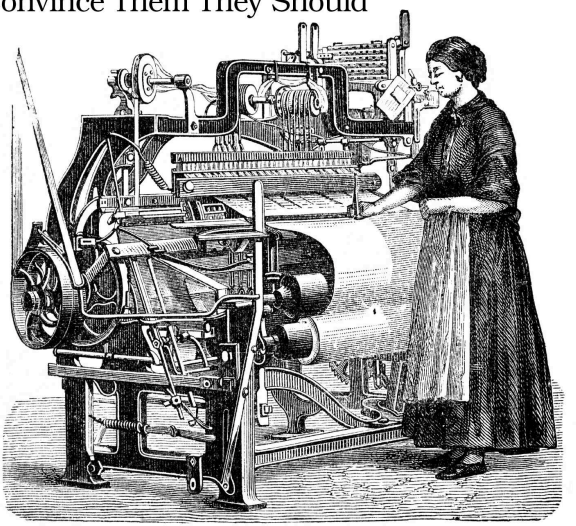
The Pragmatic Programmers

Driving Technical Change

Why People On Your Team Don't Act on Good Ideas, and How To Convince Them They Should



Terrence Ryan

Edited by Jacquelyn Carter

What Readers Are Saying About *Driving Technical Change*

At its core, *Driving Technical Change* is a fantastic book about design patterns. In it, Terrence Ryan clearly outlines common, problematic personalities—"skeptics"—and provides proven solutions for bringing about progressive change. It is certainly an unfortunate fact of human behavior that people are oftentimes resistant to implementing best practices; however, using Terry's book as a guide, you will now be able to identify why people push back against change and what you can do to remain successful in the face of adversity.

► **Ben Nadel**
Chief software engineer, Epicenter Consulting

Politics is one of the most challenging and underestimated subjects in the field of technology. Terrence Ryan has tackled this problem courageously and with a methodical approach. His book can help you understand many types of resistance (both rational and irrational) and make a strategy for getting people on board with your technology vision.

► **Bill Karwin**
Author of *SQL Antipatterns: Avoiding the Pitfalls of Database Programming*

Ryan combines the eye of an engineer, the insight of a psychotherapist, and the experience of a soldier in the trenches to provide a flowchart approach to your most immediate problem, as well as a fascinating overview of how to be more productive and less frustrated with your technical work. *Driving Technical Change* speaks in the language of the people who have the most to learn from Ryan's success with organizational management.

► **Jeff Porten**
Internet consultant and author, *Twentysomething Guide to Creative Self-Employment*

This book covers a very important topic I have never seen covered in book form and answers questions every one of us in application or web development has asked. Terrence Ryan manages to create a fun and easy-to-read narrative with examples so accurate and familiar they that will often leave you wondering whether he was sitting next to you in a recent office meeting.

► **Brian Rinaldi**
Web community manager, Adobe Systems Inc.

Driving Technical Change

Why People on Your Team Don't Act on Good Ideas, and How to Convince Them They Should

Terrence Ryan

The Pragmatic Bookshelf
Raleigh, North Carolina Dallas, Texas

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at http://www.pragprog.com.

The team that produced this book includes:

Editor:	Jacquelyn Carter
Indexing:	Potomac Indexing, LLC
Copy edit:	Kim Wimpsett
Layout:	Steve Peter
Production:	Janet Furlow
Customer support:	Ellie Callahan
International:	Juliet Benda

Printed in the United States of America.

ISBN-10: 1-934356-60-3

ISBN-13: 978-1-934356-60-9

Printed on acid-free paper.

P1.0 printing, November 2010

Version: 2010-11-8

Contents

Acknowledgments

For many reasons, this book would not exist if not for Dave Thomas and Andy Hunt. If I hadn't read *The Pragmatic Programmer* [HT00], I wouldn't have the list of techniques and tools to push for. To go from reading their words to writing for their publishing company was a journey I can't believe I made.

Jackie Carter earned her pay editing this book. She always kept on top of me, chasing me down when I had writer's block and offering ideas and suggestions that made this book much more readable and understandable. Also deserving of much thanks are my technical reviewers, who really helped polish a lot of rough edges: Rachel Davies, Ben Nadel, Karl W. Pfalzer, Craig Riecke, Johanna Rothman, and Brian Rinaldi.

I have to thank all of my colleagues at the Wharton School. Both skeptics and the converted taught me a great deal about how one should and should not go about doing this. I especially want to thank Dave Siedell, Bob Zarazowski, Gerry McCartney, and Deirdre Woods for always being open to and supportive of my efforts even if they were not always convinced.

I also have to acknowledge my current colleagues at Adobe. I work for a group dedicated to evangelism. My first staff meeting was an education measured in volumes. I especially want to thank my boss, Kevin Hoyt, for being a well of good advice. Ryan Stewart is a constant source of inspiration, good morale, and encouragement. Adam Lehman has been a great help and verbal sparring partner. Ben Forta is the original example for me that you must put yourself out there and convince people to try something bigger and better.

Avish Parashar has been my informal mentor in many things around speaking and standing up and being heard. He taught me to leap before I look. Pitching this book is an example of that type of thinking and would never have happened without him.

Mom, Dad, and Casey, thanks for building a home where making yourself better through learning was a noble choice. Jack and Ellie, keeping you in diapers, food, and smiles is worth all the work.

Finally, I need to give tremendous amounts of thanks to my wonderful wife, Janice. You never seem to doubt I can do anything. That confidence is contagious. Thanks for believing I could do it. If it weren't for you, I simply couldn't.

Part I

Introduction

Chapter 1

Why This Book?

I was in the middle of a very typical meeting, with a very typical group, in my very typical company. I was in charge of our web application servers. My responsibilities included maintaining software, maintaining hardware, enforcing best practices, and getting people to upgrade. I was always trying to get people to upgrade.

In fact, we were talking about upgrades.

"I need you guys to set up some sort of schedule for moving your applications from ColdFusion 6 to ColdFusion 8," I said, for the fifth time in as many meetings.

The expected response was delivered with a sigh, "We can't move to the new servers. Every time we move our applications to a new server we have problems and incompatibilities. We just can't have that with our users."

I fired back, "That's a common problem in general. Have you considered using unit tests to be able to have more confidence when you move from version to version? That's not just a problem with an application server but also web servers, database servers, and your code base. You have to move from version to version. Unit tests help with this."

The excuses then flew, "It would take too much time. Why should we have to do this? We don't know how to do unit tests."

I could tell you that I argued with them. I could detail the rest of the argument. I don't really have to do that. You know how it went. I didn't get them on board. I couldn't convince them.

Over my time in that position, I had to try to sell several different advancements and techniques. I had the argument I just described and others like it many, many times. I lost a lot, I won a few, and some just ended up being wars of attrition. I did start to notice certain patterns:

- The same people tend to make the same arguments.
- Some people always went along with new things.
- Other people jumped on to an advancement once others had already converted.
- Some people can never be convinced.
- Certain arguments I made worked on some people but not on other people.
- Sometimes getting management involved was the only way of getting people on board.

I took those patterns, wrote down what I observed about them, and figured out that certain tactics worked better on some than others. I started reusing the same tactics on the same skeptics for different issues. My batting average went up. It became easier to sell advancements.

That's what this book is about—those advancements, those patterns, those arguments. My hope is that what I have to say can allow you to skip all of the go-nowhere arguments, avoid the frustration, and actually drive your organization forward technologically.

1.1 How Is This Book Organized

This book is a patterns book. That means the subject matter is based on a set of repeating forms, or *patterns*. Two main parts of the book are broken into collections of patterns. One part is about skeptic patterns focused on the reason some people resist your efforts. The other is about techniques that can be used to counter skeptics. Ultimately, this means the chapters in these sections are going to be highly structured.

While the two patterns parts are about who your co-workers are and how you should approach them, the final part of this book breaks away from the patterns and talks about strategy. It will help you sort out who to approach first, who to avoid, and how to turn your efforts into real change.

1.2 Why You Should Read This Book

The goal of this book is to enable you to convince co-workers to adopt new tools and techniques. You should be able to do this without having to become some sort of cutthroat office politician. This doesn't mean that you won't need to use politics, just that you don't have to use them evilly.

I will outline a cast of characters; some of them will remind you of people you work with. Once you identify those people, you should be able to match them up with countering techniques. You apply those techniques on your skeptics in a strategy I will lay out. Then change magically happens.

Well, not magically. It does take some work and effort. But it is that straightforward. You should be able to reap some benefits immediately after reading this book. The rest will come as you gain experience doing what this book outlines.

1.3 Who I Think You Are

I think you are a technical person. Perhaps you're a developer or programmer. You could be a server administrator, network engineer, or hardware engineer. Maybe you're a database administrator or even a designer who works with technical people.

It doesn't really matter what type of technical person you are, as long as you do some sort of technical work with other people.

My anecdotes and scenarios are going to be about developer topics. Sorry, that's who I am. However, it doesn't matter what language or tool set you are using. This is going to apply evenly, whether you are a .NET or a Java programmer, an open source fan, or someone in love with some company. This is for anyone who has tried to get co-workers to change the way they work, regardless of how you wanted them to work.

Chapter 2

Defining the Problem

In this book we're going to be talking about selling professional development to skeptics. To get started with that, I think we need to lay out just what we are talking about:

- What do we mean by professional development?
- Who are these skeptics?
- Why do we need to sell it?

2.1 What Do We Mean by Professional Development?

Professional development includes any tool or technique that makes you more productive as a developer, your work less vulnerable to failure, or your code more understandable to your teammates. That covers a lot, so let's get more concrete.

Productivity would at first glance point to things such as automation and code generation that allow you to produce more code in a shorter amount of time. But it can also extend to languages. If you are able to create more functionality in fewer lines of code by using another language, that counts. I'll even go one controversial step further: you can make the argument that you can be more productive through your choice in operating systems.

As for vulnerability, the big thing that comes to mind here is source control. You can't feel safe today unless you are running some sort of source control. But it goes beyond that. Unit tests make you less vulnerable to bugs, as does UI testing. Code reviews can also make you

safer. Basically, anything that helps you sleep better at night or allows you to avoid the *getting hit by a bus* problem makes you less vulnerable.

We often overlook the value of communicating better with your teammates. However, all of the arguments over commenting, tabbing, and variable naming all come down to the core need of good communication. It's important to make it easy for other developers to read your code. That could mean adopting company standards or a code framework. In any case, it's part of the professional developer toolkit, and so it goes here.

Just because I didn't mention a specific technique doesn't mean that it doesn't qualify. When in doubt, ask, does it make me more productive, less vulnerable, or more understandable? If the answer is yes, then you're dealing with some sort of professional development tool or technique.

2.2 Who Are These Skeptics?

The skeptics are by and large your co-workers who are not using the tool or technique that you want them to adopt. Some don't know about it. Some don't care to know about it. Some know about it and just refuse. By and large, skeptics tend to fall into patterns. I'll go into detail later on the actual resistance patterns.

But what's important to get now about the skeptics is that you need to figure out why they aren't using the technique already or why they rebuff your attempts to introduce it. There are lots of reasons. Some may be technical, some may be political, and some are even personal if you can believe that. The important thing to do is to put yourself in their shoes and try to figure out where they are coming from.

Now, I'm careful to use the term *skeptic* and not something stronger like intransigent, shortsighted, hostile, or even some stronger words you might think of. It's easy in the height of frustration to think of them as adversaries, the opposition, or even enemies. It might feel good to vent about them in that way every once in a while, but don't get stuck there. They are your co-workers and friends, and perhaps you have played this role to someone else. Don't lose sight of that.

2.3 Why Do We Need to Sell It?

Selling usually refers to getting people to hand you money for a product. When it comes to selling professional development, though, we're usually but not always looking for another kind of investment. We're usually looking for time or effort. It takes time and effort to learn new things. Sometimes people have trouble seeing the worth of that time and effort, especially if their current tools are inefficiently keeping them working at a frenzied pace. This frenzied pace doesn't give them the ability to step back and see the bigger picture: that they are wasting time using slow methods and tools.

That cost can be a little more subtle sometimes. Programmers tend to define themselves by their language: "I'm a Java developer." or "I'm a .NET developer." Getting a Java developer to try Ruby is more than getting them to spend the time; it's about getting them to rethink their identity, even for a bit. Don't even get me started on trying to get people to try other OS platforms.

It can get even more ephemeral than that. There comes a point when some people think they have gained mastery over their field. Maybe it's when they don't have to look at reference docs anymore, maybe it's after ten years in the field, or maybe it's after an advanced degree. Regardless of the particular milestone, some people think they have all of the answers. You're saying "Something may be an improvement on their methods." They hear "I think you are wrong." If they are wrong now, perhaps they have been wrong for the past few years. Even the most evolved and enlightened people can't always take that, because you are messing with not just their identities but their pride.

In all these cases, you're trying to get more than mere money out of people. You're looking for time, effort, identity shifts, and pride. All of these are more valuable than money. If you think you have to sell to get money, then you'll have to sell even harder to get these.

By now you should have a good introduction of the issues around your tool and technique. You know what it is, you know the people who are in your way, and you know why you need to sell it. However, you need to consider one more important matter: should you be selling your tool or technique in the first place? The next chapter will help us with this issue.

Chapter 3

Solve the Right Problem

Before we make any moves to convince others of our solution, we have to ask ourselves a very important question: *are we solving a problem or pushing a solution?* Solving a problem is good; it's about fixing what ails our groups. But pushing a solution is neutral at best and usually detrimental. But often it is what happens. Why?

In our excitement over the solution we found, we lose sight of the fact that we are trying to address a problem. We forget that most problems have more than one solution. Instead, we focus on pushing our solution, which may not be right for the particulars of the problem. So, despite that our tool can be a fix for our problem, there may be better fixes for the problem that work better with the technical environment, team skill set, or organizational politics.

We have to be open-minded, exactly the way we wish the people we are trying to convince were. You have to gauge whether your solution actually fits. You have to gauge whether another solution fits better. Then you have to be strong enough to let go of your preferred solution in favor of what is best for your team.

Rails Trail

Chris was a Java developer who had been cheating on his chosen language with the upstart Ruby. Specifically, he'd been lured in by a "Build a blog in ten minutes" Ruby on Rails[1] demo. He loved it. Rails made him so productive. He had to get his fellow Java-using co-workers to give it a try.

1. A rapid web application framework for the Ruby language. For more information, see *Agile Web Development with Rails, 3rd Edition* [RTH08].

His co-workers really need Rails. They were slogging it out building application after application with a homegrown framework that required them to do a lot of busy work. Consequently, they were spending lots of time on writing rote code and less time working on a great UI or a sustainable model.

His initial attempts were met with lots of resistance. No one wanted to learn a new language or a new framework. They had also been exposed to some FUD[2] about Ruby. They thought scaffolding was cool, but frankly, the cost of switching languages to get it was just too high.

Chris asked around and found that a lot of people in his group had at least looked at Groovy at one time or another. Chris knew about the Grails project. He spent some time getting used to it. Between his Java experience and his Rails experience, it didn't take too long.

He tried again, this time with Grails.[3] The opposition based on Ruby FUD was gone; the language problem was gone. The only problem was learning a new framework, which the group felt was worth it considering the productivity boost of scaffolding.

Their next project was released using Grails, and now the group is hooked.

As the story illustrates, Chris was trying to sell Ruby on Rails. The group needed a way of writing code more productively to reduce busy work and focus on where they really added value. Ruby on Rails was only one of the many possible solutions to this. There were other options that included rewriting the homegrown framework to include code generation, finding IDE tools that make data modeling easier, and ultimately using Grails. The trick here is to take that next step—see the other options, and weigh them objectively.

3.1 Why Do It?

There are a few reasons why you must think hard about the problem you are trying to solve before you try to sell a solution:

- It requires you to question whether there is really a problem.
- It forces you to think about the problem from your audience's perspective.

2. Fear, Uncertainty, and Doubt; see the sidebar on page 80.
3. A rapid web application framework much like Rails but for the Groovy language. For more information, see *Getting Started with Grails* [Rud07].

- It makes you come up with an answer that is more of a custom fit to your audience.

You have to question if there even is a problem. In the previous story, there happened to be one, but it's possible that the group was already using Grails. If that was the case, then you have to ask yourself, what would Ruby on Rails offer that team? Honestly, not a tremendous amount when you consider the cost of switching technology platforms. It's at this juncture that you figure out whether you are a concerned co-worker trying to improve the work lives of your team or a enthusiastic fanboi trying to convince others to love your new toy. After you are sure there is a problem, you then can define it and make sure that it's an issue worth solving.

Once you're sure there is a problem, you then have to consider whether it is worth fixing and what would make it worth fixing to your team. Perhaps there is a lot of busy work in the group, but perhaps they push it off to junior developers and use it to train them, while the advanced people focus on the model and UI. In that case, it's possible that the group's current solution of the problem creates more value than you think. You then have to modify your reasons for selling a solution. You also have to figure out what you're going to do with the junior developers now that their training exercises are gone.

