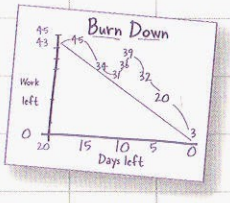
A Brain-Friendly Guide

Head First Software Development

Learn the real user story of how Mary satisfied her customers

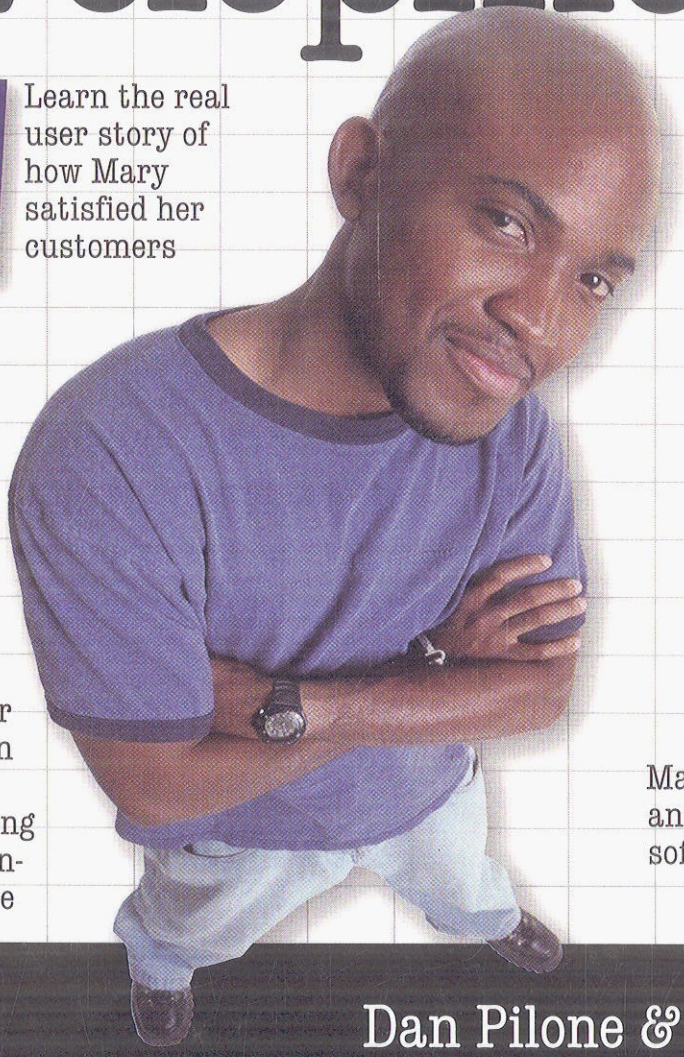
Score big by using velocity to figure out how fast your team can produce

Use test-driven development to avoid unsightly software disasters



Keep your project on schedule by tracking your burn-down rate

Master the techniques and tools of seasoned software developers



O'REILLY®

Dan Pilone & Russ Miles

Table of Contents (Summary)

Table of Contents (the real thing)

Intro

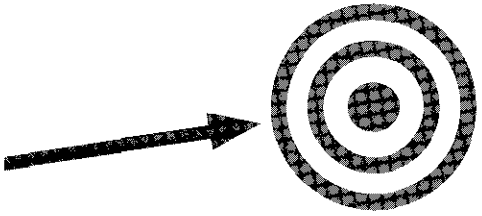
Your brain on Software Development. You're sitting around trying to *learn* something, but your *brain* keeps telling you all that learning *isn't important*. Your brain's saying, "Better leave room for more important things, like which wild animals to avoid and whether naked rock-climbing is a bad idea." So how *do* you trick your brain into thinking that your life really depends on learning how to develop great software?

