Ship It!

A Practical Guide to Successful Software Projects

We are what we repeatedly do. Excellence, then, is not an act, but a habit.

► Aristotle

Chapter 1 Introduction

Many software developers today are frustrated. They work long, hard hours, but their teams can't seem to finish the current project. It's not for lack of effort or desire; everyone on the team wants to wrap the project up cleanly, but no one knows how to pull it all together. It's very difficult to find the time to do the reading and experimentation to find out what works and how to make it work in your shop. Most people are too busy working to embark on this type of research.

That's where *Ship It!* steps in. This book is a collection of basic, practical advice that has been proven in the field, on multiple projects, and in companies of all sizes. It's what we've encountered that works. We're not consultants who were in and out in a few weeks; we worked day in and day out at these companies. We didn't get to drop in ideas that sounded good and then move off to the next engagement. When things didn't work, we were still there to see them fail. On the other hand, we also got to see when things went really well.

Some of these ideas have been blatantly lifted from well-known software methodologies, and we've tried to give credit where it's due. Other ideas were forged from blood, sweat, and tears. We've experimented with many tools, techniques, and best practices, and when something worked, we kept it. When it flopped, we tossed it. Very little you will see here is blindingly original (this is a Good Thing). Instead, we "stood on the shoulders of giants," selecting ideas from the best minds in the industry, and transformed them into what you see here.

Fifty to seventy percent of software teams today don't use basic, well-known software practices ([Cus03]). Quite often, this isn't because they don't know what to do but because they simply don't know how to get the practices started in the here and now. We'll show you how to sell management on each idea, lay out practical steps to get you started, and then offer warning signs to look for so you won't veer off-track.

Ship It! was written by developers who have been "in the trenches." This book is our experience, not theory, ranging from small startups to the largest privately held software company in the world. It's a methodology-agnostic, down-to-earth guide for making projects *work*.

We've tried to model the book after the popular Pragmatic Bookshelf titles: a practical, light, easy read. Our hope is to build on the foundation the other Pragmatic titles have begun.

1.1 Habitual Excellence

So how does Aristotle's quote fit here? "We are what we repeatedly do. Excellence, then, is not an act, but a habit." Excellence isn't defined by turning out one great product (or a number of great products). It comes out of what we do each day: our habits. *Extraordinary products are merely side effects of good habits.*

Applying this quote to ourselves (both professionally and personally) requires that we recognize our lives are side effects of our habits, so we'd better choose our habits carefully. Most people just randomly fall into their work routines, for a variety of reasons: this is how you learned it, it's how your boss used to do it, and so on. We can do better.

Purposely seek out good habits, and add them to your daily routine.

Try this experiment. Find a development methodology to research, and extract one habit that looks good to you (and that can be used by itself). Put it to use for a week. If you like it and it seems beneficial, continue using it for a month. Practice the new habit until it becomes a natural part of your routine, and then start the process all over again. Just as you lay a foundation brick by brick, repeat this process, and build a foundation of excellence one new habit at a time. Never be afraid to remove something that doesn't work in your circumstance, and don't keep a practice just because it's well-known or popular. Forge out your own way, based on what works for you and what doesn't.

"How we spend our days is, of course, how we spend our lives."[1] If that's so, then we must be careful how we spend our days.

[1]Annie Dillard (U.S. author, poet, and Pulitzer prize winner in 1975 for nonfiction)

TIP 1

Choose your habits

Don't fall into habits by accident. Choose your habits deliberately.

1.2 A Pragmatic Point of View

This book is not an academic analysis of why something should or shouldn't work, and it isn't a catalog of available practices and methodologies that you can choose from.

Instead, this book presents what has worked for us on real-life software projects. We would introduce a new tool or practice and use it until it was evident whether it worked. We kept those that worked in our software development toolbox and carried them with us. Eventually, it actually appeared we knew what we were doing! We hope these tools and practices will work well for you also.

We've spent time in startups that didn't have the luxury of using a methodology simply because it was "the right one." Our circumstances forced us to find ideas that solved problems that we could put to work immediately. We've also worked in larger companies that had significant resources and technology at their disposal. We've found that even large companies don't want to use a tool just because it's elegant or because some guru endorses it. They want solutions that solve today's problems quickly and inexpensively. So we picked up a habit here, and dropped a habit there, until our toolkit was generic enough to be portable but still solved problems effectively. This book is a collection of good habits that we've used that will make a difference in your shop as well—the results can be astonishing.

To illustrate: let us tell you a Tale of Two Software Shops (with apologies to Charles Dickens).