Also, figure out whether you are pushing a custom solution when there is a off-the-rack solution. Whether in tailoring or IT, custom solutions are always a premium product. The reason in both worlds is the same. Custom solutions have success built into them, because they mold to the contours of the landscape. By ensuring that your solutions fits your group, you remove pain points, or uncomfortable bunching.

3.2 Seeing Solutions

The most difficult part of solving the right problem is seeing past your desired solutions to alternatives. It requires opening your mind and letting go of preconceptions. There are a few things you can do to make this easier.

Research the Problem

Look at how other organizations are solving the same problem that you are seeking to solve. When you search, make sure you search for the problem and not a specific implementation.

Look up the following:

- Source control, not SVN
- Rapid application development, not Ruby on Rails
- Rich Internet applications, not Flex
- Object relational mapping, not Hibernate

Sometimes, however, you can't do it. You can't find materials for your broad problems. Then search for your solution in another technology:

- Visual Studio equivalent in OSX
- Open source version of Exchange
- Safari for Linux

Take Inventory

The next thing to do is take inventory of the skills and ideas of your team. As Chris did in the story, walk around and talk to people. Find out what they know. Ask what they think about the problem. You might have your suspicions confirmed. Or, you may find out a completely different route to take. In any case, even if the inventory yields no inspiration, at least you will know people's comfort level with any possible solution you throw at them.

List Options

Force yourself to list alternatives, even if they aren't actually better. Have a list that you can reference of other solutions that you have researched. People don't believe you when you say, "There are no alternatives." You can say your solution is the best, but you can seldom claim it is the only one. Invariably there are alternatives, and you have to at least consider them, even if you end up rejecting them.

This also has the side effect of making your arguments more compelling. You researched, you prepared, and you're not just picking the first thing you thought of. This will make an impression on your audience.

3.3 Challenges

The main challenge here is that you could fail to really consider alternatives objectively. The more you consider alternatives fairly and the

more you combat zeal as outlined in Chapter 14, *Deliver Your Message*, on page 62, the more open your mind becomes and the less likely you will fall into the trap.

Another difficulty is that this burns up time. If your organization needs to come up with a solution fast, then you might not have the time to do this properly. The risk of this is low, though, because most organizations will maintain the status quo before jumping to a new solution if you can point out that it hasn't been analyzed yet.

3.4 Things to Try

If you are having trouble considering the alternatives, here are a couple of things that you can try to see more:

- Do research on the beginnings of your solution to see what the creators were trying to solve with their efforts.
- Start to learn an alternative to your solution. You don't necessarily have to become an expert, but be able to fool around with a competing solution.
- Play devil's advocate. Imagine that you hate this solution and want to do everything you can do to stop your company from implementing it. What arguments would you use? Then ask yourself whether there are any of these legitimate showstoppers.

It seems a little obvious that you should make sure that what you are pushing actually helps and fits your organization, but many people fail to do this and suffer the consequences. Enough people in our industry have suffered through projects because a decision maker has fallen prey to not solving the right problem. Don't be that guy or gal—push only what would be good for your organization. Not only will it be easier, but it will also be the right thing.

Part II

Skeptic Patterns

Chapter 4

Who Are the People in Your Neighborhood?

The next set of chapters will take you through the patterns that skeptics of technical change tend to fall under. The idea here is that you will look at the patterns and start to identify who from your life matches these patterns.

Once you know who you are dealing with, you can figure out how to deal with them. Later, we will match these skeptic patterns to countering techniques. But first, you must know who they are.

These are the skeptic patterns are:

- The Uninformed
- The Herd
- The Cynic
- The Burned
- The Time Crunched
- The Boss
- The Irrational

At first, you might have trouble identifying who is what. As you read the descriptions, you may find yourself saying "George is sort of a Cynic, but he's not that bad." That means George is a Cynic, severity notwithstanding. The patterns, as portrayed in the book, are exaggerations.

These are supposed to be illustrative. What's key here is the type of behavior, not the magnitude. Most people are professional enough to not be like the over-the-top examples listed here.

Also, don't get caught up in the details of whether someone is being a Cynic or Burned. People can be more than one type of skeptic at the same time. They usually are. For instance, the Boss and the Time Crunched often coexist. Cynics about one technology are often Burned on other technologies. This is a good thing; most of the skeptic patterns have overlapping techniques for overcoming them. You can then focus on those overlapping techniques to affect multiple skeptic types for the same amount of effort.

If you are having trouble identifying people, remember some tips:

- The Uninformed cannot also be Burned.
- Most people are a little Time Crunched.
- The Herd are hard to see as skeptics.
- The Irrational will often masquerade as other skeptics.

Now, let's meet our everyday, ordinary, neighborhood skeptics.

Chapter 5

The Uninformed

It's 6 p.m. on a Friday evening, and you're working with a co-worker whose hard drive crashed yesterday. He's rebuilt his machine and restored from backup, but he's lost everything he did Wednesday and Thursday including some important bug fixes. You point out that he can still pull down his changes from the source control server. He then asks this question:

"What's source control?"

Of course, he doesn't use source control; in fact, he's never heard about it. He's seen your emails about the new Subversion server,[1] but seeing the emails and reading them are not the same thing. He didn't think it was something he had to worry about.

Whoever said "ignorance is bliss" has never tried to kludge together the contents of a restored backup and a folder named main -- copy20070811 at 7 p.m. on a Friday night. Those of us who have know that ignorance is *not* bliss; it's getting home at 9 p.m. after listening to gripes from a co-worker who expected the office expert to know how to protect them from themselves.

5.1 Why Don't They Use the Technology?

Usually, it's because they don't know about it. Some haven't run across it yet; some may have encountered it but didn't understand the problem

1. A centralized version control system. For more information, see *Pragmatic Version Control using Subversion, 2nd Edition* [Mas06].

it was solving. More importantly, they didn't realize that they had the problem that the technique solves.

Whatever the reason, they need but know not that they need.

5.2 Underlying Causes

There are many reasons why people don't know about the technique. Not everyone reads blogs and keeps up with industry best practices. Sometimes this ignorance is willful; some people are just nine-to-fivers and don't care about their craft. However, usually it's just that it's a big industry, there is a tremendous amount of information out there, and unless you're looking for it, you don't necessary run across everything.

5.3 Effective Countering Techniques

The Uninformed generally don't go to the mountain, so you'll have to bring the mountain to them. The techniques that are most effective on the Uninformed are those that provide information to them, described in these chapters:

- Chapter 13, *Gain Expertise*, on page 53
- Chapter 14, *Deliver Your Message*, on page 62
- Chapter 3, *Solve the Right Problem*, on page 21
- Chapter 15, *Demonstrate Your Technique*, on page 68
- Chapter 18, *Get Publicity*, on page 85

5.4 Prognosis

First the good news: it is extremely easy to change the Uninformed to another one of the resistance patterns. All you have to do is tell them about it—BOOM, they're no longer uninformed. The bad news: they are more likely to become one of the other skeptic patterns than they are to become converted.

This is why I wouldn't just rely on sitting them down and giving them "the talk" about version control or unit testing. You need to use countering techniques, especially *Demonstrate the Technique*, to show these individuals not just the *what* of the technique but the *why*.

Chapter 6

The Herd

Today is the day of the big code review. You've wasted your time sifting through 200 printed pages of code. Most of it is routine CRUD[1] code that could have been swapped out for some generic objects or an ORM[2] solution. You're ready for the big fight that comes when you have any major criticisms at a code review.

It's Hector's first major app with the group. He's been out of school for about two minutes, and this is his first major effort with your company. He's already a bit twitchy at the onset, and you're dreading the reaction he's going to have when you tell him, "Some generics would eliminate about 75 percent of his work."

The time comes, and you confront him. Instead of anger, arguments, or tables flipped over in anger, he just looks you straight in the eyes and says, "I'm allowed to do that? I wanted to, but I didn't think you all would let me." That feeling of shock and disbelief you feel—because someone who actually knew how to do it right wasted their time doing work they knew they didn't have to do—that's pretty common around the herd.

6.1 Underlying Causes

The Herd are who they are because they are followers and not leaders. Balance between leaders and followers is good for an organization. Too

1. It's stands for Create, Read, Update, and Delete. It's basically what every application that touches a database has to do.
2. Object relational mapping, a technique that maps records from a database objects in an object-oriented system.

many leaders will lead to an organization not being able to move forward because the leaders are trying to get the group moving in their individual directions. But more often groups end up with too many followers and not enough leaders. When this occurs, the situation slides into status quo. Ironically, the Herd, who are usually followers, lead the group there, by doing nothing.

Within the Herd there are two main groups:

- Those who don't know they can lead
- Those who have other priorities

People who don't know they can lead are commonly younger. They are new to the workforce and haven't come from a background that encouraged self-motivation. Often they have ideas and want to do new things, but they haven't learned the secret to leadership: you can be promoted to management, but no one appoints you a leader.

The people with other priorities can often be disparaged by more passionate developers. They are the nine-to-fivers. They work to earn a living, leave work at work, and go home and devote themselves to other pursuits: family, hobbies, community, and so on. They don't see value in keeping up with the latest technologies themselves.

6.2 Effective Countering Techniques

The Herd aren't going to seek you out to be led. You have to go to them and lead. Techniques that encourage them to advance are mildly effective, but ones that force them are even better. Additionally, forcing them to advance isn't viewed as negatively by the Herd as they would be by other groups. They want the leadership. Therefore, many techniques are effective on them. Read the following chapters for techniques that work with the Herd:

- Chapter 13, *Gain Expertise*, on page 53
- Chapter 15, *Demonstrate Your Technique*, on page 68
- Chapter 21, *Create Something Compelling*, on page 101
- Chapter 20, *Build a Bridge*, on page 95
- Chapter 18, *Get Publicity*, on page 85

6.3 Prognosis

The Herd are the second easiest group to move. Basically, you lead them, and they follow. It's simple but still requires work. It will take the effort of leadership from you. This might seem like an small cost to pay, and up front it is, but leadership wears on you. Think of it like maintenance costs in an application. Building applications is cheap compared to maintaining them. An ongoing leadership or mentoring relationship has higher costs than the initial conversion. I'm not discouraging that relationship, just pointing out that it doesn't come free. Also, if you can pick up some younger team members who can become leaders and add them to your cause, the investment is well worth it.

Chapter 7

The Cynic

You've just spent twenty minutes doing a presentation for your teammates on adopting source control. Yeah, they don't do source control at all. Yep, not at all—it's as if the last twenty years of computer science never happened. But better late than never, and frankly any source control is better than none, because disaster is one errant delete away. You did your homework, and you've tested for yourself. You tried and retried a number of solutions, but the combination of this particular group, solution maturity, availability of tools, and industry acceptance have led you to choose Subversion. Anyway, you've rocked the presentation, and now you're taking questions.

"I've heard that Subversion is yesterday's news. With Git on the rise as the next big thing in source control, do we want to do Subversion just to change in two years?" asks Cindy.

You were ready for that, "Great question, Cindy. I reviewed a bunch of solutions including Git. The long and short of it is that in our group we need tools that integrate into Eclipse. It's my opinion that the Git tools for Eclipse are just not there yet. They might be there in a year or two, but we need a solution yesterday. So, Subversion is more correct for us today. If we do need a feature set that Git provides, there are tools for migrating from Subversion to Git, so I'm not worried."

You assume that such a tome of an answer will be enough, but still Cindy fires the questions at you.

"I've heard that Subversion adds all of this extra metadata to a project, and the bigger the project, the more extra metadata gets created. I hear the metadata is easily corruptible."

"It's an issue, but I think most of our projects are small enough that it shouldn't be a huge problem. There are best practices for dealing with our larger projects," you fire back.

It goes back and forth like this for awhile. Sometimes you nail the answer, sometimes you don't. Sometimes you wonder if Cindy didn't just call up the Wikipedia article for Subversion and just scrolled down to the "Current limitations and problems" section. How else does someone who has never touched Subversion know about the downsides of it?

At the end Cindy announces with assumed authority, "With all of these issues and considering we've been doing well without it, I don't see why we need to add extra complexity to our environment."

Other people in the meeting may not completely agree, but they've been sitting watching your verbal tennis match for a while. Much coffee was consumed. So, now you're in a fight with Cindy, with people's attention spans, and with their bladders. What should have been a slam dunk is now a political battle.

It's a common story when you're dealing with a Cynic.

7.1 Underlying Causes

There are a lot of causes to this behavior. Some people simply like to argue. Others like to prove that they are smarter than someone else. Others have worked in industry for a while and have been repeatedly disappointed and therefore never see the upside of anything.

But there is one really important reason that you will encounter this: *in our industry, this behavior is rewarded.*

Most of our currency in this industry is based on what we can produce with our minds. Smarts are important, but more important than *being smart* is *looking smart.*

Being smart can happen anywhere, doesn't require an audience, and therefore often goes unnoticed. In fact, the distractions of being in front of people often sabotage smarts. That's why you usually prepare well before you talk to a group about an issue. You can make reasonably sure you'll deliver your full brain power to an issue in the privacy of your own cubicle, but in front of a crowd, you'll have unknown variables, nervousness, and often bad luck keeping you from your full potential.

Criticism Is Good. Cynicism Is Bad

Semantics are a tough thing. One person's cynicism is another's due diligence. I don't want you to think you need to handle any criticism as an attack of a Cynic. You should justify your tool or technique. People shouldn't blindly accept that you're steering them the right way.

However, the criticism I describe here as cynicism isn't trying to defend people from poor choices. It's designed to block progress for blocking progress's sake or to score cheap points. Some of what I say here should help you differentiate. But at the end of the day, it's like Potter Stewart said about obscenity: "I know it when I see it."* It's hard to define but easy to feel; sometimes people are criticizing to achieve the best outcome for all, and sometimes people are being cynical to keep themselves from having to grow or to make themselves look good at someone else's expense.

*. Potter Stewart was a Supreme Court Justice who when deciding a case about obscenity famously said, "I shall not today attempt further to define the kinds of material I understand to be embraced within that shorthand description ("hard-core pornography"); and perhaps I could never succeed in intelligibly doing so. But I know it when I see it."

Looking smart, on the other hand, requires an audience but delivers the impression that you can be smart without necessarily having to have the mental horsepower to actually pull it off.

What does all of this have to do with our healthy cynic? There are two ways to look smart:

- Be very smart in front of an audience (which we've established is tough).
- Be smarter than someone else in front of an audience.

The second one is pretty easy to do, and this is where our Cynic comes in. In challenging you every step of the way, they are keeping up with you, which makes them looks smart. All it will take is one weak spot in your prep for them to look smarter than you.

7.2 Effective Countering Techniques

Countering the Cynic is about two things:

- Refusing them entry points to arguments
- Preparing enough so that you cannot be refuted

Refusing them entry points is about not allowing them to ask the questions in the first place. You can deliver your message smoothly. You can anticipate likely questions and answer them as part of your pitch. If you've done this correctly and they do ask a question, you can put them off until you answer it as part of your prepared material. This allows you to control the conversation, not them.

On the other hand, you can't prepare for everything. You can't anticipate every question. First answer their questions authoritatively; even if you don't know the answer, say "I don't know" with confidence. Going further, gain and use knowledge of the subject to prevent them from having opportunities to look smart at your expense.

To those ends, the following methods are especially helpful with these skeptics:

- Chapter 13, *Gain Expertise*, on page 53
- Chapter 14, *Deliver Your Message*, on page 62
- Chapter 3, *Solve the Right Problem*, on page 21
- Chapter 15, *Demonstrate Your Technique*, on page 68
- Chapter 21, *Create Something Compelling*, on page 101
- Chapter 18, *Get Publicity*, on page 85
- Chapter 20, *Build a Bridge*, on page 95

7.3 Prognosis

Assuming that you can prepare and refute as outlined here, you can usually do well by this group. With your delivery and expertise, you can make your solution *the smart solution*. By positioning your solution as the smart one, it follows that questioning it makes them less than smart. Their self-imposed pressure to look smart will keep them from sniping at you. Who knows, if you make your solution look smart enough, they might even become your biggest boosters.

Chapter 8

The Burned

A small group of developers has gotten together to plan the next project. Now is the time you can really shape what tools and technologies will get used. You've been itching to get Hibernate used on a company project, and this is the perfect case. The application is going to be one giant administrative front end for a straightforward database. It's complex enough to warrant ORM but not so complex that the team will have crazy problems with it.

You make the suggestion to use Hibernate.[1]

"I used Hibernate before...," chimes in Bernard.

You're overjoyed; it doesn't take a lot to create a consensus with this group. If Bernard is with you, it will be a snap.

Bernard continues, "It was horrible. The system ground to a halt. It's terribly inefficient. To do a single update, you'd have to pull an entire row out of a database and all of the related rows in other tables. All that to accomplish what you can do with a single update statement. No thanks!"

Like Lucy to your Charlie Brown, Bernard has given you hope and snatched it away. His criticism pretty much stops the group from moving forward—not because he is necessarily right but because he is experienced. He's been burned by the technology before, and now he's never going back.