great software development

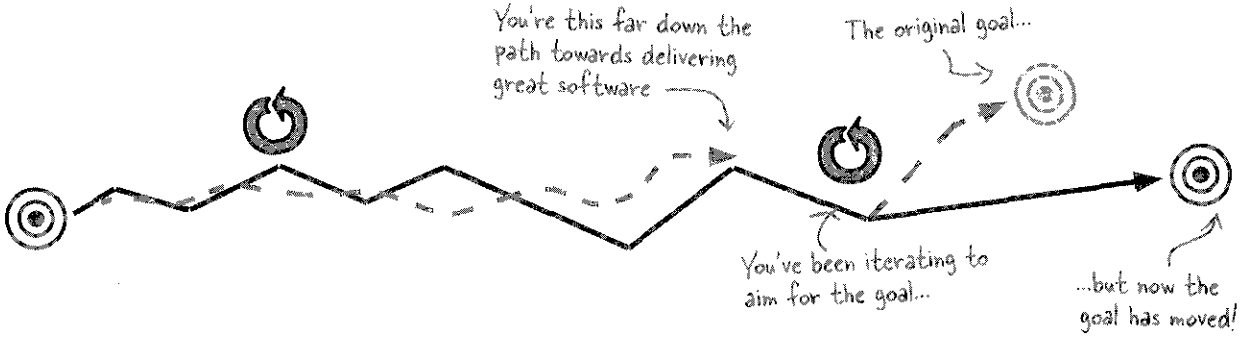
1 Pleasing your customer

If the customer's unhappy, *everyone's* unhappy!

Every great piece of software starts with a customer's big idea. It's your job as a professional software developer to **bring those ideas to life**. But taking a vague idea and turning it into working code—code that **satisfies your customer**—isn't so easy. In this chapter you'll learn how to avoid being a software development casualty by delivering software that is **needed**, **on-time**, and **on-budget**. Grab your laptop and let's set out on the road to shipping great software.



The Goal

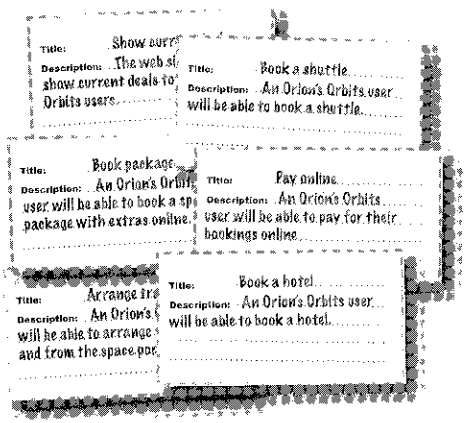


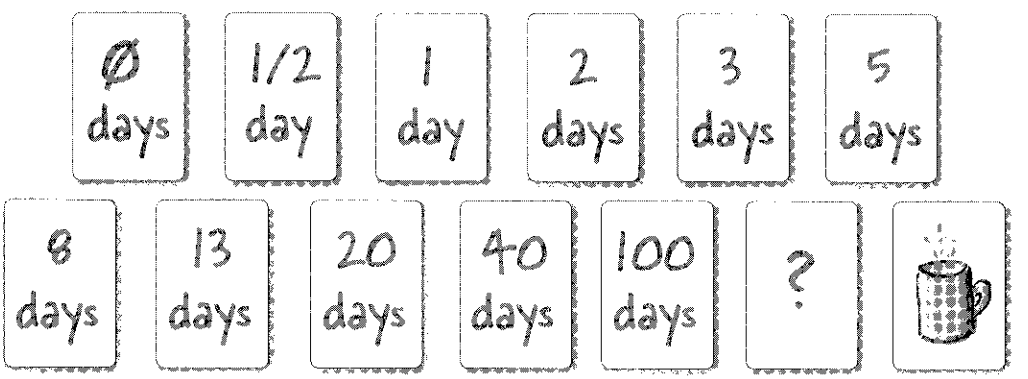
gathering requirements

2 Knowing what the customer wants

You can't always get what you want...but the customer better!

Great software development delivers **what the customer wants**. This chapter is all about **talking to the customer** to figure out what their **requirements** are for your software. You'll learn how **user stories**, **brainstorming**, and the **estimation game** help you get inside your customer's head. That way, by the time you finish your project, you'll be confident you've built what your customer wants... and not just a poor imitation.





project planning

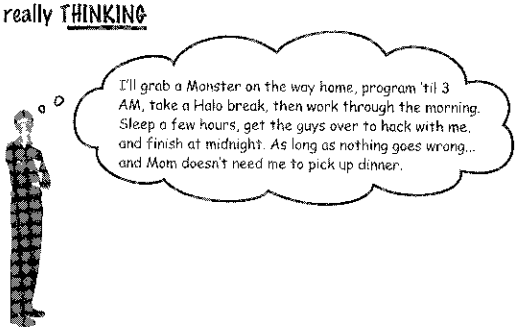
3 Planning for success

Every great piece of software starts with a great plan.

