycle, ask the team whether something is preventing them from working more in parallel. See Figure 4.2, on the next page, to see exactly the same project as Figure 4.1—except organized in a more serial way.
- You might see long sequences of many parallel tasks. You have to worry about this only in a serial life cycle, which does not—by its nature—lend itself to parallelism. However, it's a risk to the project in any life cycle other than agile. The risk is that people will fall out of sync and extend the critical path where you did not expect the critical path to be.

Once the team has created the schedule, the team is ready to estimate how long each task will take.

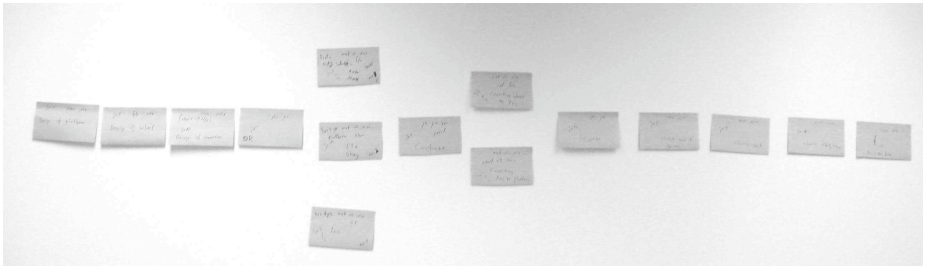


Figure 4.2: Another project's yellow sticky schedule

Sticky Scheduling with Arrows

One of my clients starts with yellow-sticky scheduling as described here, but once the schedule is "set," they draw arrows from one sticky to another. The arrows help them in several ways. The first is that if a sticky falls down, they know where to put it back. The second is that after they do the initial yellow-sticky scheduling, they transition to an electronic scheduling tool. A project coordinator transcribes each sticky into a task into the tool, and the arrows help them keep track of dependencies.

Sticky Scheduling for Each Group

If you're stuck with a phase-gate schedule and can't create a cross-functional team to implement by feature, you might need the help of a schedule to convince your management that there are other options. I've used sticky scheduling for a week-by-week look at the schedule to help management understand that organizing by functional team slows the project down.

On a large whiteboard or on paper taped to the wall, draw vertical lines down, one for each week. Use different colored stickies to show when different people in different functional organizations are working on the project.

Don't forget to show the end of the project. The end of functionally organized projects tends to be difficult. Because management thinks the developers are free to start another project, they are less dedicated to the project at the time the project needs them most—when it's time to fix defects.

Sticky Scheduling for Features

Recently, I've started using sticky scheduling to show how each feature will integrate with the others. If you are working on complex projects where you have dependencies during the project for integration, you might find it useful to plan an iteration's worth of work with stickies.

Generate a sticky for each deliverable. Sometimes, a single feature will have several interim deliverables. Put the stickies up on the wall. Ask the project team to organize when they need which deliverable delivered into the code base. Ask the team to add any hard dates; "If you don't deliver that piece then, we can't finish before the end of the iteration."

Especially if you're working in short iterations, you don't need to transcribe the stickies into a Gantt chart. If you're working in an incremental life cycle, you might need to tape the stickies up for a longer project or use a Gantt to manage the dependencies.

Benefits of Using Sticky Scheduling

If you use sticky scheduling, you will not have a beautiful Gantt chart that can show you the critical path. That's good, because the critical path for a software project runs through the tasks, the people, and sometimes the equipment. And, I bet your critical path changes day by day, depending on what people finished. Even if your critical path doesn't change daily, it changes weekly. If you don't have a line on the Gantt chart that purports to show you the critical path, you and the team will have to think about it. Thinking about it more consciously will help everyone to manage it.

In addition, a yellow-sticky schedule will not show you the end date. That's because you should never estimate a single-point end date [DL03]. But since a scheduling tool does calculate the end date (and it's the earliest possible end date you can't prove the project won't be complete by), people—especially senior management—believes that end date.

If you're running a multisite project, you can still use sticky scheduling. If each team is responsible for a complete deliverable (a set of tested implemented features; see Section 12.3, *Make Sure Each Site Has Complete Deliverables to the Project*, on page 251), each team does its own day-to-day scheduling. You gather the team leads or project managers together to make sure they understand who is delivering what to whom and when. Since you're dealing with major milestones, you can use

videoconferencing or webconferencing to use the equivalent of sticky scheduling.