1. A popular ORM for Java.

Change Weary

Early in the process of writing this book, I considered an additional skeptic type named the Change Weary. The more I thought about it, the more I realized the Change Weary were really just Burned.

Good change can be a lot of work but is seldom demoralizing. You want a break afterward but not permanently. Bad change can cause the Change Weary. But often bad changes only create easy-to-detect Burned people.

The basic problem is that Change Weary people are a result of neither good nor bad change but change that doesn't really result in anything other than a change. Their burn, then, isn't dramatic. It's not that they've seen this before, and it was terrible; they've seen it before, and it was lame.

Change with no positive results is bad change, even if the results were yawn-inducing instead of catastrophic. So, the Change Weary are Burned, even if they were just slow Burned.

8.1 Underlying Causes

This one is pretty straightforward. The cause of this behavior is that they have used the tool you are pushing before, and it didn't work for them. The problems they encountered can run the gamut. They could have used the technology and seen no significant advantage. It's also possible they tried it and had a spectacular failure. They might have installed it but had no idea how to get it to do anything. In any case, regardless of the quality of their attempt, they have credibility with others, because their knowledge is firsthand. This makes their skepticism both believable and more challenging to counter.

8.2 Effective Countering Techniques

The first key to countering the Burned is to understand what happened to them. If you have experience with Hibernate, you know what happened to Bernard. Somehow in the previous use of Hibernate, someone made some mistakes with their fetching strategy and failed to lazy load properly. It might have been Bernard, it might have been someone else, but the effect was the same. Their *implementation* of Hibernate was

severely flawed. It's not possible to tell from this example, but it is also possible that Hibernate was not the correct tool to use on that previous project. If that is the case, even if implemented properly, Hibernate would have still failed.

Next you have to figure out whether your proposed solution is the correct one. Is it worth changing the Burned's mind? Would it be better to choose another technology that can perform a similar function but might get more readily adopted by the Burned?

Finally, you have to pass on your information to the Burned. You have to make sure that in perhaps suggesting that their previous implementation was flawed, you are not telling them that they screwed up. They have to believe you. You have to walk a fine line in communicating with them.

For those reasons, the following techniques work well on these skeptics:

- Chapter 3, *Solve the Right Problem*, on page 21
- Chapter 13, *Gain Expertise*, on page 53
- Chapter 14, *Deliver Your Message*, on page 62
- Chapter 15, *Demonstrate Your Technique*, on page 68
- Chapter 21, *Create Something Compelling*, on page 101
- Chapter 18, *Get Publicity*, on page 85
- Chapter 17, *Create Trust*, on page 79
- Chapter 20, *Build a Bridge*, on page 95

8.3 Prognosis

This can be a tough group to completely convert. People tend not to like spending time learning new things. To try something new and then have it fail can simply be too much of a hassle to give it another try. It is often easier to go around these previous experiences by selecting an alternative technology. But with lots of effort, support from others, and the right Burned co-worker, you can turn them around.

Chapter 9

The Time Crunched

Let's go back to that meeting I had in the introduction, Chapter 1, *Why This Book?*, on page 15. I was discussing with other managers the need to move to a new version of the application server.

Their main objection was that they had code that they were confident in, and moving to another server would require them to test and potentially miss bugs that could embarrass them in front of their constituents. I suggested unit testing as a solution to this, and their first objection was "It would take too much time."

In fact, almost every argument I ever had with this group boiled down to that they didn't have enough time. It didn't matter that many other problems within their group would have been mitigated by unit tests. It didn't matter that the reason they didn't have more time was that they were dealing with issues that would largely have been solved by having unit tests. They suffered from that philosophy of "There's never time to do it right, but there's time to do it twice."

9.1 Underlying Causes

A lot of this type of thinking comes from people being shortsighted, penny-wise and pound-foolish, or otherwise unable to see the real cost of things. To be fair, it can be hard to see past the weeds when you're stuck in the middle of them. But most of the time people are unwilling to alter schedules to accommodate something new, especially when they are already dubious of the result. Think of it this way—their current method may waste time, but they know exactly how much time they'll waste. A new method will definitely cost time to learn, and it might not

make up the time it costs. That would leave them behind and more time strapped than before.

Going deeper, some people are Time Crunched because the amount of work they have to do exceeds their resources. Understaffing, over-promising, or other forms of poor planning contribute to this type of environment. These people are Time Crunched all the time. Finding some breathing room for these guys is tough.

Others, though, are not Time Crunched all the time. There are many industries that are cyclical in their demand. It goes up and goes down over time. Take higher education in the United States. The main part of the school year starts in September and goes to December. It has a short break and then starts back up in January and runs to May. There's a short break, and a lighter summer session starts. People who develop new applications at schools are crazy-busy in the lead-up to September. They are busy over the winter break with upgrades. They then start new projects over the summer. The rest of year they are maintaining and patching their applications. New ideas need to happen during their lulls and not their spikes. Therefore, pitching to these developers in August will always yield a Time Crunched response. In the end, the cause is the cyclical nature of their work, not any flaw in their thinking.

Finally, there really are some types of work that are Time Crunched, and doing it on time beats doing it right. I did programming work for the local office of a national political campaign once. They needed a basic database CRUD application. I started 24 days before Election Day. The day after Election Day, the application was worthless. Any work that went into maintainability or extensibility was worthless. Trying to sell those types of features would have been a waste of time.

The trick here is to figure out whether the Time Crunched thinking is coming from flawed thinking, bad timing, or the particular type of project. Most of it comes from the first. However, making sure you are not selling it at a bad time or to an audience that will never listen will help save *you* time.

9.2 Effective Countering Techniques

Success with the Time Crunched all comes down to making them see your technology or tool as relief for them. They need to see it work. They might need to be bribed with some other time savings. In any

case, they need to be assured that if they choose your method, they will get back more time than they put in. Because of that, the methods in these chapters work with them:

- Chapter 3, *Solve the Right Problem*, on page 21
- Chapter 15, *Demonstrate Your Technique*, on page 68
- Chapter 16, *Propose Compromise*, on page 74
- Chapter 21, *Create Something Compelling*, on page 101
- Chapter 20, *Build a Bridge*, on page 95
- Chapter 19, *Focus on Synergy*, on page 91

9.3 Prognosis

No one likes being crunched for time. It's an awful feeling that usually makes you feel constantly a little nauseous. They want out; they just might not know how to get there.

In any case, with these guys, it comes down to quid pro quo. Save them time, and they'll play ball with you. The tough part is convincing them they will actually save time. If you can do that, however, their desperation to get out of the time crunch can work to your advantage.

Chapter 10

The Boss

You've been called to your boss's office for a somewhat regular status meeting. It goes normally. She talks, you listen. You talk, she checks her BlackBerry. You talk about the projects you have been working on and the projects you have finished. Once you're done with that, you have the opportunity to tell her about your special baby.

You've been working on an automation engine that can introspect a database, build a CRUD application, and style it with a company-branded look and feel. After it is built, you can spend a little time tweaking the model and tweaking the UI to perfect your application. It's simple scaffolding, but it's awesome.

You tell your boss about it. She doesn't seem to get it. She just asks how much time you've been spending on it. You tell her, and she goes from not interested to downright pissed. She tells you to stop working on it. Then she dismisses you so she can meet with her next subordinate.

You've gone from feeling like a huge winner to being told to kill your baby and get out of her office. That's what can happen when you try to sell professional development to management.

10.1 Underlying Causes

The single biggest reason that management resists professional development techniques is because management doesn't understand them. That might sound harsh, but it isn't. It's not their bailiwick. Even if they were promoted from the ranks of developer, if they aren't doing development anymore, they may wonder why you need this thing that they never needed to do their job. If they still develop while managing,

Isn't This Insincere?

The difficult part here is meeting management where they live, talking to them in a way they will listen. Some people will refuse to do this because they think it is somehow fake or insincere. But most of the time we do these sorts of things anyway.

Cashiers say "Please" and "Thank you" despite that they are the ones providing service. We go to job interviews in suits, despite that most programmers detest working in them. We try to learn a little of a foreign country's language before we visit.

In short, meeting people where they are and talking to them in their language isn't insincere; it's practical. You have to meet people where they are, not where you want them to be when you are the one requesting things. In the case of management, it is just one more acquiescence we have to make to people with more power than us. Considering how valuable it can be, it's foolish not to do it.

then they are almost definitely Time Crunched (see Chapter 9, *The Time Crunched*, on page 41) and will probably be closed to your technique on that basis.

Now in fairness to them, you are part of the problem. You see, developers have developer problems. You are used to talking about your tool as a solution to a developer problem. Management has management problems. You have to talk about your tools as solutions to management problems.

"Making code more maintainable" can be sold to management as "reducing ongoing project costs." "Automate rote code" can become "Complete projects faster." Yes, it can lead to talking in buzzwords and marketingese, but it can get you what you want too.

10.2 Effective Countering Techniques

Changing management's mind comes down to changing the way you talk about your tools and techniques. Don't talk about lines of code and n-tier databases. Talk about less developer time and less downtime time. Or go even further and convert it straight to cost savings.

Additionally, you can tie your solution to an overriding concern that has management in a lather. If a new government regulation requires compliance and your tool can help with that, then you can always sell it to management in that way. Finally, for some reason, management tends to value the opinions of outsiders more than internal ones. Getting outsiders to speak well of your solutions can go a long way. For all of these reasons, the following techniques are the ones you want to focus on with the Boss:

- Chapter 14, *Deliver Your Message*, on page 62
- Chapter 15, *Demonstrate Your Technique*, on page 68
- Chapter 19, *Focus on Synergy*, on page 91
- Chapter 18, *Get Publicity*, on page 85

10.3 Prognosis

Success with the Boss is going to rely on how well you make your tools the solutions to their problems. Usually if you can do that, they are more than happy to come on board. This is really fortunate, because they are the ones who can mandate things. They can force others into coming around to your solution. This is part of the larger strategy of converting your organization.

Chapter 11

The Irrational

The Irrational can be the most difficult skeptic to deal with. Every other skeptic type, at the heart of their opposition, has a logical and rational argument. For the Burned, their premise is "This failed before; therefore, it will fail again." For the Time Crunched, it is "I have so much work currently that I cannot afford the time it would require to use this new technique." That doesn't mean they are right, just that they are grounded on a premise. That premise can be examined, challenged, and ultimately defeated. You can convince the Burned that past failures don't mean every use of your technique will be a failure. You can convince the Time Crunched that they can afford that time, and so on.

This is not so with the Irrational. They don't want to use your technique or tool. The reasons can be wide and varied, and we'll see in a moment they don't matter, but there is no rational premise. There is no argument that can overcome that. Any argument is designed to get you to stop trying to convince them. Therefore, they will say anything it takes to get you to stop.

C'mon Irene!

You're arguing with Irene again. The last argument you had with her was over database indexes. She claimed that they are not necessary. She further claimed that "the performance hit you take is worth the freedom you gain by not having to manage indexes." You didn't buy it then, and you don't now. But that's not today's argument.

Today's argument is over a new ORM system you are pushing. The rest of the group is on board, but Irene is their manager. Irene pushes back.

> "My crew doesn't have the time to learn a new system; they're already time crunched as it is."
>
> One of her employees pipes up, "Actually, I used it on my part of our last project, and it saved me a lot of time. Doing this group-wide could really help us with our time crunch problem.
>
> Irene's eyes narrow; her argument has been brushed aside. She thinks for a moment, and then says, "ORM causes a performance hit. We really can't justify across-the-board performance hits on our applications going forward."

Now you might be thinking, reading that story, that her reasons and opposition are full of holes. You'd be right. You'll also notice that once she was defeated with one argument, she pulled out a new one. Going further, that argument contradicted previous arguments that you had with her. It seems to indicate that the underlying points she uses at any time in these arguments are not things she believes. They are things that you'll buy enough to leave her alone and let her continue without taking up what you are pushing.

11.1 Underlying Causes

There are many potential causes for this behavior. There could be an interpersonal beef between you and the Irrational. They could be a zealot or fanboi for another technology or solution. The Irrational person could have one foot out the door of your organization. They could be a manager/developer struggling with the demands of both and unwilling to add one new thing to their plate.

By and large the reason doesn't matter, because the behavior is always the same. The Irrational jump in to an argument, they pretend to be another skeptic type, and if their arguments are defeated, they change arguments and even the skeptic type they seem to be. They have an underlying reason, but it will rarely if ever bubble up. If it does, expect an even tougher argument because it was important enough to them for them to hide it, dissemble to protect it, and argue to defend it.

11.2 Effective Countering Techniques

The techniques you throw at the Irrational aren't about changing their minds. You use tools and techniques to diffuse their opposition and deflect their arguments. But make no mistake, you are not trying to

Don't Feed the Crazies

There exists a group that I like to call the Crazies. The Crazies are a type of Irrational, in that you cannot oppose them with reason, but to be fair to the Irrational, the Irrational aren't that bad. By and large the Irrational have an actual issue, even if it isn't rational. The Irrational don't want to try a new language because they *personally* hate the syntax of the language you are proposing. The Crazies don't want to try a new language because they are convinced desktop publishing is the future of your industry and therefore are just treading water until that paper-based solution comes back into vogue.

The difference is academic I suppose. You don't deal with the Crazies any differently. But be aware that the Crazies are out there—and out there!

convince them; you are just trying to contain them. You can read about techniques that work with the Irrational in these chapters:

- Chapter 14, *Deliver Your Message*, on page 62
- Chapter 17, *Create Trust*, on page 79

11.3 Prognosis

If it isn't entirely clear from everything else I've said, *you cannot sell to the Irrational.* This is not to say that you cannot get them to participate, just that you won't do it by convincing them. You can bribe, punish, or compel them. In fact, the overall strategy I recommend is to ignore them and get management to mandate your technique.

Bottom line, you just can't convince them. So, don't spend a lot of time on them. It just leads to wasted effort.

Part III

Techniques

Chapter 12

Filling Your Toolbox

The next set of chapters contains countering techniques you can use on the skeptics to bring them around to your line of thinking. Most of the countering techniques work on more than one skeptic. Also, you will usually need to employ several of these together at the same time to effect change.

Some of the techniques are universal. They work in most cases, they're good to focus on for their own sake, they are completely within your power to do, and they don't depend on favorable circumstances. They are as follows:

- Gain Expertise
- Deliver Your Message
- Demonstrate Your Technique
- Create Trust

The rest of the techniques aren't completely dependent on you. In addition to your willingness to do them, they require favorable circumstances. For example, getting publicity can be really effective, but if you are working on code that includes your company's intellectual property, you may not be allowed to share any details about it. The trade-off is that when you have an opportunity to use them, and do so, they can be extremely effective, much more so than the ones listed earlier. These techniques are as follows:

- Propose Compromise
- Get Publicity
- Focus on Synergy
- Build a Bridge
- Create Something Compelling

To put these into practice, you create a list of those techniques that will work on your office of skeptics. Start with techniques you find on the first list, because you don't require anything but your own will with that list. Then look out for opportunities to use techniques from the second list.

Chapter 13

Gain Expertise

Don't try to push a tool or technique with which you are not familiar. If you conduct an afternoon of research and based on a few FAQs conclude that one particular tool is the one your organization needs, then you haven't done enough. The Uninformed are not going to inform themselves, the Cynic types aren't going to go easy on you, and the Burned are not going to magically forget the time your technique cost them a week of work. You're going to have to counter their opposition with knowledge of how your tool of choice works and what can go wrong with it.

Expertise in any particular tool or technique can take months, if not years, to acquire. I'm not suggesting that you must wait until you are an expert before you try to push a technique or tool. What I am saying is that you need to start down the path to becoming an expert. You need to know more than the marketing bullet points; you need to know when and where the tool shows its limitations. If you're a beginner, then you have to start becoming familiar with it; if you are familiar, go for fluent; if fluent, go for advanced. The point here is to travel toward expertise, not wait until you get there.

Push to Production

Ed suffers from déjà vu. The topic of conversation in the lunch meeting once again falls to deployment. Currently deployment is a nightmare of procedures, checklists, and manual tests that takes about half an hour, if it's done correctly. About one out of every three times it fails and causes the project's public Internet face to show a 500 error for half an hour until fixed. That's usually followed by a less than happy visit from management. It's unacceptable, but change comes slowly around here.

Ed has been advocating Ant to replace it. He's tried it on a few of his solo projects and figured out what it can and cannot do. In fact, he has a proof of concept of an Ant build, test, and deploy script for the group's project that works. It takes forty-five seconds, every time. He repeats his case to the group.

A co-worker, Bernard, interjects:

"I installed Ant, I ran it, but when I ran your script, it couldn't run the FTP tasks. Without them, why bother deploying?"

Ed knows that problem; he's had that problem. He's even documented how to fix that problem in the install directions.

"That's a common problem. You have to make sure the Jakarta ORO jar and commons-net jar are in your classpath. I have directions on how to do that. I can resend them to you."