In this chapter you're going to learn how to create that plan. You're going to learn how to work with the customer to **prioritize their requirements**. You'll **define iterations** that you and your team can then work towards. Finally you'll create an achievable **development plan** that you and your team can confidently **execute** and **monitor.** By the time you're done, you'll know exactly how to get from requirements to milestone 1.0.

Here's what a programmer SAYS...

...but here's what he's really THINKING

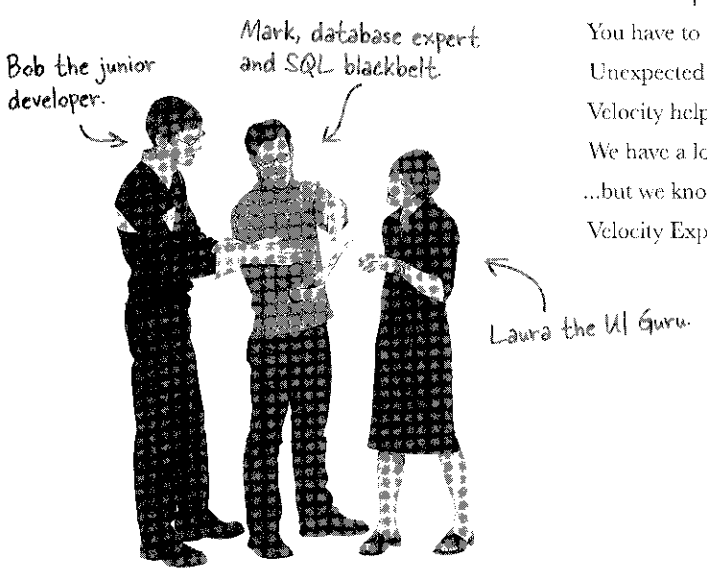


user stories and tasks

4 Getting to the real work

It's time to go to work. User stories captured what you need to develop, but now it's time to knuckle down and **dish out the work that needs to be done** so that you can bring those user stories to life. In this chapter you'll learn how to **break your user stories into tasks,** and how your **task estimates** help you track your project from inception to completion. You'll learn how to update your board, moving tasks from in-progress, to complete, to finally **completing an entire user story**. Along the way, you'll handle and prioritize the inevitable **unexpected work** your customer will add to your plate.

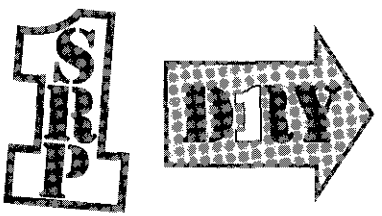
Your first standup meeting...