Deliverable-Based Planning

Yellow-sticky planning lends itself well to deliverable-based planning. As people think about what they have to deliver to the rest of the project, they develop milestones based on deliverables, not on the ending of phases.

Phase-based planning or functional-based planning assumes that teams of people from a particular function are responsible for a piece of the project. And you can assume a phase of the project is done when those people say they are done. If you've ever worked on a project that had a milestone such "requirements freeze" or "code freeze," you've worked on a phase-planned project.

The problem is that although those people try hard to complete their deliverables, the freezes are rarely frozen, and the completes are mostly incomplete. You end up with slushy milestones. The way to avoid slushy milestones is to plan for the milestone as a rollup of the tasks before it. If you know you have several areas of requirements, the milestone "requirements freeze" is a rollup of "requirement 1 written and reviewed," "requirement 2 written and reviewed," "requirement 3 written and reviewed," and so on, until all the requirements are in the rollup.

You can use deliverable-based planning in any life cycle. Especially if you must use a serial life cycle, use deliverable-based planning to obtain feedback early about the project's progress. If you can't meet requirements freeze, how can you know you'll meet any of the later milestones?

Tip: Late Projects Don't Make Up Time; They Get Later

> If you realize at the beginning of the project that the team is not making the progress you want to see, decide what to do differently. Late projects never make up time. They get later and later and later. . . .
>
> If you do think the project will make up time, you will find yourself in the schedule game discussed in Section 6.15, *We'll Go Faster Now*, on page 133.

Remember This

- Start scheduling with low-tech tools. If you really need a scheduling tool, transfer the data later. Be aware of the costs associated with losing the Big Visible Chart or Information Radiator.
- Schedule by deliverables, not by functions.
- Plan to iterate the schedule. A write-once schedule is not worth the time you spent generating it.

A Pragmatic Career

Welcome to the Pragmatic Community. We hope you've enjoyed this title.

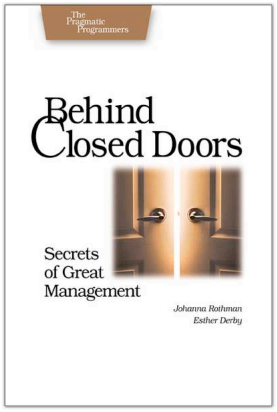
If you've enjoyed this book by Johanna Rothman, and want to advance your management career, you'll be interested in seeing what happens *Behind Closed Doors*. And see how you can lead you team to success by using *Agile Retrospectives*.

Behind Closed Doors

You can learn to be a better manager—even a great manager—with this guide. You'll find powerful tips covering:

• Delegating effectively • Using feedback and goal-setting • Developing influence • Handling one-on-one meetings • Coaching and mentoring • Deciding what work to do-and what not to do • . . . and more!

Behind Closed Doors Secrets of Great Management
Johanna Rothman and Esther Derby
(192 pages) ISBN: 0-9766940-2-6. $24.95
http://pragmaticprogrammer.com/titles/rdbcd

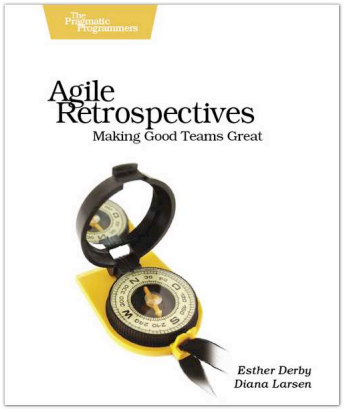


Agile Retrospectives

Mine the experience of your software development team continually throughout the life of the project. Rather than waiting until the end of the project—as with a traditional retrospective, when it's too late to help—agile retrospectives help you adjust to change *today*.

The tools and recipes in this book will help you uncover and solve hidden (and not-so-hidden) problems with your technology, your methodology, and those difficult "people issues" on your team.