Cynthia pipes up at this point:

"I've read a lot of blog posts that say that Ant is passé and Maven is better, especially since it doesn't use XML."

Ed has the answer to this as well:

"First, Maven uses XML too, so I don't know whether you're thinking of the correct tool. Second, Maven pushes a lot of conventions on the build process that don't line up with what we do. Sure, Maven might have more geek cred, but we could assimilate Ant into our particular processes a lot faster."

Herbert pipes up at this:

"I used it at my last job, but I just figured that you guys had a problem with it. Yeah, I'd switch to it."

Umberto is sold:

"I've never heard of Ant or Maven before. But Ed seems to have done his research, and frankly if I can just press a button to deploy in forty-five seconds, then I'm not going to quibble. Ed, can you maybe take some time and sit down with me to get started?"

Ed agrees. It's now three of them against the other two. One more public outage, and management might have to listen to Ed, Herbert, and Umberto, who for some reason never seem to be the cause of the outage.

13.1 Why Does It Work?

Picking up new technologies or tools is work. Even if it's going to save us time and effort down the road, sometimes that future promise is not enough to overcome the present pain. To do so, one needs either motivation or shortcuts. Presumably you had the motivation. Others aren't going to be as motivated as you. It's not their fault; they just don't have your drive. In absence of that drive, you need to be their shortcut.

Think back to your journey across the learning curve for the tool you are pushing. Remember when you knew it wasn't working because you didn't know the right syntax or forgot an option, but when you tried to Google how to fix it, you couldn't properly formulate the right query? At that point, you had two options: keep trying or give up. You kept trying, because you were motivated to do so. Make sure that when your co-workers reach that point, they have a third option: ask you.

A major part of the reason that Umberto was willing to give it a shot in the previous story was that Ed was willing to take the time to assist him. That ability and willingness to help is key in converting the Uninformed. Additionally, by defusing Bernard's issue and refuting Cynthia's claims quickly, knowledgeably, and respectfully, Ed was able to project the confidence to lead Herbert and educate Umberto.

13.2 How Do You Become an Expert?

Each tool or technique you push will have its own path to mastery. But that doesn't mean that there aren't a few common roads to travel down. Here are a few of the ways you can go about gaining expertise.

Research the Technique or Tool

Know the history of this tool or technique and its original purpose. Find out when, where, why, how, and by whom it was created. Know what competing technologies are available. Determine whether this tool is actually a proper fit for your organization. Know why it is and why another technology is not the right choice. In the previous story, it was obvious that Ed was ready for questions about Maven. By knowing where Ant fits in with its competitors, he was able to shut down the Cynthia's cynical voice; he even had the bonus of correcting a mistake she made.

A Journey, Not a Destination

Some people I've talked to about this get daunted and intimidated at this step. They don't feel that they could ever be the expert on something and therefore see this as out of their range. Nothing could be further from the truth.

Expertise, even for the very experienced, is a fluid thing. With the exception of abandoned technology, every tool, technique, or product that you are trying to push is changing every day. The producers are adding more features. The community is posting more commentary. Users discover bugs, and product teams fix them. Nothing is static. Therefore, even the most expert of experts is only as good as the last thing they read. There is no standing still as an expert. And yesterday's expert is tomorrow's dinosaur after as little as two years of not keeping up.

Since knowledge becomes obsolete so quickly, the difference between true experts and dinosaurs is constant learning and updating. Since that's what you have to do to become an expert anyway, there is not that big a difference between a true expert and one who is striving to be an expert. They're just a little further along the journey.

So, keep that in mind for both encouragement and humility. An expert that isn't growing isn't an expert, because growing is what makes you an expert.

Use It

Every tool or technique has its limit. Everyone has a task or a category of tasks it does not handle well. The designers, developers, or marketing people usually don't tell you about them on the landing page of their websites. You might be able to figure out the pain points via the FAQ or support forums, but you're never really going to know its pain points with your environment unless you bite the bullet and give it a try.

Ed was able to respond so quickly to Bernard's issue because he had gone through the steps and installed Ant himself. He knew that the JAR files needed for the FTP tasks aren't included in the standard distribution of the product. He knew it would trip up his co-workers, so he even documented it specifically. That sort of preparation is possible only when you've already walked the path.

Be careful when you choose a project on which to learn. If you are using a very pervasive technique or tool, don't choose a project that you share with other people who may be resistant. Otherwise, you might expose them before you are ready to defend it and tarnish the tool in their eyes.

In the example story, Ant was safe when it was used to automate a manual task. It isn't a big risk because it was only replicating an existing procedure and only Ed had to use it. Even using source control on your local version of the code might not be a big problem. However, rewriting your entire application to follow particular design patterns that aren't understood by the rest of your team probably won't go over big.

Seek Out Existing Experts

Seek out and form relationships with people who have expertise with your tool or technique already. If your town has a local users group for it, participate in the users group. If not, seek out other experts online in forums and blogs.

See whether you can learn the common newbie mistakes from existing experts. Find out the horror stories they have for the wrong use of the tool or technique. Learn anything you can to make it easier to handle your new users.

Teach It

This is one of the later stages of gaining expertise. Nothing will make you quite as knowledgeable about a subject as teaching it. Often with tools or techniques, we only put enough thought into it to make it work for ourselves. This is especially true of things we've abstracted—we don't go that extra mile and really understand what it does. When you teach it, you don't have the luxury of just describing encapsulated steps; you have to explain what happens in those steps.

The tough part will be finding someone willing to learn. Your best bet here is to find someone completely outside the group you are targeting to adopt the technique. If you can't find someone external, then try an uninformed member of your team who doesn't usually become cynical, burned, or irrational.

13.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Uninformed

The Uninformed (Chapter 5, *The Uninformed*, on page 29) are defined by their lack of knowledge about a tool or topic. By being expert, you give yourself large amounts of that knowledge they lack. The more you know, the more you can pass on.

The second point is very important. I assume you will share your knowledge. If you don't, you can't really be successful with the Uninformed. You have to make yourself available to nudge, tutor, and encourage your charges. But that effort will be rewarded—the Uninformed are the easiest to move, so every minute you spend on them is equivalent to two or three times that with another group.

If that benefit wasn't enough, it's worth it to point out again that teaching is one of the best ways to learn. Having to explain how something works requires more thought and understanding than just using it. Therefore, teaching makes you more of an expert, which makes you better able to teach, which means you can teach more, which means you can become more of an expert.... In computing that's an infinite loop, but in psychology that's a positive feedback loop.

The Herd

The Herd (Chapter 6, *The Herd*, on page 31) requires leadership. It's hard to lead when you're not sure where you are going. Knowing what your tool can do, and more importantly what it will mean for your organization, is critical to leading the herd.

Your ability to lead is not the only consideration. The Herd's willingness to follow you is another. The confidence that comes from being comfortable with your knowledge will go a long way to inspiring them to follow you.

The Cynic

The Cynic (Chapter 7, *The Cynic*, on page 34) is contrary out of reflex. They've been promised great things in the past by other pieces of technology and have always been disappointed. They're that kid in college who always asked your professors annoying questions to show how smart they were. Consequently, they love to come up with nitpicky "gotcha" scenarios so that they shoot down progress before it can disappoint them. Answering with a confident, knowledgeable response goes a long way to shutting them down.

The other important fact about the Cynic is that they tend to be broad but not deep. This isn't an unfair generalization. When you don't think anything new can be of use, you don't give anything a try. Therefore, they know superficial details about a lot of things but just enough to shoot them down. This allows them to throw competing technologies and potential horror stories at you. Being able to go deeper than the Cynics will allow you shut them down.

The Burned

The key issue with the Burned (Chapter 8, *The Burned*, on page 38) is that they used the technique and it failed for them. Being an expert and having a good amount of experience with a technique will put you in a better place to understand what exactly happened to the Burned. If you really understand what happened, you can better respond to their issues and perhaps more tactfully tell them where they went wrong.

This tactful part of this equation is an important one. People will dig their heals in if they feel they are being attacked or unfairly criticized. You don't get to define unfair criticism; they'll do that for themselves. So, remember, be an expert, not a know-it-all.

13.4 Pitfalls

There are two major pitfalls when using this technique:

- While you're learning, you force others to use it.
- In selling, you patronize or bully others.

Forcing Others

When most people begin with a technology, they go for easy demos or test code. Most tools work perfectly fine in these cases because they're not too complicated. We know this, so we start to want to work on a real use case with the tool or technique. At this point, you'll be tempted to use it in your working code. But before you do so, consider whether by doing so you force others to have to switch to it. Depending on the technique, you may be forcing it on people when you aren't prepared to show it to them.

In the story, Ed just took an existing procedure and ported it to Ant. Nothing was changed for everyone else. On the opposite extreme is switching to a framework with an MVC architecture when you weren't

using it before. It's hard to localize that change to just your part of a project.

Those are the obvious cases; the real challenges exist in between. Consider unit testing. At first glance it would seem to not affect other people, because you write unit tests to work with your own code. However, when you start writing unit tests, you start to change the way you write the code you test. Granted, it generally forces you to write better code, but it might be incompatible with older code, or it's possible that people will be uncomfortable with the new style because they won't see the point.

For this reason, I recommend you do your learning on your side projects. You never know what effects your learning will have on others. You're trying to convert them, not turn them into the Burned.

Patronizing or Bullying

The difference between kindly expert and patronizing know-it-all can be thin. It comes down to whether you listen to others. A know-it-all listens only to themselves—as soon as they have an answer, they spout it out regardless of whether it applies. On the other hand, an expert listens to everyone, and from that listening develops answers.

In some cases, expertise can also be used to bully people. You can start to take the attitude that I am an expert, and therefore I am always right. It's the wrong attitude for several reasons: it's elitist, annoying, and ineffective. On top of all that, it's flawed in analyzing causation: you get the label of expert because you are often right, not the other way around.

13.5 Wrapping Up

Gain Expertise by itself is very effective in converting a few of the most easily converted types—*low-hanging fruit*. Converting them is an important step in the overall strategy of converting skeptics. But, it usually needs to be combined with other techniques to be effective in a broader group.

There are also some practical benefits to be gained here, independent of your efforts to promote your tool or technique. Gain Expertise can be done either on your own time or as part of your day job. It doesn't require the input of others. It can add bullet points to your resume.

In short, Gain Expertise is a technique that has a lot of upside and relatively little risk. It should be the first technique you try. I would posit that in most cases you should seek to be an expert in any tool you are promoting and that no array of influence techniques is complete without it.

Putting It Into Practice

Here are a few suggestions to try to increase your expertise going forward:

- Read the entire manual for the product you are pushing.
- Jump onto the public forums for the tool your are trying push. See whether you can answer questions on it.
- Start blogging about your tool or technology.

Chapter 14

Deliver Your Message

So, you've picked a technology that's better than what the rest of your co-workers are using. It's better, so that should be enough, right? Unfortunately, being better is seldom good enough. If a program never crashes in the woods, does it make a sound? No, weird metaphor aside, if you don't tell people about the tool, they'll never adopt it.

You'll have to talk about the tools you are pushing. While you're talking about them, you have to do two things:

- Not turn off your audience
- Actually turn them on

Yes, not turning them off comes first. It doesn't matter how well you do at turning them on; if you turn them off, they're gone. You won't be able to sell these tools to them without a lot more attention.

Version Control Conundrum

John had discovered Git,[1] and he was in love. He loved every single feature of it. He even loved obscure little features that he never used. He simply loved it, perhaps even a little bit too much.

Patrick was a longtime CVS[2] user who was thinking about switching to a new version control system, because he was tired of the limitations of CVS. He had been researching, and SVN was the system that was coming out on top. Not too different, but much better. The SVN motto of CVS done right hit home for him. He was ready to switch until he ran into John.

1. A distributed version control system. For more information, see *Pragmatic Version Control Using Git* [Swi08].
2. A centralized version control system. For more information, see *Pragmatic Version Control Using CVS* [TH03].

> "How can it be any good when CVS is your starting point?" John exclaimed.
>
> Patrick became defensive, "Well, it works with the tools that I use."
>
> "That's crap! We both use Eclipse. Git works just as well in it. And it performs better, and it's distributed, which means...," droned John.
>
> By this point, Patrick had stopped really listening and instead just nodded until John left him alone. Patrick gave up on upgrading source control altogether.
>
> A few months later someone else introduced him to Git again. This time, they extolled the virtues of Git without beating up on SVN. Patrick gave it a try. He found he liked it. If only John hadn't been such zealot about it....

In the previous story, John took someone who was excited about making a change and drained the excitement out of him. What was worse, John didn't just lose a chance to make Patrick a Git fan, he stopped Patrick from advancing at all. The moral here is: don't be John. Don't drive people away; bring them in. Easy enough said, let's talk about how you can do it.

14.1 Why Does It Work?

People, even technical people, are just that: people. As much as we like to pride ourselves on being able to judge information based on facts, sometimes we go for the flashy shiny things instead of the "right" thing. The technological world is full of superior technologies that never caught on because of this very reason: Betamax, Smalltalk, FireWire.

That's because people don't just make decisions intellectually. We use our emotions. In fact, we usually use them more than our brains.

So, packaging matters; message matters. Don't worry, we don't have to tell the marketing department that we know it's true. It will just be our secret.

14.2 Mastering Delivery

Many people think that talking, connecting with, and influencing people are all something you can either do or not do. Either you're a people person or you're not. That's just not true. Yes, some people are better at this naturally; they have a talent for it. But at the end of the day,

connecting with people is a skill—a skill that can be learned and even mastered.

Be a Person, Not a Computer

Developers spend a lot of time gazing into monitors. "And when you gaze into the abyss, the abyss gazes into you." Like Nietzsche is suggesting, you've picked up some traits from your time living with the machines. As developers, we start to see the world in binary terms: if what I am recommending is right, then I'm right, and that is the only thing other people should judge my solution on. Sadly, that is seldom the case. People are emotional creatures, even other developers. Being told their choices are no good doesn't sit well, even if you are right.

So, don't tell people their current choices are "wrong." Don't talk to them like they are misguided. In fact, don't address their current state at all if you don't have to do so. Talk about how your tool is *effective* or *productive*.

Be Passionate, Don't Be Zealous

The difference between passion and zeal is subtle, but I think it comes down to this:

- Passion is when you love your subject and want everyone else to use it because it will make their work better, easier, faster, or more enjoyable. You can admit that some situations will call for other solutions but know that your tool could work in those situations.
- Zeal is when you love your subject and think everyone must use it simply because it is absolutely better. You can't conceive of a situation where you shouldn't use it, and if confronted with the idea of such a situation, you'd decry the situation itself as somehow wrong.

It's a blurry line to define. But the long and short of it is if what you are saying is worded to make sure your co-workers' work is improved, then you're probably being passionate. If what you are saying is worded to gain acceptance for the *right way* of developing, you're probably being zealous.

The takeaway here is don't talk about your tools in terms of good or bad, right or wrong, righteous or evil. Don't speak in absolutes or suggest that your tool should be used in every case, especially when alternatives might fit better.

Suggest, Don't Declare

Many times our reaction to hearing that people are not using the technology we are advancing is to browbeat them:

- Why aren't you using Git?
- SVN? That's just plain wrong.
- You should just move to Git!

Now, when we do this, we don't mean to drive people away. We're just letting our enthusiasm get away from us. But it doesn't matter. This type of speech will put your audience on the defensive. The better way to encourage people is by suggestion, not declaration:

- Have you considered Git?
- SVN? I've had nothing but trouble with SVN.
- I've had a lot of success with Git.

Listen More Than You Speak

Telling people to do things is seldom effective. People get defensive. Even the Herd, who want leadership, don't want orders. The better thing to do is ask them questions:

- Why did you choose that?
- What problems are you trying to solve with that?
- How does fit into your workflow?

Then here's the hard part—after you ask those questions, listen to what they say. Understand where they are coming from. Know their issues. They'll respond better to what you have to say, just by virtue of the fact that they will feel listened to. Additionally, any arguments you have to give will be in the context of their problems and will be much more effective.

How can you be sure that you're listening? Paraphrase their answer, and repeat it back to your audience. Ask, "Did I understand what you were saying?" You might feel a little ham-handed or awkward when you do it, but get over it. Your audience will appreciate it, and the more you do it, the more comfortable it will become.

Remain Positive

Like a political campaign, going negative is a sign of weakness. Saying the competition is no good is a bad argument. Saying your tool is better than the competition is an improvement. Saying your solution will make your fellow developer's day more productive without qualifying it is the best way to sell it.

Now, this doesn't mean you can't ever make comparisons or honestly express your feelings about competition, but like Patrick Swayze in *Roadhouse*, be nice until it's time to not be nice.

14.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Uninformed

The Uninformed (Chapter 5, *The Uninformed*, on page 29) is the easy group to alter. However, they are just as likely to become a skeptic as they are to be converted. You need every advantage you can with them. The last thing you need is to put them off by coming on too strong.