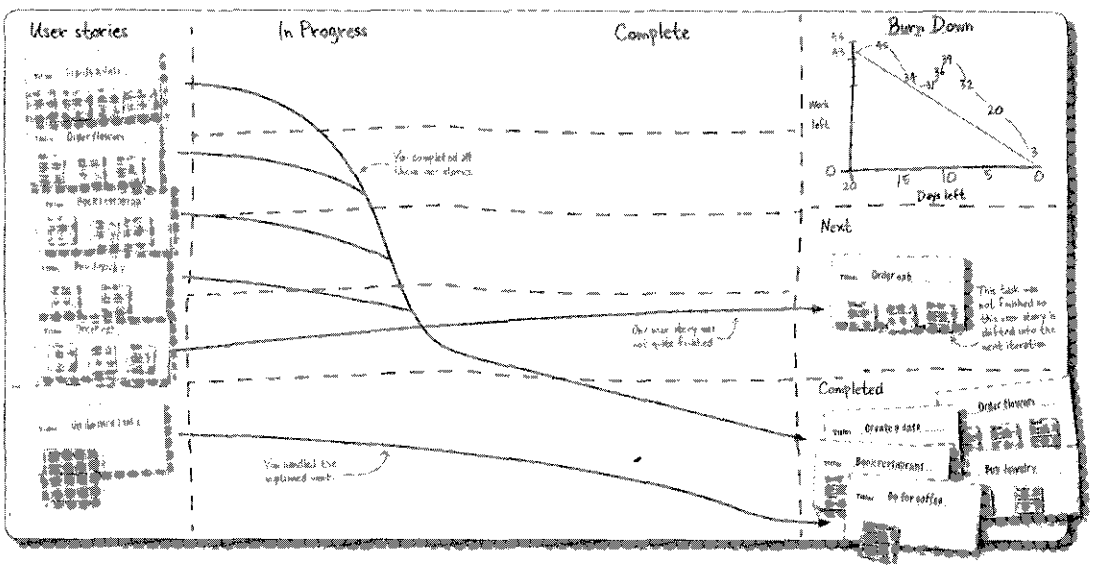


good-enough design

5 Getting it done with great design

Good design helps you deliver. In the last chapter things were looking pretty dire. A **bad design** was making life **hard for everyone** and, to make matters worse, an unplanned task cropped up. In this chapter you'll see how to **refactor** your design so that you and your team can be **more productive**. You'll apply **principles of good design**, while at the same time being wary of striving for the promise of the ***'perfect design'***. Finally you'll **handle unplanned tasks** in exactly the same way you handle all the other work on your project using the big project board on your wall.

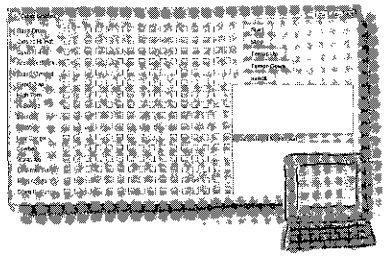




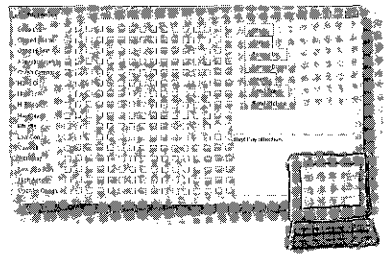
6 Defensive development

When it comes to writing great software, *Safety First!*

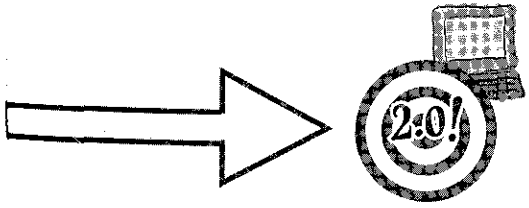
Writing great software isn't easy... especially when you've got to make sure your code works, and **make sure it keeps working**. All it takes is one typo, one bad decision from a co-worker, one crashed hard drive, and suddenly all your work goes down the drain. But with **version control**, you can make sure your **code is always safe** in a code repository, you can **undo mistakes**, and you can make **bug fixes**—to new and old versions of your software.



BeatBox Pro 1.0



BeatBox Pro 1.x



6½

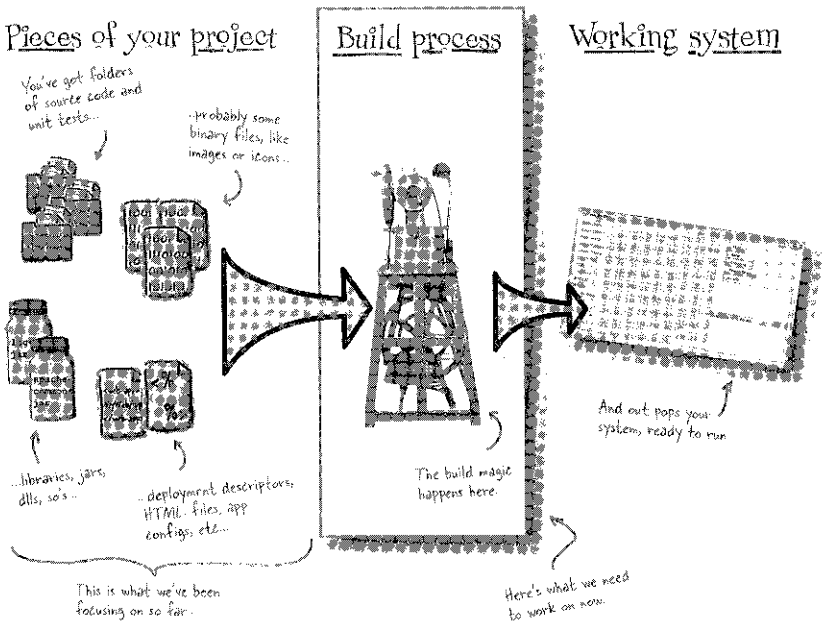
building your code

Insert tab a into slot b...