Agile Retrospectives: Making Good Teams Great
Esther Derby and Diana Larsen
(170 pages) ISBN: 0-9776166-4-9. $29.95
http://pragmaticprogrammer.com/titles/dlret

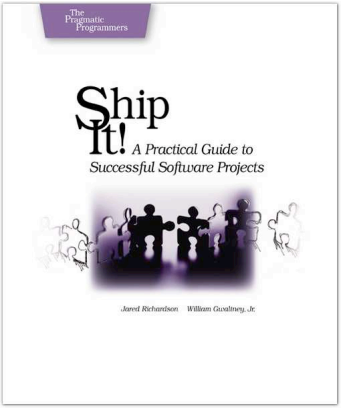


Competitive Edge

Need to get software out the door? Then you want to see how to *Ship It!* with less fuss and more features. And every developer can benefit from the *Practices of an Agile Developer*.

Ship It!

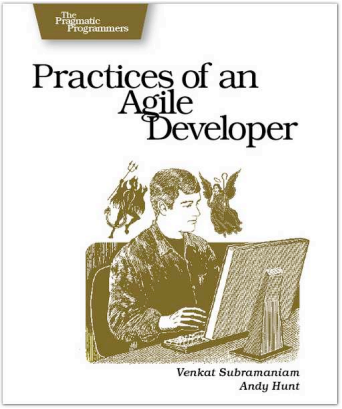
Page after page of solid advice, all tried and tested in the real world. This book offers a collection of tips that show you what tools a successful team has to use, and how to use them well. You'll get quick, easy-to-follow advice on modern techniques and when they should be applied. **You need this book if:** • You're frustrated at lack of progress on your project. • You want to make yourself and your team more valuable. • You've looked at methodologies such as Extreme Programming (XP) and felt they were too, well, extreme. • You've looked at the Rational Unified Process (RUP) or CMM/I methods and cringed at the learning curve and costs. • **You need to get software out the door without excuses**



Ship It! A Practical Guide to Successful Software Projects
Jared Richardson and Will Gwaltney
(200 pages) ISBN: 0-9745140-4-7. $29.95
http://pragmaticprogrammer.com/titles/prj

Practices of an Agile Developer

Agility is all about using feedback to respond to change. Learn how to apply the principles of agility throughout the software development process • Establish and maintain an agile working environment • Deliver what users really want • Use personal agile techniques for better coding and debugging • Use effective collaborative techniques for better teamwork • Move to an agile approach



Practices of an Agile Developer: Working in the Real World
Venkat Subramaniam and Andy Hunt
(189 pages) ISBN: 0-9745140-8-X. $29.95
http://pragmaticprogrammer.com/titles/pad

Cutting Edge

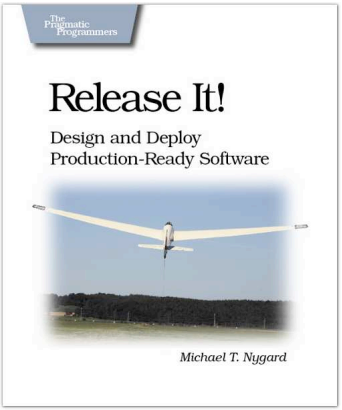
Now that you've finished your project, are you sure that it's ready for the real world? Are you truly ready to *Release It!* in this crazy world?

Interested in Ruby on Rails, but don't want to learn another framework from scratch? You don't have to! *Rails for Java Programmers*leverages you and your team's knowledge of Java to quickly learn the Rails environment.

Release It!

Whether it's in Java, .NET, or Ruby on Rails, getting your application ready to ship is only half the battle. Did you design your system to survive a sudden rush of visitors from Digg or Slashdot? Or an influx of real world customers from 100 different countries? Are you ready for a world filled with flakey networks, tangled databases, and impatient users?

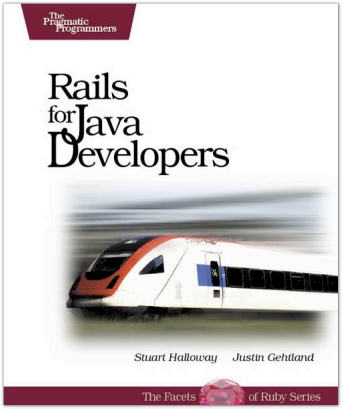
If you're a developer and don't want to be on call at 3AM for the rest of your life, this book will help.