The Cynic

The Cynic (Chapter 7, *The Cynic*, on page 34) isn't just willing to fight with you; they want to fight with you. You cannot completely prevent them from disagreeing, but you can give them fewer footholds into an argument. Delivery is the way to take away those footholds and perhaps get them to listen to you.

The Irrational

You can't really affect the Irrational (Chapter 11, *The Irrational*, on page 47)—that's their thing. So, practicing delivery isn't going to help sway them. However, it does ensure that your message is as reasonable and rational as it can be. That way, when the Irrational respond, they are more likely to identify themselves by acting unreasonable or irrational in the face of your message.

14.4 Pitfalls

You would think that there would be no downside to having good delivery. Mostly there isn't. However, while you're getting started with the

technique, you might come off a little fake or, even worse, like a marketing person. Just watch it, stay genuine, and just make sure you don't use any words like *paradigm*.

14.5 Wrapping Up

The important lesson to take away from this is, *What you say is less important than how you say it.* The other important takeaway is, *What you say matters less than what they hear.* Going forward, think about the way you've been talking to people about this, and try to improve your message.

Putting It into Practice

Here are a few suggestions to try to improve your delivery:

- Practice your pitch, either with a willing partner or even alone—the important thing is to do it out loud.
- Record those practices, and listen to them. If you weren't listening to yourself, would you be convinced?
- If you're writing something to pitch your ideas, pause for an hour before sending it out, and reread before you hit Send.

Chapter 15

Demonstrate Your Technique

One of the strongest memories I have from grade school is a class story-telling. I remember the Sister who taught the class yelling at us, "Show, don't tell." Years later in my early twenties, I did professional improv comedy. The director repeatedly yelled at me, "Show, don't tell." Today, I'm yelling at you, "Show, don't tell."

What does that mean? Let's take two potential first sentences to a story:

- John walked down the street. John was sad.
- John shuffled aimlessly down the street, his eyes clouded in tears, thinking of Lisa, his lost love.

Both convey the same meaning, but one tells you what is going on, and one shows you. Which one is more likely to grab you? Which conveys more story?

Although not exactly the same in the realm of selling professional development, the idea is similar. Telling someone about a tool—its abilities, its benefits, its uses—is never as effective as demonstrating it.

TV Marathon Coding

Ed had written a code-generating machine. Strike that—he built a code-generating dynamo. Give it a database, and it used database introspection, ORM, and company-branded UI components to build a killer application in milliseconds. It was flexible, and it allowed for flexible tweaking after the code had been generated so that the model could be manipulated without destroying changes to generated code. In short, it was the master system Ed had been dreaming about.

His boss, Bob, was less idealistic about it. Bob understood it did stuff, and Ed said it did good stuff, but his analysis didn't go deeper than that.

Ed had a tough time convincing Bob to let him work on it during office hours.

One Friday, Bob and Ed were talking about a task-tracking application for the office. Ed thinks over the problem and says his code generator can knock it out of the ballpark. Bob drops the idea because he can't justify the hours it will take to build.

Ed goes home, and luckily enough, there is a marathon of his favorite police and district attorney drama on this weekend. He plops down on the couch with a remote and laptop and gets to work. He uses his generator to create the application and spends a few hours perfecting the UI and business model.

Monday rolls around, and Ed shows Bob the application.

Bob is floored, and asks, "How did you build this application in just a weekend?"

"With that code generator you are always making fun of," Ed answers.

Bob pauses for a second and then requests, "Take me through that again, starting at the beginning."

The previous story illustrates that no matter how many times Ed told Bob what his tool could do, Bob couldn't understand it. It was only by seeing the tool in practice and seeing the scope of the problem and the speed of the solution that Bob could wrap his mind around it. And now that he's seen it, Bob can't forget the impact of seeing his idea become an application in record time.

15.1 Why Does It Work?

People believe what they are shown more than what they are told. It's that simple. The phrase doubting Thomas is a reflection of that fact. Maybe people lack trust. Perhaps they tune out when they hear lists of features, as opposed to seeing dramatic demos. It's even possible that self-interest leads people to be more impressed when they see problems they themselves have as opposed to vague hypothetical solutions.

15.2 Demonstration Opportunities

Demonstration requires opportunities to demonstrate. You can either prepare yourself as much as possible to demonstrate if the opportunity arises or force an opportunity to arise. Both have their advantages. Organic opportunities are those that come up randomly. Someone asks

about something you have expertise in, or you're hacking at something and they look over your shoulder. You're not asking them to listen; they are asking to be told. When opportunities like this crop up, people are more receptive, but you have little control over the particulars. Forced opportunities are one that you create. Maybe you volunteer to do a brown-bag session at lunchtime or have specifically scheduled time to talk about your tool. When you create an opportunity in this manner, you control the situation completely, but people are more likely to be skeptical. In the end, you can do either or both. It will depend on you, your preparation, your ability to perceive organic opportunities, and your patience to wait for them.

Waiting for Opportunities

In the previous story, Ed waited for an opportunity to arise. He was prepared, and when it presented itself, he seized the opportunity. Some tools and techniques are more likely to be sold in an organic opportunity.

Source control is a good example of this. It's hard to create opportunities for showing source control's ability to recover from failure. Well, it's not hard to create them, but if you do, you may find yourself looking for a new job. So, you have to wait for that perfect moment where someone has screwed something up and you have the ability to restore with just a command line call or two. It's a long wait, but when you do that, you have a captive audience.

Creating Opportunities

On the other hand, even the techniques that benefit the most from organic opportunities can be shown off as a forced demonstration. Source control isn't just about catastrophes; it's also about keeping track of several active releases. Showing people that in action, although not as dramatic as recovering from a failure, can still grab them.

Creating those opportunities is pretty straightforward: call people together, and show them stuff. The mechanics are up to you. Maybe it's a brown-bag session at work. Perhaps you organize and schedule a quarterly show-and-tell. Most organizations have ways of doing this.

Encouraging Opportunities

There is a hybrid method of both waiting for and creating opportunities. I call this encouraging opportunities. You set out bait, get someone to nibble, and then go into a prepared demonstration.

I am by trade a software evangelist. I have lots of code demonstrations at the ready for the tools I evangelize. However, in most cases, just launching into a demonstration will fall on deaf audiences. But if I can get someone to *ask* me to demonstrate my tools, they'll be receptive. How do I do this? There are a number of ways, but one that jumps to mind occurred while I was writing this book. I have logo stickers for the software products I promote on my laptop. Someone in a coffee shop I was in saw one and asked me about it. I copped to being an evangelist, I told them about my tools, and they wanted to know more. An organic opportunity arose. Because it wasn't forced, the person was receptive, but I was prepared, so I was at my most persuasive.

Code Reviews

Finally, another great way of creating opportunity to demonstrate languages, frameworks, and coding techniques is the code review. You can demonstrate how easily you were able to accomplish what you did with your tool. Code reviews have other benefits as well. We will talk about them in other chapters. For now, they can be a source of demonstration for other techniques. That's assuming they are not one of the tools you have to sell as a professional development technique.

15.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Uninformed

The Uninformed (see Chapter 5, *The Uninformed*, on page 29) are defined by their lack of knowledge. By demonstrating the technique, you bypass feature sets and go directly to what this tool can accomplish for them. This quick dose of information usually knocks them out of the Uninformed with one shot.

The Cynic

The Cynic (see Chapter 7, *The Cynic*, on page 34) wants to argue. You can argue with claims, features, and promises, but it's much harder to

argue with demonstrated results without sounding petty. By blocking those arguments, you get a leg up on the Cynic.

The Time Crunched

This method helps with the Time Crunched (see Chapter 9, *The Time Crunched*, on page 41) because by and large this technique packs a lot of bang for your buck. You can quickly show the full benefits of your technique. *Quickly* is the key word here. By doing it faster, you keep their attention long enough for them to see the potential benefits.

The Irrational

The Irrational (see Chapter 11, *The Irrational*, on page 47) are looking for any opportunity to block your efforts. However, when someone successfully demonstrates something, opposing it for no reason looks, well, irrational. It's this danger of exposure that can keep them at bay. Although it won't convert them, it will block their efforts to hinder you.

15.4 Pitfalls

There's one glaring risk when relying on demonstration: Murphy's law. When you are doing a live demonstration of technology, that technology will fail. Do enough demos, and you will wipe out in front of an audience and completely fail. There are ways of mitigating this. You can have prebaked versions of code that you will be writing in front of an audience. You can do a screen capture of the entire process working correctly. Eventually, though, you'll suffer through a failure. It's bad when it happens because as persuasive as a demo can be when it works, it's just as persuasive when it doesn't—it's just that you're persuading people that it doesn't work.

The best thing you can do when a demonstration fails and your contingencies don't work is stop. Go back to explaining, and don't let your audience see you get rattled. You might not make any gains by doing that, but you stop the hemorrhaging, and when not losing more is the best you can do, you take it.

15.5 Wrapping Up

This is one of the more broadly effective techniques in our arsenal. It works on many skeptics and does so in a relatively short period of time.

If you can be prepared to demonstrate at either an organic or inorganic opportunity, you'll be highly effective in bringing people to your side.

Putting It into Practice

Use these ideas to get in shape with demonstration:

- Write a demonstration of your technique, and deliver it to a co-worker live.
- Record a demonstration of your tool, and share it on your blog.

Chapter 16

Propose Compromise

As organizations age, they grow rules. Rules often come up in response to incidents—the policies prevent those incidents from occurring again. The downside of those policies is that they can end up constraining more than they protect. The other major downside of policies is that they tend to be enacted and then followed without review. Very few organizations ever ask whether they need their rules anymore, but in many cases the technologies have outgrown the need for particular rules. So, groups are left mindlessly following outdated rules that are no longer needed to prevent threats that have already been closed. It's a huge morale killer, but it's a big opportunity to sell change.

Tortured Procedures

Jeff's SQL administrators have a rule. *All database activity has to be done in stored procedures.* Much to Jeff's chagrin, there are absolutely no exceptions.

A few years back the company's public web presence was hacked using a technique called *SQL injection*. If you're not familiar with it, it basically means appending SQL commands to a form post in the hopes that a developer didn't properly process input from users. It's the equivalent of writing "and a million dollars" to the end of a bank check. (To prevent that, we all write that silly line on our checks.)

There are a couple of ways to prevent this. Training developers to keep security in mind is one way. The way Jeff's company went about solving it was forcing all code to go through stored procedures, ensuring that the DBAs looked over all of it. It also forced the use of parameterized communication with the database, which effectively prevents SQL injection.

Most of the other developers on Jeff's team hated having to write stored procedures. Most of them hated have to spend time writing any rote SQL. They especially hated having to go through the DBAs to change a simple select statement, but every time Jeff recommended a switch to some sort of ORM solution, his teammates shot him down. Some of the sentiment against the change was typical resistance. However, a big piece of the resistance was that the DBAs wouldn't allow a solution that generated SQL within the application.

On the other hand, the DBAs were constantly complaining about how busy they were. You'd be pretty busy too if you had to look at every single SQL call being made against a corporate database.

Jeff's team worked mostly with Java, so Hibernate was the ORM solution Jeff knew he would want to use. Jeff did some shallow digging into Hibernate's internals and figured out that Hibernate uses parameterized queries for most operations.

Jeff took this information to his team and to the DBAs. The team hopped on board because it freed them from having to do stored procedures. The DBA team agreed to give it a try but required a trial run with a SQL Profiler running.

A few months later, all new projects were using Hibernate. The DBAs had a lot less CRUD SQL to look at. The developers were happy to no longer be forced to use stored procedures. Everyone was happier.

In the previous story, Jeff proposed a compromise between the DBAs and the developers. He created a win-win solution that got rid of a outdated and restricted rule, and they adopted his tool set to boot. Not bad for day's work.

16.1 Why Does It Work?

You're trading a major source of pain for a new tool or technique. It's a good bet that if you ask someone to choose between something that they already hate and something they don't know much about, they'll at least give a listen to the thing they don't know much about. You're using people's hatred of a rule to be the driver of a change. Granted, we'd rather positive sentiment drive adoption, but you have to play the hand you get dealt.

16.2 Discovering Compromise

Using compromise to sell your tool or technique requires you to find a rule that is ripe for compromise. Basically this will be a rule that everyone hates, gripes about, or silently seethes about. Once you figure out the rule to go after, you have to build your case for swapping your tool with the rule.

Finding Ripe Rules

Ripe rules are often pretty obvious. Do all the developers make fun of a certain policy? Do they complain every time they run into the rule? Odds are you have a rule that can be compromised.

Other rules aren't as obvious. Perhaps the incident that precipitated the rule was so horrific that your band of developers are cowed into submission. Perhaps the rule doesn't appear that painful because people don't know any better.

Here are some rules that are ripe for compromise:

- *Absolute rules:* Rules without exceptions tend to cause pain.
- *Security rules:* People often overreact to security incidents with blanket rules that don't make people any safer, just more constrained.
- *Industry best practices:* Sometimes people apply best practices without considering whether they apply or are needed in their organization.
- *External imposed development rules:* If someone other than the developer team is dictating how applications are built, or communicate, odds are those rules can go.
- *Rules without a because:* Rules should have a clear reasons behind them. Developers should be able to say, "We must do X because Y." If no one remembers the Y, that rule needs to be questioned.

Matching Technology to Rules

Once we have the rule we want to go after, we have to line it up to the technology we want to use. In the previous case, it's pretty straightforward. The stored procedure rule was designed to prevent SQL injection, it caused the developers and DBAs a lot of work, and Hibernate reduced the threat from SQL injection and reduced the amount of work. This

one is pretty easy because the technologies are within the same area of concern: database interaction.

Let's take a slightly less obvious example. Suppose the boss has a rule that she has to test all web application changes before they can be pushed out. That's going to cause problems. It's not like the boss has a tremendous amount of unscheduled time. It reduces the web application team's speed to update, and let's not even consider what to do when the boss is out of the office. In short, this rule is ripe for change. So, the developers suggest Selenium, an HTML UI testing framework. They can write and automate tests. They can let the boss write tests if she wants, but they get to iterate faster, not to mention that the tests are better than the boss just poking around. It's not obvious because user acceptance testing and automated testing are usually two different things, but in this case one substitutes very well for the other.

16.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Time Crunched

Rules constrain. Time constrains. The Time Crunched (Chapter 9, *The Time Crunched*, on page 41) are stressed from too many constraints. Lifting rule-based constraints tends to free up time. That time comes from both the time to implement the rules and the time it takes to verify they are in compliance. Giving these folks back some time will score you some points with this group.

The Boss

Let's face it, when developers impose rules, they're called "best practices," and when the Boss (Chapter 10, *The Boss*, on page 44) does it, they are called "policies." That difference in language is very telling. Bosses are the drivers of rules. They oversee a lot of people, and one-size-fits-all rules are easier to keep track of. Trading them something for rescinding a rule is the only way to get some of them to give them up, because in these cases, rules are a solution to *their* problems. Only by offering more solutions to their problems can you get them to part with their rules.

16.4 Pitfalls

The big challenge here is that not every organization has rules that can be perfectly replaced with a technology change. Although my previous story was based on an actual occurrence, I haven't seen too many of them. These are rare but worth it when they come up.

The other challenge is creating a group of like-minded people with enough power to overturn the rules. Typically the further away the policy enforcers are from your team, the harder the change is to make. If rules are self-imposed by the development group, then it's a snap. If rules are dictated by human resources, good luck. In between there is a continuum of difficulty levels that you'll have to judge and counter to be effective.

16.5 Wrapping Up

This technique isn't as broadly applicable as others, but when it's appropriate, it can be very effective. Basically, don't count on being able to do it, but if the opportunity arises, go for it.

Putting It into Practice

- Poll your development team on the worst or most painful rules or coding standards they have to deal with.
- Periodically ask "Do we still need this rule?" for every rule your team has, even if the answer is obviously "yes."

Chapter 17

Create Trust

Having the correct facts is a huge part of convincing people to come over to your way of thinking. However, a large, often overlooked part of the story is trustworthiness. If people don't trust you, you have an uphill battle justifying even the most obvious of tools and techniques.

Trust is a little harder than the other techniques in that there is no formula or checklist for it. You can take a class to Gain Expertise, but no class will get you to be trusted. In fact, gaining trust is often more about what negative things you don't do than what you do. But have no fear, although there are no shortcuts, that doesn't mean there aren't ways to get people to trust you.

FUD Factor

Shailaja's company was in the market to invest in a new database installation. Because of the high cost, she was looking at alternatives to their current technology choices. After careful consideration, she decided to change the entire company system over to MySQL. In her case, MySQL was cheaper, was more supportable, and had more public information available.

Shailaja came up against a lot of opposition. Some of it was irrational, people clinging on to old technology, but some of it was reasonable: change has cost; in this case, they may have been too high. Also, MySQL had just had just passed to a new owner, and their future wasn't written in stone, even if all signs were good at the moment.

The discussion got heated, and at one point someone blurted out, "I've heard that the company that owns our old system is going to discontinue it next year."