The first shop was a mess. They'd purchased rather expensive source code management software but never installed it. As a result, they lost the source code for the demo they were showing to potential customers. No one was sure what features were supposed to be included in the product, but the entire development team was working on it nonetheless. The code was unstable and would crash every five minutes or so (usually at the worst possible moment—during live demos). This mess didn't do much for morale; company meetings regularly spiraled downward into shouting matches. Some developers avoided the situation by hiding in their offices all day long. All in all, it was a *bad* place to work. Everyone knew there were major problems, but nobody could fix them.

The second shop was in much better shape. With about the same number of developers, they were working on three major products simultaneously. These projects had their code in a source code management system; the code was automatically rebuilt and tested whenever it changed. The entire team had daily meetings that were short, professional, and effective. Every developer knew what features to work on because each project had a master plan. They followed the quarry worker's creed: *We who cut mere stones must always be envisioning cathedrals* [HT00]. That is, everyone was able to apply their own expertise and craft within the context of a larger, coordinated framework. Their products shipped on time with a minimum of fuss and bother and were stable because they were well-crafted.

The most amazing thing about these two companies is that they're the *same shop*, separated by less than six months and the application of the principles in this book. (But you had already guessed that, hadn't you?) After the turnaround, the CEO said we had introduced an "atmosphere of excellence" and that he "didn't even recognize the place." This company is one of the more recent places we've worked, and we brought the principles in this book to bear in almost the same form as we're presenting them to you. The transformation we went through there is one of the reasons we decided to write this book for you.

We've discovered and applied these ideas at companies of all sizes, from a four person startup to SAS, the largest privately owned software company in the world. Frankly, we've been amazed at how well these principles have worked at companies of all sizes.

Think of these ideas as the foundation to a great product. You'll reap benefits for the rest of the product's life cycle if you're willing to invest the time up front to get your infrastructure set up properly. Of course, it's easier to start a project with all of these practices in place. Like a cracked foundation of a house, some can be patched easily while others are deeply structural and can be a lot of work to go back and fix.

While you may be in the middle of a project currently, it's never too late to start good habits. You can introduce many of these ideas into an existing project and reap immediate benefits, and we'll cover ways to do that in the last chapter.

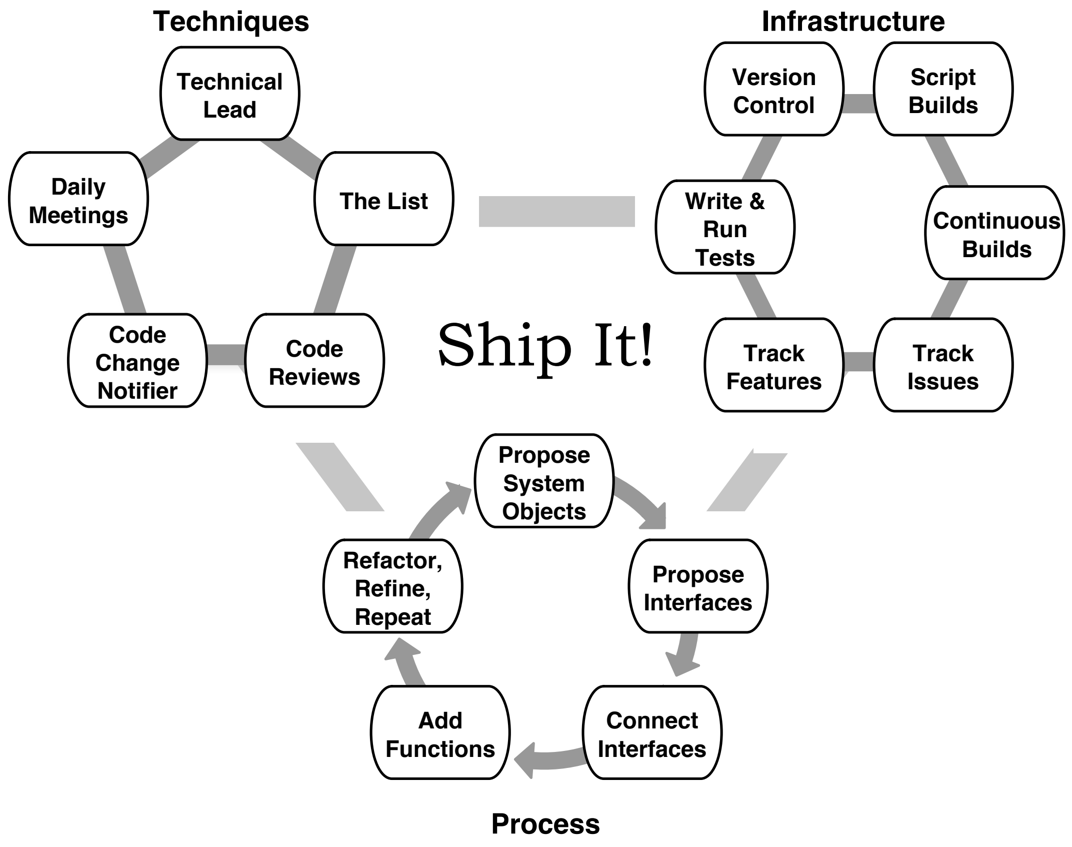

Figure 1.1: How to build a great product

1.3 Road Map

We have arranged our ideas into three main areas: infrastructure, techniques, and process (see Figure 1.1). These areas directly affect your team's ability to consistently deliver the product your customers want.