It pays to follow the instructions...

...especially when you write them yourself.

It's not enough to use configuration management to ensure your code stays safe. You've also got to worry about **compiling your code** and packaging it into a deployable unit. On top of all that, which class should be the main class of your application? How should that class be run? In this chapter, you'll learn how a **build tool** allows you to **write your own instructions** for dealing with your source code.

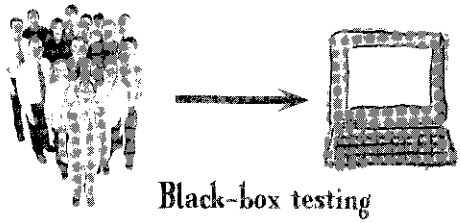


testing and continuous integration

7 Things fall apart

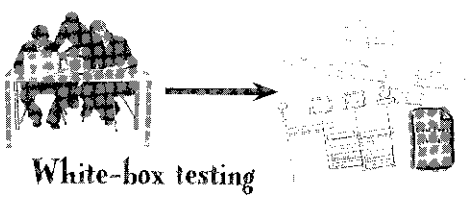
Sometimes even the best developer breaks the build.

Everyone's done it at least once. You're sure **your code compiles**, you've tested it over and over again on your machine and committed it into the repository. But somewhere between your machine and that black box they call a server *someone* must have changed your code. The unlucky soul who does the next checkout is about to have a bad morning sorting out **what used to be working code**. In this chapter we'll talk about how to put together a **safety net** to keep the build in working order and you **productive**.



Black-box testing

Grey-box testing

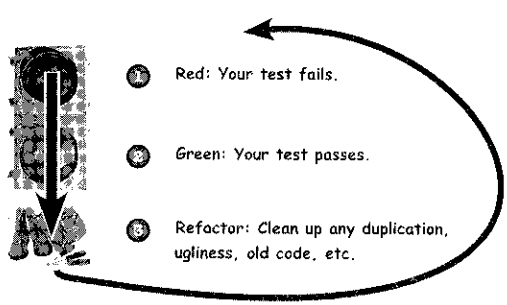


White-box testing

test-driven development

8 Holding your code accountable

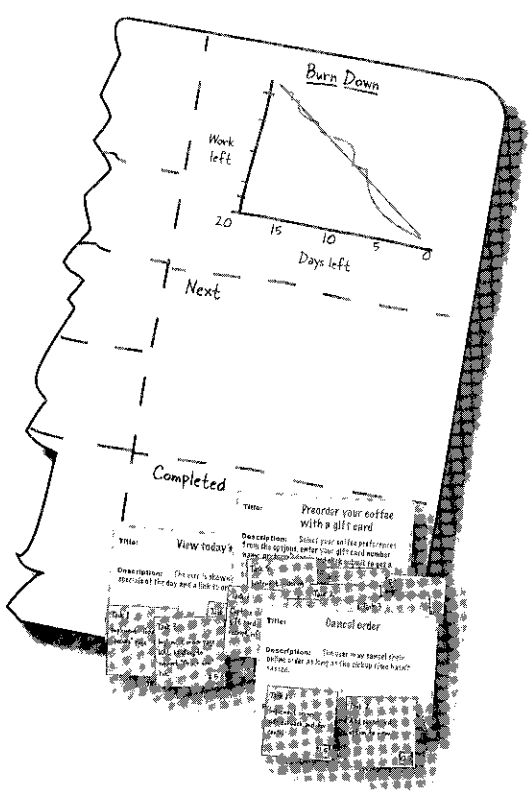
Sometimes it's all about setting expectations. Good code needs to work, everyone knows that. But how do **you know your code works**? Even with unit testing, there are still parts of most code that goes untested. But what if testing was a **fundamental part of software development**? What if you did ***everything*** with testing in mind? In this chapter, you'll take what you know about version control, CI, and automated testing and tie it all together into an environment where you can feel **confident** about **fixing bugs**, **refactoring**, and even **reimplementing** parts of your system.



ending an iteration

9 It's all coming together...

You're almost finished! The team's been working hard and things are wrapping up. Your tasks and user stories are **complete**, but what's the best way to spend that extra day you ended up with? Where does **user testing** fit in? Can you squeeze in one more round of **refactoring** and **redesign**? And there sure are a lot of lingering **bugs**... when do those get fixed? It's all part of **the end of an iteration**... so let's get started on getting finished.