FUD

FUD stands for Fear, Uncertainty, and Doubt. Though the phrase was coined in the mid-1970s, the concept has been around since the first caveman traded a rock to another one "in case the mastodons come back." More recently, it's been marketers, public relations flacks, and sales guys who use this on you. Basically, the idea is to tell you something that will make you afraid of a rival's tool, enough so that you invest with the FUDer.

At a smaller level, this happens in the workplace a lot. Developers with experience with proprietary tools spread rumors about crazy license implications of open source tools. Open source adherents spread horror stories of hidden code in proprietary toolkits.

It's ultimately self-defeating. At best it can win people some sort of short-term gains, but in the long term, it is a road to nowhere. Eventually people wise up to be bullied repeatedly, and some people speak out. This spread of information inoculates the rest, and the technique becomes ineffective.

People were really swayed by this. There was one problem with it. Shailaja knew it wasn't true. Flamebaiters had made that up as FUD in a forum somewhere, and the company had been fighting it ever since.

She was in a quandary. She liked the effect that the FUD had on her co-workers, but it was factually wrong. She knew that if people found out the real story behind it, they would be pissed at trying to be bullied. But the odds of them finding out were slim—her co-workers weren't forum people.

In the end, Shailaja corrected the FUD. MySQL was adopted anyway. Her co-workers went with her recommendation partially because she had always been aboveboard with them.

17.1 Why Does It Work?

Create Trust works because people do not like to be manipulated. Tougher than that is that people don't want to even feel like they're being manipulated. This is why the used-car salesman has such a bad common stereotype in our culture. They're seen as deceptive and manipulative (even when they often aren't).

Oftentimes when changing tools and technologies, you reach a point in the decision process where all of the empirical stuff has been revealed, choices have been weighed, and you still don't have a complete victory for one tool or another. When an organization reaches that point, often they need a tiebreaker to make that decision. That tiebreaker is going to come from something not quite so rational, not quite so empirical. It is going to come from something personal like trust. Making sure that you are trusted clearly has a lot of value when you get to that stage of the decision-making process.

17.2 Developing Trust

As I said earlier, there are no magic bullets for creating trust. There are things you can do to develop trust over the long term. I'd make the argument that you should be doing these things anyway, but to each their own.

Don't Lie by Commission

Lies by commission are pretty obvious because they're what we think of normally when we think about lies. Saying something like "MYSQL isn't a relational database because it doesn't handle foreign keys" is a lie of commission. This is pretty straightforward. It's possible that if caught, you can claim it was a mistake. But go to that play too often, and people will catch on to you.

I know our kindergarten teachers told us not to lie when we were young, but I'm telling you not to do this now because we can still find excuses to do it, especially when there is a kernel of truth to it like in the MYSQL statement I just made. By default MySQL tables don't do foreign keys, but if you set the table type to InnoDB, it can. That's where people get into trouble. Subtle issues like this make it easy to justify lying. Just don't do it.

Don't Lie by Omission

Lies of omission are a little bit trickier. These are cases where it's not necessarily that you didn't say something that was untrue. It's where you failed to say something that was true and necessary to the conversation. So, saying something like "MYSQL table names are case sensitive" is a lie of omission because it's leaving out the rejoinder "on certain operating systems." If you were talking about issues such as

converting old databases to MYSQL, this becomes very significant to the discussion.

This isn't as destructive as a lie of commission because people give the benefit of the doubt and assume it is possible that this was a mistake or oversight. This is what makes it so attractive. There's high benefit and little risk. But the more you do it, the more likely people will come to one of two conclusions about you:

- You're leaving out things intentionally and therefore untrustworthy.
- You're mistaken a lot and therefore untrustworthy.

Neither outcome is good for your credibility.

Never, Ever, Ever Resort to FUD

FUD is by its very nature destructive in that you're motivating people through fear. Sooner or later people will get this. When they do, there are personal repercussions. Getting caught spreading FUD, especially if it is untrue, will ruin your reputation for months or years. Some people will simply never trust you again.

Even if they don't necessarily get that you are manipulating them, they might think that you are "the boy who cries wolf." If you predict doom and gloom and massive catastrophe enough and it doesn't happen, people start to realize that they shouldn't listen to things you say. That is the opposite of being trustworthy.

This does beg a question, though. "How can I speak about competing technologies if I can't spread FUD? At some point, don't I have to say that the other tool or technique is worse?"

I would say, always speak from your own tool first. The other tool isn't worse; yours is better. When asked to address a competing tool's shortcomings, you do have to say something, though. So, make sure that you speak plainly, unemotionally, and unexaggerated. Also, if you can, have industry sources for that criticism at the ready—that would help.

Admit Mistakes

Unlikely as it may seem, you will make mistakes from time to time. You may even be completely wrong. You may be tempted to ignore it and take attention away from it. This temptation will be even greater when you are in the middle of trying to get something adopted.

Believe it or not, being wrong might be the best thing to happen to you. It gives you the opportunity to inform people that you are wrong. Yep, you read that right. See, most people aren't used to people owning up to their errors. Even if they are, most people have to be confronted to own their mistakes. But the fact that you willingly admit mistakes allows people to trust you. Because you tell it like it is, even when it is bad for you.

All that being said, if your mistake is about the tool or technique, you need to reevaluate. Admitting you're wrong about something but continuing to insist that people do it is pure foolishness.

17.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Burned

The Burned (Chapter 8, *The Burned*, on page 38) by nature have both experience with what you are talking about and trouble believing what you are saying by virtue of the fact that it disagrees with their experience. Any perception of dishonestly will immediately cause them to clam up. Don't give them any excuses.

The Cynic

The Cynic (Chapter 7, *The Cynic*, on page 34) doubts everything people say anyway. Giving them a reason to doubt you will only redouble their efforts to expose you for the fraud or fool they already think you to be. Don't compound this problem by actually being a fraud.

The Irrational

The Irrational (Chapter 11, *The Irrational*, on page 47) are looking for any excuse to shut down the change you are selling. You being a liar is a pretty good one. They will ride any mistakes in honesty you make for the rest of your working relationship. That's not grief you want.

17.4 Pitfalls

Telling the truth has few pitfalls. But there is one big one: in your effort to be seen as trustworthy, you point out and emphasize the flaws in your solution. You want to acknowledge that your tools have flaws,

downsides, or compromises, but advertising them is another matter altogether.

Sadly, there is little to be done about this. Because nothing is perfect, you will occasionally be put into the position of saying something bad about the tool or technique you are advocating. Own up to it, acknowledge the weakness, and spell out why you don't think that the weakness outweighs all your solution's strengths.

17.5 Wrapping Up

It seems pretty silly to have to encourage honesty. However, the temptations to lie are pretty big, especially in the short term. In the long term, though, the costs are too high. Tell the truth, ride out short-term issues, and play the long game.

Putting It Into Practice

Here are a few suggestions to try to improve your trustworthiness:

- Create in your mind a scenario where a competing technology makes more sense than yours.
- Investigate the weaknesses of your own solution. Make sure you know where it falls down.

Chapter 18

Get Publicity

We all know that if a tree falls in the woods and nobody is there, it doesn't matter if it makes a sound, because nobody hears it. Sure, getting publicity is about making sure that people see your tool or technique, but it's also more than just getting noticed; it's about validation. It's about getting outsiders—maybe experts, maybe peers—to choose your tool and technology. Sometimes that can have more influence on your co-workers than anything you have to say.

Global Bug Tracker

Jim was tired of the old way he tracked bugs for his company's project. It was just too clunky. Users got some sort of error. They wrote to tech support, but they didn't always include the error messages. When they did, it usually wasn't enough to track down the problem.

Jim added some global error handling to his applications and used this feature to email the errors along with application state and stack traces. He wrote this into a new bug-tracking system that captured the information massaged for the company's environment. In short, he wrote a decent custom bug tracker.

Predictably (you've been reading this book for a while now), his teammates didn't want to use it. Despite that this system was easier to use, faster, more searchable, and so on, they would not use it.

Not wanting his tree to fall unheard in the proverbial forest, Jim released his code as an open source project. Other people working with similar systems liked it. They started using it. Some even contributed to it. In a few months, Jim had a small but global community of users.

By that time, word had gotten back to Jim's boss that he had a global community of developers using his tool. He started to ask how it happened and why it wasn't being used internally. When management finally wrapped their minds around it, they started to mandate its use.

18.1 Why Does It Work?

There's a meme that come from the Bible, "You can never be a prophet in your hometown." Basically, the gist of it is that when you have an important but perhaps controversial truth, the people who remember you were that kid who dipped Missy Funderman's pigtail in an inkwell won't listen. So, to be believed, respected, and followed, you have to preach outside of where you grew up.

The same concept holds true at work. You co-workers probably still talk about that time, two weeks in on the job, when you pushed a bug through to production and brought the whole system down. Every time you suggest something and someone wants to block it, they can bring that up and take you down a few pegs.

In any case, familiarity breeds contempt, and people often refuse to accept that things that come from their own organization or its people might be great. Getting publicity overcomes this by getting external validation of your tool or technology. You're biased, you wrote or packaged the tool, and you can't be trusted. But these strangers over here...they wouldn't lie to us.

I hope you picked up from my tone there that I think it is silly that we have to resort to this from time to time. We get hired because we are competent, and we have knowledge in our bailiwick. We shouldn't have to do this. However, that doesn't change that you do have to resort to this from time to time.

18.2 Seeking the Limelight

It's not like you can hire a publicist to get your message out there. You have to get recognition within technical communities. There are a few tried-and-true ways to do this.

Open Source Your Work

Open sourcing your work is a great way to get some recognition. It allows you to get noticed by people who are trying to solve similar problems you are trying to solve. Large open source repositories like SourceForge or GitHub make this easy and help you get noticed by allowing you to categorize and document your project. A side benefit is that you might even get people to contribute and make your solution better.

You Might Want to Check...

In this chapter, I recommend open sourcing software you're having trouble getting internal acceptance for. Depending on your industry and employer, though, you might need to secure their permission before doing so. Many employers have rules about the intellectual property that employees create during business hours.

However, don't let this discourage you from asking. I've worked at both a major research institute and a large software company. Both are stereotypically big protectors of intellectual property. Both let me open source things. However, your mileage may vary, so check your employee policy manual first before you do this.

However, before you rush out and do this, there are some things to consider. Managing an open source project is a lot of work even if no one uses it. Documentation, code readability, portable builds—these are all things you must have in place before you go open source. If people start participating, then you have to manage contributions, forums, and bug reports.

Also, there is a danger that you are imitating an existing open source project and will not get any traction with yours. You can mitigate this slightly by doing a little research ahead of time to see what else is out there. You can also ask the technical communities you participate in if anyone has any need for it.

I don't say all of these things to directly discourage you from open sourcing your project, but you shouldn't just knee-jerk do it. If your project is a good case for an open source project, you can get publicity by releasing it. If it's not, you'll just waste your time creating yet another abandoned open source project.

Participate in Contests

Contests are usually held by vendors, communities, or publications that participate in a technological niche. They tend to be designed to be a win-win for themselves and participants. They get demos, samples, white paper fodder, and use cases, while participants get prizes and publicity.

You usually have to jump through some hoops, use particular features, or demonstrate certain problems, but as long as your tool or technique can do that, you should feel free to use it to work on your contest entry. Then whenever you discuss your entry, include mention of your tool or technique.

Obviously, the risk here is that you will do work but not get recognition. That's going to depend on the size of the contest pool and other factors. Most contests try to spread out the recognition, but if you have 1,000 competitors, you're in for a fight. But the greater the competition, the greater the glory, so choose accordingly.

Put Your Work Up for Awards

Awards are given by the same groups to run contests. The difference here is that they accept preexisting work. So, there is little risk in doing a whole bunch of work for nothing—the work has already been done.

However, most awards have agendas. The sponsoring group has something that they want to promote, and they want examples of it. They may make that criteria public, or they might not. So, your control over success is not as great as when you enter a contest. However, there's little harm in trying, so I say take a swing.

Get Your Project Reviewed

You're not limited to just going outside of your organization to get publicity. Get some internal notice by using formal or informal code or project reviews. Reviews usually require some summary information that allows you to tell the story of how your tool of choice benefitted you. Then you show it as part of your code base or project plan. This gets your advancement out but under your terms.

The key here, though, is to make it a synchronous meeting. Face to face, remote meetings, or some sort of conference calls are the way to go. If you make it asynchronous, like via email or a wiki, you remove your ability to guide the story as well as you can with a meeting, where everyone is forced to focus on the same thing at the same time. And let's face it, if everyone paid attention to every email or wiki, you wouldn't have to be figuring out how to get attention, because everyone would already know about what you've been doing.

18.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Uninformed

Getting any sort of attention is obviously a way to turn the Uninformed into the informed. You might just have to make sure that they actually pay attention to where you are getting your publicity, but an email announcing a contest run or award victory seems to be a reasonable way to get on their radar.

The Cynic

Outside validation can definitely take the wind out of the Cynic's sails. Someone else values and respects your work. However, the Cynic may respond by simply devaluing the accomplishment: "Everyone knows that contest is crap." There's not much you can do to prevent it, but you might need to be prepared to combat that.

The Burned

This tactic will result either in other people valuing your tool or technique or in other people having success with your tool or technology. Either way, it can act as a powerful counterbalance to the experience the Burned have.

The Boss

As I have said, these tools or technologies often are outside the area of concern of a Boss figure. They cannot always wrap their minds around the technological advantages to your contributions. An "award-winning solution," on the other hand, is easy to understand.

18.4 Pitfalls

There are a few practical considerations here. Obviously, if the tool or technique you're trying to sell is an external one, you can't use this. If you are pushing Subversion, you can't enter that in any contests or release it as open source. So, only your own developed tools and techniques are eligible.

Your company may not like publicity for whatever reason and therefore will not allow you share company-developed systems with outsiders.

Additionally, there is luck involved: just because you release as open source or put your app into consideration doesn't mean you'll actually get the publicity. However, when you do get it, publicity is really effective. So, it comes down to a gamble—is the time you put in worth the possible reward? You have to answer that for yourself.

18.5 Wrapping Up

Getting publicity can be very effective at influencing people, especially the Boss. It overcomes a bias against internally discovered solutions. It's not appropriate in every case, though, because you have to own the solution to promote it.

Putting It into Practice

Here are a few suggestions to try to research publicity options:

- Figure out whether your tool has any competing open source solutions.
- Determine your employer's policy on open sourcing your work.
- Create a Google Alert for "contest" and your technology area.
- Do the same for "award" and your technology area.

Chapter 19

Focus on Synergy

Although we may live and breathe technology all day, unless you are employed by a technical firm, technology doesn't set the direction for your company. Your company works within an industry. That industry has regulations or specific concerns. On top of the industry-specific concerns, there are universal concerns such as security, waste, or environmentalism that all companies are grappling with. Those regulations and concerns sometimes require technology to implement them. Often they have the force of law or industry sanction behind them. If you can tie your tool or technique into implementing one of these concerns, then you get a powerful driving force behind you.

Spring Forward

Word came down from on high that because of the latest terrorist plots, sales of all components in every product in Argon Electronic's industry now had to be meticulously logged to a separate location in a specific format. The inventory system, written in Java, was not equipped to deal with it.

Markos had been trying to push Spring. It would solve many problems around the office. He was primarily looking at it to manage dependency injection. To date, he had met with little success.

However, Spring AOP allows for aspect-oriented programming, which lets you to wrap objects and intercept method calls to run code before and after without changing the underlying object. It's useful for managing *cross-cutting concerns*, or collections of code that need to be reused throughout all areas of an application. This wasn't just a good fit—logging was the textbook example of a crosscutting concern for use with Spring AOP.

Management was expecting a large and long rewrite. Markos's proposal showed that by using Spring AOP and some new logging classes,

it could be done in a fraction of the time. Markos got the go-ahead to use Spring in the project and got Spring in the door at the company.

19.1 Why Does It Work?

Despite what we in technology may like to believe, technology doesn't drive business; business drives business. Business needs will always trump technology concerns. By aligning your tool or technology to these business needs, you create a stronger argument for it. Additionally, you typically gain the notice/concern/protection of management by doing this. This gives you added ability and incentive to get your co-workers to adopt your methods.

19.2 Developing Synergy

There isn't a lot you can do to manufacture this. Either opportunities exist or they don't. You can be prepared for them, though.

First, most of these sorts of regulation take a long time to come to fruition. They rarely happen overnight. Keeping an eye out for upcoming opportunities is as easy as subscribing to a few industry-specific sites.

Second, try to fit the tool or technology you are trying to promote against larger concerns. Does it have security implications? Are they serious enough to use security as your lever?

Finally, listen to management. What are they saying their concerns are for the next year? Are you going through a financial crunch? Could your technique free up some waste spending? Are you looking to expand rapidly? Can your technique make scaling easier?