Infrastructure

In *Tools and Infrastructure* we cover the software tools that will make your life and your team's life easier. For instance, a good source code management system keeps the "crown jewels" of your project—your source code—safe and sound. An automated build system gives you repeatable builds anywhere and anytime you like. And we discuss how to keep track of the bug reports, feature requests, and other issues that come up the moment you let the rest of the world see what you've been developing. Finally, we show you how a good test harness can give you confidence that your code does what you think it does.

Techniques

In *Pragmatic Project Techniques* we cover specific practices that you and your team can use every day to "work smarter, not harder." We tell you how to put a tech lead on your team to insulate you from the outside world and to get you only the information you need to know. Use The List by yourself to organize your own work and teamwide to keep the group on-track. Is your team not communicating? Can't tell who's doing what? Start holding Daily Meetings to keep everyone on the same page while sneaking in some opportunities to pick teammates' minds. Short code reviews help leverage the expertise of your co-workers and let you share a little of your expertise too. And once the review is over, show the rest of your team what you've done with code change notifications.

Process

No book on software development would be complete without a presentation of the authors' pet development methodology, and this one is no different. So we humbly added a plug for what we call Tracer Bullet Development. When you use Tracer Bullet Development you create an end-to-end working system that's mostly stubbed out and then fill in the missing parts to make it real. It's good for splitting large projects apart and letting teams work on the pieces in parallel, and it also lends itself nicely to automated testing.

Common Problems and How to Fix Them

Finally, we present common problems—and danger signs—that arise and offer real-world advice on how to solve them using the tools, techniques, and processes we talk about in the rest of the book. A lot of these problems we encountered ourselves over the years. Some we solved, others we figured out how to solve after the fact (hindsight *is* 20/20, after all...). We hope our experience will keep you from making the same mistakes we did.

What's Missing?

People and requirements gathering are two areas that we didn't include. Good people trump tools, techniques, and process as the most important part of a project; however, assembling and keeping a great team is a subject worthy of its own book (or a series of books!). Instead, we focus on ways to leverage and grow the skills your team already has.

Similarly, learning about product requirements is another deep subject. There are many ways to collect requirements, ranging from note cards to complicated systems full of checks and balances. Rather than attempt to address another large issue that we couldn't do justice to in a single chapter, we chose to present ideas that are flexible enough to handle changing requirements, no matter where you get the requirements from. The ideas in this book can accommodate the project whose requirements never change as well as the project whose requirements are constantly shifting. So you can use these ideas whether you get your requirements list from a small stack of 3x5 cards or a 10,000-page contract.

We've tried to keep the discussions generic enough that you can use them in any shop and with any technology. That's why we didn't add sections on installer technologies or code optimizing tools.

1.4 Moving On

Our hope is that you will use the ideas and habits presented here in the spirit in which they were forged (i.e., pragmatically). Read them, and try them out. Keep what works in your environment, and discard the rest.

Stop after each section to determine whether you're using the idea. If you aren't, then read *How Do I Get Started?* If you are using the idea, read *Am I Doing This Right?* or *Warning Signs* to make sure you're on the right track.

1.5 How Should I Read This Book?

How you approach this book depends on your role in the project. Naturally, when you work as a developer or tester, you will approach the book differently than your team lead, but you can get a lot of value from this book when working in either role.

You Are a Developer or Tester

If you are a front-line practitioner (or implementor), read this book from front to back. Each section contains practical ideas you can use daily as an individual contributor or a team leader. Often developers skip sections with a team focus if they are not team leads. That's a really bad idea. Most team environments are a collection of what team members have requested or have had direct experience doing. Put yourself in a position to know what tools, techniques, and processes can make a positive impact in your shop and be able to present solid reasons for each request you make. Many times we've heard developers argue for a given tool or technique because "it's the right way." This argument never sways management and is, in fact, counterproductive. Before you present an idea, be sure to understand the benefit to the team.

Which request would sway you? "We need a source code management system from Acme Code Systems because it's a good thing and everyone is using it. It's a best practice!" or "We should have a source code management system because it will let us access past releases, roll back specific code changes and allow our developers to work on parallel code trees safely. It's the easiest way to safeguard our company's development investment. Acme Code Systems makes a great product that we should look at. Joe and I have been using it for several months now and it has made a real difference in our productivity. Here's a list of how it's helped us."