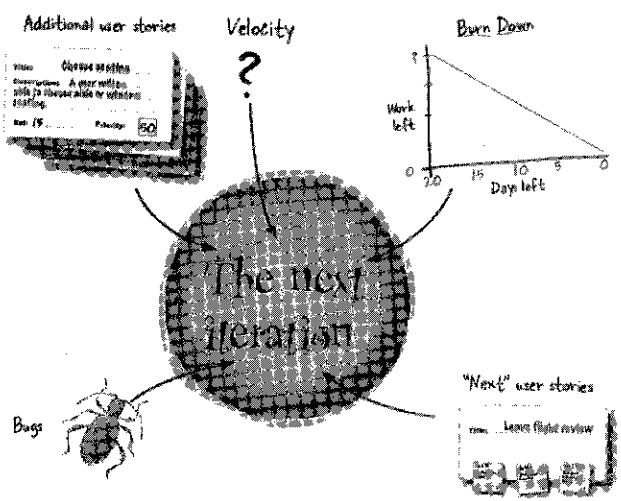


the next iteration

10 If it ain't broke...you still better fix it

Think things are going well? Hold on, that just might change...

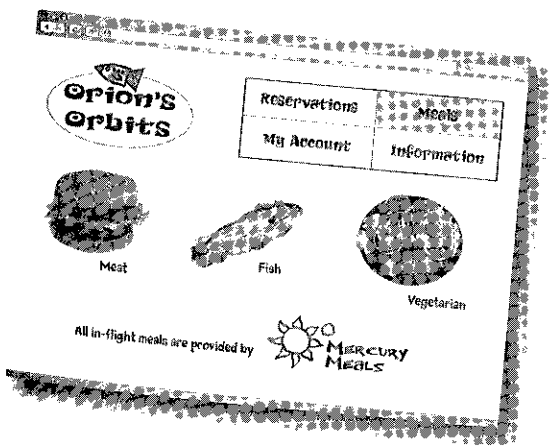
Your iteration went great, and you're delivering working software on-time. Time for the next iteration? No problem, right? Unfortunately, not right at all. Software development is all about **change,** and **moving to your next iteration** is no exception. In this chapter you'll learn how to prepare for the ***next*** iteration. You've got to **rebuild your board** and **adjust your stories** and expecations based on what the customer wants ***NOW***, not a month ago.



11 Squashing bugs like a pro

Your code, your responsibility...your bug, your reputation!

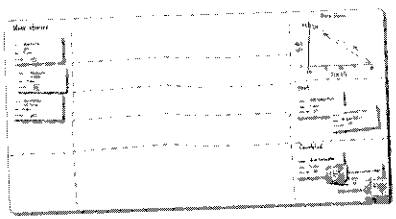
When things get tough, it's **up to you** to bring them back from the brink. **Bugs**, whether they're in your code or just in code that your software uses, are a **fact of life** in software development. And, like everything else, the way you handle bugs should fit into the rest of your process. You'll need to **prepare your board**, **keep your customer in the loop**, **confidently estimate** the work it will take to fix your bugs, and apply **refactoring** and **prefactoring** to fix and avoid bugs in the future.



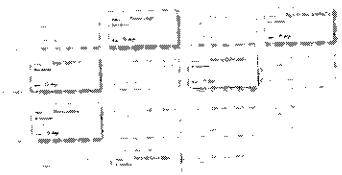
the real world

12 Having a process in life

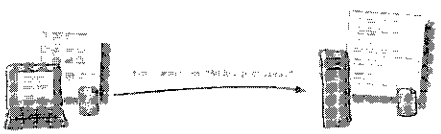
You've learned a lot about Software Development. But before you go pinning burn down graphs in everyone's office, there's just a little more you need to know about dealing with each project... on its own terms. There are a lot of **similarities** and **best practices** you should carry from project to project, but there are **unique** things everywhere you go, and you need to be ready for them. It's time to look at how to apply what you've learned to **your particular project**, and where to go next for **more learning**.