19.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Time Crunched

Problems that you can solve with synergy are usually unavoidable, which means that the whole point of these issues is that they have to be handled. You can't ignore policy changes set by management or regulation, which means that time will be allocated regardless of people's

objections. This is where you can gain ground with the Time Crunched. They've already been forced to spend time; that fight is already done. Now you are giving them a solution that might save them some of that time in the long run. You won't find a better time to try to get them on your side.

The Boss

It goes without saying that this tactic is designed with the Boss in mind. This is a prime example of making your tool a solution to their problems. The challenge here is to make sure that you frame things properly to ensure that they can see your tool or technique as the solution to their problem. Remember to focus on the business justifications such as cost, time, compliance, and work, as opposed to technical ones such as performance, encapsulation, and so on.

19.4 Pitfalls

This is one of those situational tactics. You have no control over whether this option becomes available to you. You can't orchestrate a change in regulation policy to force this to be an option. You need to have the right circumstances to have this happen.

Even if you get the opportunity, people may see this play as cynical or opportunistic and resist out of spite. Some of that is just human nature. However, you can mitigate this. There is the cliche "If all you have is a hammer, everything starts to look like a nail." Don't fall into that trap. Make sure that you really have a good match between business concern and technique. Make sure that you're not trying to shoehorn your solution somewhere you have to stretch to get it to fit. Reviewing Chapter 3, *Solve the Right Problem*, on page 21 can help you out here.

19.5 Wrapping Up

This can be a powerful technique, but only if the opportunity presents itself. The key here to keep in mind that your company exists for a reason. That reason is usually not technology. If you can align your tools around that reason, then not only are you going to get help from management to implement them, but you'll be serving the best interests of the company in a direct way.

Putting It into Practice

- Compile a list of major regulations in your industry. See whether any can be served with your technique.
- Determine whether your technique has implications for any of the standard hot-button topics:
  - Security
  - Financial waste
  - Environment

Chapter 20

Build a Bridge

Sometimes a particular tool or technology is just too different from what people are used to using. The barriers to fully adopting it are just too big. People have a limit to how much change they are willing to put up with in one shot. Perhaps your tool or technique is a new language in a different software platform. That's two jumps, not just one. You cannot get there in one step from where your organization currently is. In these cases, I suggest using a bridge.

Bridges are tools or techniques that aren't the ultimate solution you are looking for, but they also are not the status quo; they are an intermediate step between the two. The idea here is to use it for a while and then, when people are comfortable, move them on to the ultimate solution.

In some cases, you can find some sort of intermediate solution already available. This will be the "x for y" solution as in "Rails for .NET" or "code-behind for Java." They exist, and if you can take advantage of them, do so; they will save you a great deal of time. However, many times you have to be prepared to roll your own and build your own solution.

It's Not a Framework

I had just discovered a framework for ColdFusion named Model-Glue 2. It is a pretty typical MVC framework that integrated with a couple other buzzword frameworks (dependency injection and ORM). Through all of this it provided scaffolding so that from a database you could generate an application. It was pretty standard stuff across all languages and programming shops.

However, my co-workers were not too keen to jump to it. There were a few stumbling blocks:

- Our organization had a policy that all SQL had to be wrapped into stored procedures; this made adopting an ORM solution unlikely.
- The framework was an active generator, which resulted in a performance hit. This was merely perception, because there was a production flag in the configuration that improved performance. However, the default install took two seconds to display a "Hello World," which was a tough perception to overcome.
- There was a general bias against frameworks in general because of a bad experience with another one.
- MVC wasn't a generally liked pattern within the organization.

So, I focused on the part of Model-Glue I liked best: scaffolding. I started writing a code generator to do this for me. It grew over a few months, but here's the general idea of how it worked:

- It analyzed the database and generated CRUD stored procedures for each table.
- It then inspected the stored procedures to generate model components (analogous to classes in Java).
- It then generated UIs based on the stored procedures.
- It wired all the various CRUD views through a single controller per table. It wasn't true MVC, but it was closer than it was before.
- It did it all passively, so the actual application didn't have any creation overhead.

This had a few advantages. It didn't violate the constraint to only work in stored procedures. Since the code was passively generated, there was no perception of a performance problem. Also, I never called it a "framework." Instead, I used the term "code generator," which caused some skeptics to give it a fair shot.

Months later, every coding group in the organization had used my code generator on at least one new project. Even more impressive, some groups were even fooling around with Model-Glue, because the ground had been softened up a bit for it.

20.1 Why Does It Work?

It works because change is hard, and big change is harder than little change. Multiple little changes are often easier to achieve than one big

massive change. However, many little changes don't necessarily take longer than one big change. Big changes usually require pulling in more people. Big changes require getting clearance from higher and higher up the org chart. More people and more management add extra time, usually more than would be required of the same amount of change spread out over many, smaller changes.

It also works because you know your group. You know the gripes that people in your organization make, and you know what problems are the most painful—those problems are the ones you make sure you address in your bridge. In the case of the previous story, having to wire up all the database connection code to do CRUD was the pain point. Our bridge solved that problem and didn't require too much other effort by forcing other changes. In other words, we provided maximum benefit and added minimal cost.

20.2 Developing a Bridge

Developing a bridge is pretty straightforward. You have to look at your solution and your organization and figure out the pain points. Then either create or use a bridge that addresses them.

Survey the Situation

You have to take a look at your environment and your solution and figure out what the main points of contention between the two are. You then have to figure out what pieces of your solution are most going to resonate with your organization.

Suppose you are trying to introduce test-driven development (TTD) to a group that has never even done unit testing before. Unless highly motivated, your group is never going to jump to TTD—it's just too far. However, if your organization has problems with bugs fixes frequently conflicting with each other, you have a good case for unit testing on bug fixes. Looking at this, you can conclude that instituting unit tests on bug fixes, but not forcing TTD, is the way to go, because unit testing will fix a pain point in your environment without forcing the overhead of TTD. Further down the road, you can try TTD again, and the group will be closer to accepting it before they started unit testing.

Find a Bridge

When a solution becomes popular in one platform, it frequently gets ported to others. Look up *Hibernate in .NET* or *Rails for Java* or *Spring for Python*. Many bridges are out there and have already been written. In these cases, you can get away with just pulling one of those existing solutions in. It won't be custom molded to your organization, but you might not need it to be.

Build the Bridge

In the story example in this chapter, I didn't find any solution that could be a bridge between my desired solution and the status quo, so I built it myself. The idea here is that you can perfectly mold it to your environment. If you have conventions or best practices, you can make sure that your solution follows them. If all of your applications have boilerplate code, then include that. Basically provide as much advantage as you can squeeze into your solution, since you are designing and developing it yourself. The only downside is that it can be a great deal of work.

20.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Herd

The Herd need to be led. Building a bridge makes it easier to get to where you are leading. So, it's not a stretch to assume that this method is really effective on the Herd.

The Cynic

The Cynic (Chapter 7, *The Cynic*, on page 34) will disagree with you on points. That's a given. But if you are custom creating a bridge solution, instead of fighting with them, you can just give in. They don't like this aspect of it, change it. They have a problem with being forced to do something else; make it optional. Granted, you won't be able to change everything they criticize, but you'll be able to give them a good deal of it.

The Burned

The Burned (Chapter 8, *The Burned*, on page 38), like the Cynic, will benefit from the fact that you can customize. Whatever their past expe-

rience, you can customize around it. If a particular convention or pattern caused them trouble in the past, don't use it. Building in this extra custom padding around their pain points can make it much easier to get them on board.

The Time Crunched

The Time Crunched's main pushback is that they would love to try something better but can't afford the time. So, you meet them part or most of the way there. You formulate a bridged solution that doesn't require that they spend a lot of time learning something new, but they get a large benefit out of it.

20.4 Pitfalls

Even when you find and repurpose someone else's solution, building a bridge is a great deal of work. If you have to build your own solution, it's an order of magnitude more work. It can be daunting when you consider that it doesn't lead you to your desired destination but merely an intermediate stop along the way. But that intermediate step is almost always better than the status quo.

However, this does lead to the possibility that the bridge gets accepted, and you never move past it. This isn't always a bad thing. You just have to be flexible enough to accept that. However, it means that the temporary solution you developed now has to be maintained as its own entity going forward. If you aren't prepared for it, it's annoying when it happens, but I have to refer to the previous paragraph—it's better than what you have now.

20.5 Wrapping Up

I won't lie to you, bridge building can be a lot of work. But because you can custom tailor a bridged solution, it has a high probability of being effective as an intermediate step. You'll get people to move.

In some ways, it's stating the obvious, like telling you, "If you work hard, diet, and exercise, you too can lose weight." But it happens to be true. If you're not getting results with other methods, this is one to consider.

Putting It into Practice

Practicing bridge building is a bit daunting, but you can do some research to find something that exists already to act as a bridge, or you can build small bridges on your own.

- Look at your targeted solution, and see whether there are any clones that exist in the platform your team uses.
- Spend some time formulating a plan for writing a bridge solution:
  - Figure out what rules are sacrosanct in your group and need to be worked around.
  - Determine what parts of your solution will be the biggest winners.
  - Estimate how much work it would be.

Chapter 21

Create Something Compelling

Let's face it. If you could force people to use your tool or technique, you probably would, and you wouldn't need this book. But there is an indirect way of forcing people into using what you want them to use. You can build an internal solution requiring your tool or technique. If that internal solution is compelling enough, people will be forced to use the required bits.

Creating something compelling shares a lot with some of the other tools and techniques and in a way is composed of several of them. By creating something very useful, you Demonstrate the Technique. You'll probably have to Gain Expertise to accomplish it. You also Focus on Synergy, using your tool or technique to solve a company problem. All that is true, but this one does sit apart from the others because it goes beyond them in significant ways. It doesn't just show expertise and demonstration; it creates a real direct incentive for using the tool. It also is different from synergy because that requires business concerns to drive and management force to implement. This is much softer and gentler. People aren't forced externally; they're compelled internally.

Eclipse of the Art

Rupesh wanted to get his web developing co-workers onto Eclipse. He had a few reasons for this:

- He hated the proprietary tools they were using.
- He genuinely believed that Eclipse was a better tool set.
- He wanted to make is easier to push source control, build scripts, and continuous integration further down the line.

Every web project in the organization had one pain point: using company web templates for projects. It wasn't terribly hard, but it was very time-consuming. It required developers to add project branding to dozens of artwork images before any work could be done. The really annoying part was the work was rote and frequently had to be redone by hand if either the project's branding or the company's branding changed mid-project.

Rupesh had a little experience using Java and Java Image Management Interface (JIMI). After some trial and error, he made a program that could programmatically combine the branding in the images. Instead of just releasing it, he integrated it into an Eclipse plug-in. What used to take hours at the beginning of a project now took a few seconds.

Rupesh showed it off to other developers on the team. People were wowed. Even a few heretofore Eclipse haters made the jump, just to not have to manually work with all those images.

Now Rupesh could focus on the rest of the changes he wanted to push. Even better, he hadn't burned a lot of political capital on getting Eclipse accepted. Now he could spend that capital on getting other things mandated because he didn't have to get management to force this one.

21.1 Why Does It Work?

It works because you use people's own desires against them. Instead of being forced, nagged, or otherwise sold, a person takes up your tool or technique because it is the cost of getting something else that they really want. People are much more accepting of something they've resisted when they are the ones to choose to stop resisting, rather than having someone else force them into using it.

21.2 Creating That Something

The first step to creating something compelling is to look at your group's pain points. Every team has them. In the movie *Office Space*, one of the small repeating joke is the TPS report. It's never explained. All you know is that the character forgot to use the new cover sheet. But every single person in the company comments that he messed it up. No one seems to care about the content, simply that he didn't "Do it right." More importantly, that's a horrible process if it's that easy to screw up.

Think about your TPS report. Maybe it's documentation, maybe it's expense reports, and it could even be moves to production. It's some-

thing that takes up your time, is somehow necessary, but isn't part of your core work. Whatever it is, that is what you need to target with your solution.

Your next step is fixing that problem. Getting the solution right isn't guaranteed. Oftentimes, the reason the pain points remain is that there is no good way of automating or otherwise getting around them. However, a good deal of the time, the main obstacle is that no one has looked at the problem in a while. Many times, even as little as a year can yield new tools and technologies that can allow you to overcome the problem.

Once you have your tool written and ready to go, you have to show it off to your co-workers. Now, here comes a tough question, when do you tell them about the requirement? You have to be honest, but you don't have to lead with it. There's a big difference between "I have this great thing...just so you know, you have to use x to take advantage of it." and "Before I get started, you should know you have to...." Putting your best foot forward doesn't make you dishonest, and creating a disclaimer, when you don't need one, doesn't make you somehow more honest.

21.3 Skeptics That It Counters

This technique tends to be very effective against these skeptics.

The Uninformed

This can be a fast track for getting to the Uniformed. They don't know about the target tool or technique before your project. They learn about it over the course of using your project. Assuming you do things right, they'll have a favorable impression of both.

The Herd

Again, like the Uninformed, this can be a quick technique for getting the Herd on board. They are already forced to deal with the painful process that your demonstration project addresses. This gives them a choice: deal with the painful status quo, or use your easier painless options. You don't even have to lead them; it ends up being the path of least resistance.

The Time Crunched

Because this technique attacks a pain point, it is particularly effective on the Time Crunched. Not all pain points are time constraints, but

most are. The others tend to be financial constraints, and if you remove them, you can use that money to buy more time.

The Cynic

The Cynic types are where they are because they bet against success. Using your tool or technique in a tool that demonstrably fixes a pain point takes the wind out of their sails. That doesn't stop them from disagreeing that you adequately fixed the pain point, but it decreases their ability to convince other people of their position.

The Burned

Like the Cynic, you are proving the Burned wrong. The tool or technique can be used successfully. As always, you have to make sure you aren't attacking the Burned, but you have proof that they can get back up on the horse.

21.4 Pitfalls

First, this technique is predicated on a bunch of *ifs*. If you can find a pain point, if you can use your solution, if you can make it compelling, and if you can convince people to use it, this can be successful. So, some of this, like other techniques, depends on having the correct opportunities. But you need to be looking for them. Opportunities rarely gift wrap themselves.

Just because you tie your solution to something doesn't mean a co-worker can't untie that knot. In the previous Eclipse story, it's just a bunch of Java code thrown into an Eclipse plug-in. You don't have to be a genius to ask whether it could be pulled out. If that happens, you have cool solution that doesn't help you with your goal.

Also just because you get people to use your tool or technique, this doesn't promise general usage. Again, looking at the story, there's nothing preventing Rupesh's co-workers from firing up Eclipse just to do the pain point step and going back to some other IDE to do the rest of their work. In other words, this method might get your foot in the door, but it's not going to get the whole job done by itself.

21.5 Wrapping Up

So, this solution isn't going to get the whole job done for you, but it has a few major pluses. People don't resent the tool or technique, because adopting a compelling tool feels better than being mandated, directed, or commanded. In the process, you fix a pain point for your group, which isn't a bad thing. And if you can get the opportunities to further demonstrate that this is a viable technique, it can be really effective.

Putting It into Practice

Doing a full-blown solution for a group-wide pain point from nothing may be a little daunting. There are a few things you can do to make it a little less intimidating:

- Figure out your group's pain points. (You should do that anyway.)
- See whether any of them could be solved with your tool or technique.
- Spec out a solution, and see whether it is worth the work.

Part IV

Strategy

Chapter 22

Simple, Not Easy

Over the past chapters, you have amassed a collection of skeptic patterns to identify and techniques for combating them. Which technique to use with which skeptic is summed up in Figure 22.1, on the following page.

Together the skeptic and technique pairings form a great set of tactics for you to use. But that's not enough; you have to apply them effectively. For that, we need a strategy.

My strategy for winning this fight is extremely simple. But simple does not mean easy. Getting a boulder to the top of a mountain is pretty simple: just roll the boulder uphill until you reach the top. But that is by no means an easy task.

Keeping the "simple but not easy" mantra in mind, here's the grand strategy:

- Ignore the Hostile
- Target the Willing
- Harness the Converted
- Convince Management

That's it? That's it. You basically do your best to ignore those who are downright antagonistic, not wasting any effort on them. Then you apply your tactics to the most willing people you have access to. Then you motivate your converts to participate in a tactic or two. They become advocates, and you continue to the next most easy group and repeat. You do this until you convert everyone you can.