Design and Deploy Production-Ready Software
Michael T. Nygard
(368 pages) ISBN: 0-9787392-1-3. $34.95
http://pragmaticprogrammer.com/titles/mnee

Rails for Java Developers

Enterprise Java developers already have most of the skills needed to create Rails applications. They just need a guide which shows how their Java knowledge maps to the Rails world. That's what this book does. It covers: • The Ruby language • Building MVC Applications • Unit and Functional Testing • Security • Project Automation • Configuration • Web Services This book is the fast track for Java programmers who are learning or evaluating Ruby on Rails.



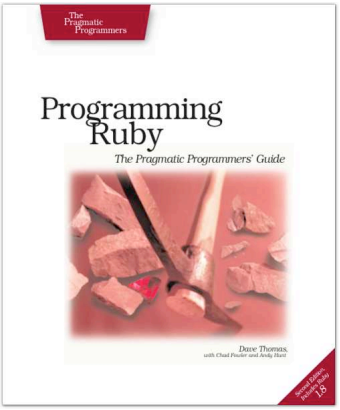
Rails for Java Developers
Stuart Halloway and Justin Gehtland
(300 pages) ISBN: 0-9776166-9-X. $34.95
http://pragmaticprogrammer.com/titles/fr_r4j

Facets of Ruby Series

If you're serious about Ruby, you need the definitive reference to the language. The Pickaxe: *Programming Ruby: The Pragmatic Programmer's Guide, Second Edition*. This is *the* definitive guide for all Ruby programmers. And you'll need a good text editor, too. On the Mac, we recommend TextMate.

Programming Ruby (The Pickaxe)

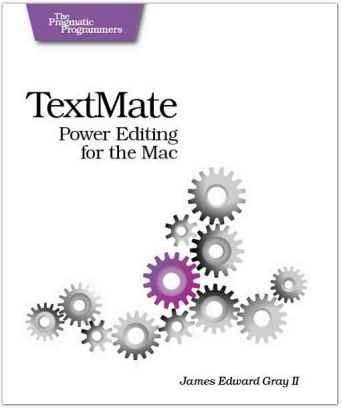
The Pickaxe book, named for the tool on the cover, is the definitive reference to this highly-regarded language. • Up-to-date and expanded for Ruby version 1.8 • Complete documentation of all the built-in classes, modules, and methods • Complete descriptions of all ninety-eight standard libraries • 200+ pages of new content in this edition • Learn more about Ruby's web tools, unit testing, and programming philosophy



Programming Ruby: The Pragmatic Programmer's Guide, 2nd Edition
Dave Thomas with Chad Fowler and Andy Hunt
(864 pages) ISBN: 0-9745140-5-5. $44.95
http://pragmaticprogrammer.com/titles/ruby

TextMate

If you're coding Ruby or Rails on a Mac, then you owe it to yourself to get the TextMate editor. And, once you're using TextMate, you owe it to yourself to pick up this book. It's packed with information which will help you automate all your editing tasks, saving you time to concentrate on the important stuff. Use snippets to insert boilerplate code and refactorings to move stuff around. Learn how to write your own extensions to customize it to the way you work.



TextMate: Power Editing for the Mac
James Edward Gray II
(200 pages) ISBN: 0-9787392-3-X. $29.95
http://pragmaticprogrammer.com/titles/textmate

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style, and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Manage It! Home Page
http://pragmaticprogrammer.com/titles/jrpm
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragmaticprogrammer.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragmaticprogrammer.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragmaticprogrammer.com/news
Check out the latest pragmatic developments in the news.

Buy the Book

If you liked this PDF, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragmaticprogrammer.com/titles/jrpm.

Contact Us

Phone Orders:	1-800-699-PROG (+1 919 847 3884)
Online Orders:	www.pragmaticprogrammer.com/catalog
Customer Service:	orders@pragmaticprogrammer.com
Non-English Versions:	translations@pragmaticprogrammer.com
Pragmatic Teaching:	academic@pragmaticprogrammer.com
Author Proposals:	proposals@pragmaticprogrammer.com