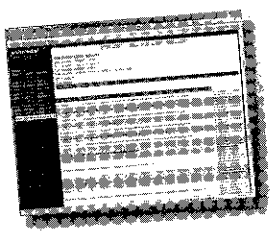
Story and Burn Down board



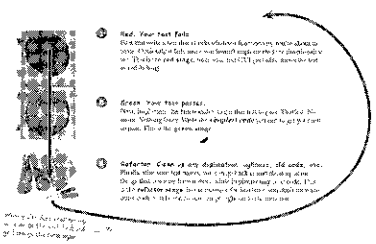
User Stories



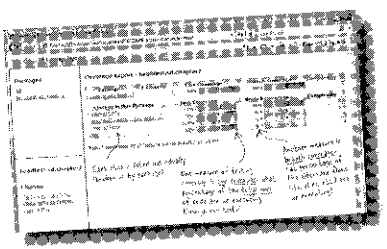
Configuration Management (CM)



Continuous Integration (CI)



Test Driven Development (TDD)



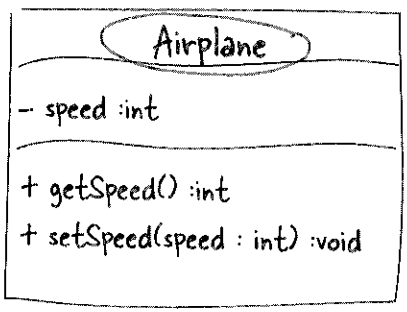
Test Coverage

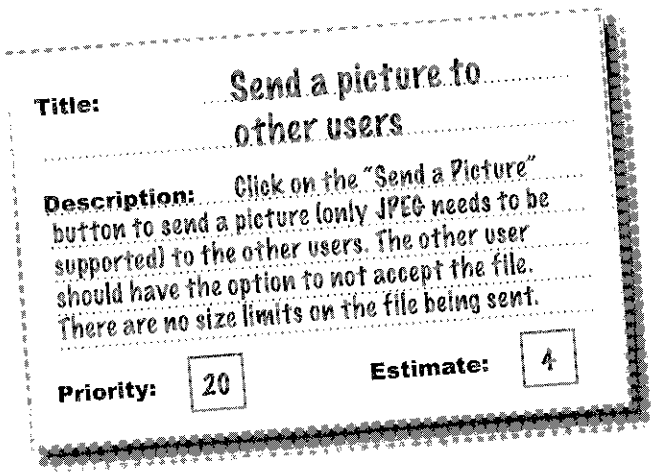
appendix 1: leftovers

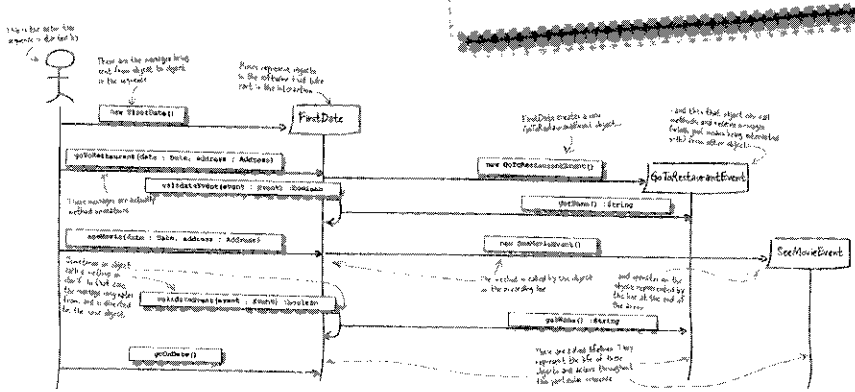
The top 5 things (we didn't cover)

Ever feel like something's missing? We know what you mean...

Just when you thought you were done... there's more. We couldn't leave you without a few extra things, things we just couldn't fit into the rest of the book. At least, not if you want to be able to carry this book around without a metallic case and castor wheels on the bottom. So take a peek and see what you (still) might be missing out on.







appendix 2: techniques and principles

Tools for the experienced software developer

Ever wished all those great tools and techniques were in one place? This is a roundup of all the software development **techniques** and **principles** we've covered. Take a look over them all, and see if you can **remember what each one means**. You might even want to **cut these pages out** and tape them to the bottom of your **big board**, for everyone to see in your daily standup meetings.