You Are a Project Team Lead

Use this book to perform an audit of your team's environment and work flow. (You do this from time to time already, right?) Take the opportunity to reexamine how your team works. Do you have a basic set of tools that cover your foundational requirements? Are your team's techniques building solid product and solid developers? Do you have a clean, well-defined process?

As you review how your team is working, be sure to consider each item's relevance. Are you using tools or practices that once fit but are no longer effective?

Did you hear the story of the woman who always cooked ham by cutting off and discarding a third of it first? When asked why, she said that was how her mother always cooked hams. When asked, her mother said that was how *her* mother had always cooked. They finally confronted

Grandma, who admitted that when she was young, she didn't have a pan big enough for an entire ham, so she always just cut the end off, and it became a habit.

Be sure that your habits are formed out of your needs today, not last year's project or Grandma's eccentricities.

Be sure that your team has access to the tools, techniques, and processes that it needs today. Knowing what works and why is the only way you can effectively guide your team. Each section has tips to help you get started and warning signs to alert you to problems before they get out of hand.

You Are a Manager (or Involved Customer)

Upper-level management can do a great deal to influence how teams work simply by asking for the right information. This book can show you some of the key components your teams should be using and what types of questions you should be asking. For example, when you ask for a list of fixes in the last release, you are saying that you want that information tracked. As you read each section, look for *deliverables* you can ask your team leads to submit that can guide them in the direction you want them to work. Be very careful with these requests, though; you don't want to create bureaucratic busywork for the teams. You want to guide with carefully placed requests.

deliverables

Since you are removed from the day-to-day work, you will probably skim over the *How Do I Get Started?* sections, but you'll want to understand the what and why of each topic.

Individuals Make a Team

Nearly every concept in this book has been used by team members, entire teams, *and* managers. A team member is often the one who first uses the practice, proves its worth, and then shares it with the team. We have done this repeatedly ourselves and seen others do it, and you can do the same thing. Here's a story about someone who did just that.

The Rapid Growth of Agility Support Systems at CafePress.com
by Dominique Plante and Justin McCarthy

When we started working at CafePress.com early last year, management was enthusiastic about adopting agile practices, but the development environment lacked basic support systems required to make changes with confidence.

> Enter the Create and Buy project—an extension of CafePress's core offering that allowed individuals to easily design and buy customized merchandise (t-shirts, mugs, etc.). This project was the first attempt at introducing an explicit business and persistence layer in addition to the web presentation layer. Most of the business and persistence tier was designed test first using the NUnit framework for writing developer tests. Simultaneously, we introduced NAnt for repeatable compilation and deployment of classes used in the web tier. Next, we wired up CruiseControl.NET for continuous integration (i.e., compile and run tests) upon every check-in to our Subversion code repository. For the finishing touch, we commissioned Chicken Little, a small but highly visible (and audible!) workstation running CCTray for build status notification.
>
> The interoperability between Subversion, CruiseControl.NET, NAnt, and NUnit helped us evolve a hospitable collaboration environment without contentious vendor analysis or purchasing decisions. Moreover, these support systems were developer initiated, never because of explicit management requests.
>
> Since we started doing all of this automation, our team has grown, and many new team members have advanced the maturity of our test suite and project automation tools. Some recent upgrades include 100 percent scripted development environment creation, as well as automated test environment deployment; although, looking back, `nant test` is still the most frequently used target.
>
> Before the advent of these tools, most of our daily communication consisted of broadcasting questions about build failures or notifications of API changes. Now that CruiseControl.NET handles our "buddy builds," developers understand and honor a commitment to keeping the build pristine. Nobody enjoys a work stoppage because of an errant check-in. With the support systems in place, our conversations naturally trended toward software design and implementation and away from venting our environmental frustrations.
>
> All of our early efforts aided the immediate delivery of tested code, and our initial investment continues to pay dividends every time Chicken Little squawks!

We feel this is the best way for change to occur. We would never discourage a manager from introducing change, but we find the best and most applicable changes come up from the trenches. The people doing the work usually have a good idea about what specific problems need to be solved.

So we say, "use the book" whether you are a manager, a developer, a tester, or a tech lead. Find the parts your shop (or your own personal work) is missing, and see how it can make your life a little easier today.

Joe Asks...

What Is Agility?

Agility refers to a software team's ability to adapt to changing conditions quickly. This sometimes means redesigning to accommodate changing requirements, and other times it means responding to new bugs quickly or adopting new technologies quickly. Generally, agile teams are more concerned with results than bureaucracy. You can read more about agile software at `http://www.agilemanifesto.org/`.

The following quote from the web site sums up the agile point of view pretty well:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value

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

That is, while there is value in the items on the right, we value the items on the left more."