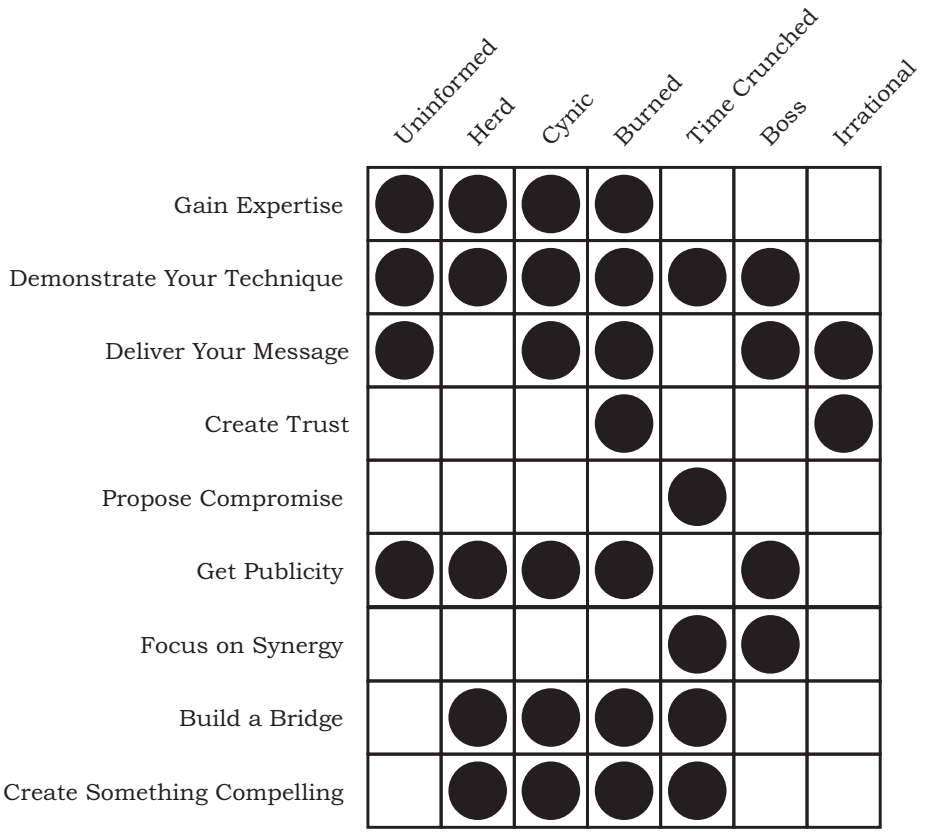


Figure 22.1: The skeptic/technique matrix

If you convert everyone, congratulations—you're done for now. If not, you have a decision to make. Do you have enough people on board to make the tool or technique worth it? If so, you're done as well. If not, you finish up by getting management on board. If all goes well, management then makes your tool or technique the mandated solution.

Simple, right? But have I told you yet that it won't be easy?

Chapter 23

Ignore the Irrational

This strategy might seem a little childish at first. Not including people, even the Irrational, in your efforts might seem a bit passive aggressive. You have to eventually deal with these people, right?

No.

Oh, I'm sorry, you wanted a deeper answer?

OK, the whole point of the Irrational skeptic is that they aren't resisting for valid rational reasons. Their motives are mysterious and often unknowable. Because of this, they require an inordinate amount of effort to first discern their true motive and then to overcome it. Typically, this is going to be some sort of personalized solution. Just showing them, for example, that you're an expert in the tool will not be enough. You'll have to demonstrate that whatever irrational, maybe even impossible problem they have is not relevant to the professional development tool you are trying to introduce.

It's also possible that once you figure it out, you won't be able to overcome their motive. Maybe they have a personal beef with you or your boss or your department. In any case, they're not going to budge. Getting them to relent on these types of issues may take months or even years.

So, all of this adds up to a high cost, low return for our effort. Why bother? There are other skeptics in the sea. We don't have to waste our effort on these guys.

23.1 What Exactly Does This Mean?

So, do you need to perform some sort of ritualistic shunning process for these people? No. What I specifically mean is don't try to sell them. Don't actively pursue them. Don't seek them out.

However, that doesn't mean don't talk to them. If you're in the midst of some sort of tactic-driven event, such as a team meeting where you are discussing a development tool and they engage, go ahead—engage right back. Give them information while still using your tactics like expertise and delivery.

While doing all this, make sure they don't manipulate you by stealing all of your attention. Answer them, and move on. If they are trying to monopolize, gently point out that there are others present, and you would gladly meet with them later to discuss.

23.2 Why Is This Challenging?

Conflict, disagreements, and other arguments can be incredibly productive. However, they can raise hostility or anger. The longer a conflict goes on without any weakening of resolve on either party's part, the more likely anger and hostility will grow. With the Irrational, this means that anger and hostility are almost certain.

Prolonged hostility will cause us to view our opponent as an enemy. Enemies are to be overcome; that is our instinct. We overcome enemies by engaging them. Engaging the Irrational is the opposite of what we want to be doing.

The key here is to try to think of the Irrational as an obstacle and not an enemy. Enemies are defeated; obstacles are overcome. Overcoming an obstacle can be as easy as walking around it.

So, engaging the Irrational is a waste of time and energy. But they can argue with you enough to make you want to defeat them as an enemy, which requires engagement. Therefore, the key is to avoid them as one would a pit of spikes, instead of seeking to beat them in a bare-knuckled fight to the death. That sounds a lot like "Ignoring the Irrational," which is what I said in the first place, but you wanted a deeper answer.

Chapter 24

Target the Willing

You've put a pin in the Irrational for now, and you want to start converting people. Who do you start with? Well, start with the easiest to convert, that's who.

You might think it was the other way—convert the hardest, and then the rest will fall in line. But in practice that doesn't work. It's too easy to claim that no one else is doing it, so why should I, in these situations.

However, as we will see in Chapter 25, *Harness the Converted*, on page 115, the converted can be powerful allies in this effort. The more converted you have, the better off you are. So, this is about quantity and not quality. It doesn't matter that Bob is a better programmer than Ed and Steve combined; you're better off focusing on Ed and Steve if they are easier to convert and then enlisting their aid in bringing Bob into the fold.

24.1 Order of Difficulty

In my experience, you can break the groups of skeptics into three groups of difficulty: easy, hard, and harder. Why no *medium*? Well, in my experience, there is a big delta between the first group and the second group. When you are finished with the first group, be prepared for a much harder time when you move to the second. It's important that you understand this, because taking on the second group can be a real morale killer for you if you are unprepared for the difference in difficulty.

24.2 Easy

Far be it for me to call anyone easy. This group of developers isn't easy in the sense that they are pushovers. They're easy in the sense that you know what you have to do. You inform or lead depending on the group. You're taking a stationary rock and starting it rolling.

Uninformed

The Uninformed are your first stop. You completely control their first exposure to a technology. Everything they think about it will be compared and contrasted to what you told them about it. That's a huge advantage for you.

Think about every email forward you've gotten from a relative warning you about an urban legend. Have you ever tried to refute them? It's almost impossible. Even with great skeptical sites like snopes.com, people don't listen. Why? It's because you're showing up after the item has been absorbed and therefore are refuting what they now believe. It's much easier to shape an opinion than it is to change one.

Now, I'm not suggesting everyone is as willing to believe something as your email-forwarding relatives. But the first impression effect is very real and very powerful. So, use it to your advantage—be the first to define your tool or technique.

Herd

The Herd are for the most part looking for leadership. The cost of getting them on board is providing that leadership. Now, don't think that it's as easy as "Use this."

True leadership in this case will mean providing ongoing support and guidance to those you seek to lead. You'll have to help them out when they hit a pain point. You need to provide them with arguments for why they are using your tools from time to time.

24.3 Hard

If converting the first group is like starting a stationary object rolling, then this group is like reversing the direction of an already rolling object. It much harder. You have to overcome the inertia of these guys, who are not neutral on what you're pitching; they're opposed.

Add to it that there isn't one thing that will convert them. One demonstration, shared project, or group success will not convince them. It takes combining multiple successes to do so. And that combination is unique per person. So, the difficulty factor is higher. But they can be convinced.

Burned

Of this group, the Burned are, by a slight margin, the easiest. At one point, they thought your tool or technique was a good idea, even if they don't now. This is a good thing; they have the capacity to accept your tool or technique. You just have to reignite it. However, it might not feel like they are the easiest, because the Burned can be the most passionate of skeptics. They can be violently disagreeing with you until a moment before they decide to give your tool a try.

Time Crunched

The only thing in your favor with the Time Crunched is that their opposition to your tool or technique is not specific. They don't oppose your tool; they oppose a change to their current way of doing so—they cannot tolerate the time cost of a change.

The Time Crunched won't give you a shot unless you can prove that your tool will ultimately save them time. So, the point of hitting them up later has to do with providing proof. By having the Herd, the Uninformed, and the Burned on board already, you have opportunities to provide third-party proof that you can save them time. It's not definitely going to bring them in, but it does make it more likely.

Cynic

The Cynic often feels like a lost cause. They have a horror story or statistic for every argument or point you have. It doesn't look like they can be brought over. However, they may have a huge weakness: they are motivated by a need to appear smart (see Section 7.1, *Underlying Causes*, on page 35). So if other smart people are jumping onto a technology, they can be brought on board too by appealing to that desire. But to do that, you have to have smart people on board, which is why they are one of last groups to try to get.

24.4 Hardest

The hardest cannot be approached until you at least have some other group on board. You can skip the easy if they don't exist at your workplace. But you cannot approach the hard group, until you at least have some converts on your side.

Boss

Bosses are the hardest to convince. They take a lot of effort. You have to approach them on special terms, specifically their terms as noted in Chapter 10, *The Boss*, on page 44. Additionally, you're not trying to influence them in a vacuum. Everyone wants a piece of their time and attention, not just for issues related to your matter but all sorts of administrative minutiae. It's not just about making the argument to them; it's about making them care.

Adding to the difficulty factor, in most cases you pretty much cannot do this alone. You need other converts to prove to the Boss that your idea has merit. If you cannot convince co-workers who are ostensibly your peers to come over, why should the Boss? And they're not wrong to have that attitude. What's more likely? You didn't mention your tool to the rest of your team and instead talked to your Boss first, or you tried to get them on board, no one bought it, and you're trying to get them to mandate it. It's pretty clear that if you're talking to the boss without support, it's because you failed to get it, not that you were keeping your discoveries a secret.

Now these are general guidelines. They tend to work. But don't be rigid with these. If you believe you have a good chance to convert a harder skeptic out of order, don't hesitate. The idea here is to go after the easiest first. If circumstances align to make a Cynic or Boss temporarily easier, you should definitely go for it.

Chapter 25

Harness the Converted

At this point, you should be avoiding the Irrational like the plague. You should have some people converted to your technique. But you still have skeptics. You think that you can bring them over, but they haven't succumbed to your wiles yet.

There are lots of reasons that people might not be swayed by you. Some might remember that first day on the job when you wiped out 1,000 accounts with a typo. Some may only listen to advice from specific people, having always been happy with what they say.

At this point, you need some help. You need to call upon all the people you have converted thus far to join the fight. You need them to try to convince skeptics. You need them to give your solution visibility.

For them to do so, you need to equip them to be able to add to the discussion. You need them to go after people they have influence with. Finally, you need them to help drown out the naysayers.

25.1 Request Help

If you intend to enlist the aid of your converts, you have to ask them for their help. If you want them to actively participate in the effort or even if you are merely repeating any opinions they have expressed, you have to absolutely have to ask. Having one of your converts find out that you are speaking about them or their using words without their permission will probably lose you a convert and damage your credibility.

In the process of asking, you need to let them know exactly what you are doing, as in, "I am trying to get the company to standardize on [x].

You converted to [x]. I would like your help—are you in?" This example might be a bit stiff, but you get the idea—full disclosure.

Sometimes people are hesitant at this part. It's one thing to ask people to join in working on a technical issue. It's another to ask them to join in on a workplace political issue. But remember you're trying to make the workplace better. You're not engaged in politics for politics sake; you're working to a result. If you can grasp that, so can your co-workers.

Without being completely honest with your converts, you run the risk of manipulating them instead of working with them. If you manipulate them, you're using them. If you honestly ask them, you're working with them.

In addition to the problem of actually using people, there's the problem of looking like you are using them. This is even tougher to fix than using them. You have control over whether you are using people, but you only have influence when other people *think* you are using them. Honesty and disclosure up front will usually prevent this.

So, in all cases, just open up, explain what you are doing, and ask for help. Some will say yes, and some will say no. Work with the yes people, and respect the choice of the no people. Then get busy converting the rest of your co-workers.

25.2 Create Evangelists

They've agreed to help, and now you have to help them help you. You need to arm them with tools that they can use when they are faced with skeptics. Again, disclosure is key.

First let them know exactly why your tool or technique is important to the wider group. "We spend too much time on maintenance of old spaghetti code. If we standardized on framework [x], we could reduce maintenance time and have more time to focus on...." Be that explicit and simple; make sure people know what they are fighting for.

Next, come clean about the skeptics they're up against. Let them know why one or the other is skeptical. Let them know who is the Burned and who is Time Crunched.

Then, let them in on the whole plan. Let them know who you are targeting, in what order, and why. Let them know your idea of success and when you would consider this campaign done.

This Book Isn't Secret

You don't have to hide the information, tactics, and strategies in this book from your converted. This system is a way of structuring your efforts, but it doesn't actually do anything by itself. It's your content, arguments, and examples that do so. This system doesn't rely on the ignorance of others to achieve your goals. So when disclosing, don't hide it.

Finally, send them out. Have them engage your skeptics, and have a go at changing some minds.

25.3 Cross-Promote

Once your converts are out working for you, you need to work for them. You have to promote their efforts. Talk to co-workers or management about their successful projects they've produced using your tool or technique. However, when describing their successes, don't make that they are using your preferred tool or technique the leading story. Just promote your converts. Make your efforts here about them, not about your solution. If and when the interested party asks for more information, then include your solution in the description.

Again, clue your converts into this. Have them talk your technique up with the same caveat. Ask them to mention your solution first and then mention their own success as part of a deeper dive.

Why the emphasis on the other person's story first? Because people are less willing to hear you blow your own horn than trumpeting another person's success. We've gotten numb to the shameless self-promoter. We ignore them. So, don't self-promote. Promote others, and let the audience's curiosity lead them to you. It won't always. That's fine. It's powerful enough when it happens that it's worth the effort.

Cross-promotion is a tool of opportunity. You do have to wait for it to be appropriate to talk your converted up. Then you have to be asked for more information. But you can make your own opportunities: "Did you hear about the Beta group?" And for the most part, when you discuss a success, most people want to know how it was done.

25.4 Consume Attention

Attention within a company or organization is a zero-sum game. People's attention is finite, and for one group to gain attention, others have to lose it. This means that by seizing attention, not only do you and your converts give yourself a leg up, but you are diminishing the attention your skeptics have, making it easier to drown them out.

Grabbing this attention will vary with your organization. Do you have an internal blog or message board? Then you and your converts should be discussing your tool or technique there. Do all of your co-workers use either public or private social networking tools? You should be shouting it there. Do you never have anything to say when you are in an elevator with a higher-up? Cross-promote then and there.

Basically, if there is an appropriate place to either toot your own horn or cross-promote, do it. If there are organic opportunities to do so, do it. Create an environment where, for every skeptic's message, there are multiple messages from your side, drowning it out.

Harnessing converts is a powerful part of your strategy. One person can be easily dismissed, but with each person that repeats your message, it becomes harder to ignore it. However, this part is also full of ethical peril. Yes, you should cross-promote and stand out from the skeptics, but be careful not to lose sight of the lessons of trust from Chapter 8, *The Burned*, on page 38.

Chapter 26

Sway Management

Some tools and techniques require 100 percent compliance. You can't always convince everyone, and you sure can't convince the Irrational. So, your goal in this final step is to convince management to mandate, or force through policy, the use of your tool or technique. Often a mandate from management is the only path to get those stragglers on board.

26.1 What Do You Want from Management?

Simply stated, you want management to set a policy that your tool and technique must be used in its appropriate setting. Quite simply, for all of your skeptics, doing what you want them to do is now part of their job.

The mechanics of how this is enforced will vary. Usually people are told they must do things. They must justify noncompliance. Depending on the tool or technique and your organization, employees might have compliance become part of their annual review.

Gatekeepers are also a possible enforcement avenue. Gatekeepers control resources that your skeptics need to do their job or get into production. For example, if you need a server admin to put your code onto production servers, then they can check for compliance with policy before they publish.

26.2 How Do You Get It?

Convincing management will vary in difficulty depending on the formality of the mandate and the flexibility of your workplace. Regardless of these factors, you will not be able to sway them at all unless you are capable of showing them that they need to, that you're not a crazy lone wolf, and that you have a reason for involving them.

Solve Their Problems

As stated earlier in Chapter 10, *The Boss*, on page 44, you need to solve your boss's problems, not your own. You need to show that the technical problem that you have translates into a business problem they have.

Translation is not terribly hard to do, once you train yourself to do it. Wasted developer time is wasted money. Performance drains are wasted hardware resources. Insecure environments are potentially wasted money or legal liability depending on your industry.

You do need to go one step further. It's not enough to say that a change will save developer time. You have to calculate just how much time you are saving. You then need to convert that to money using salaries or estimates. Then you show the potential savings to management. That's solving their problem, in their language, on their terms.

Show Your Numbers

You're not alone. (You did the previous steps and have some converts, right?) You must not appear to be alone. You need to make sure management knows that you now represent a group and aren't some lone troublemaker with an axe to grind.

Even if you are right and they believe you are right, they won't be swayed by someone alone. There's just too much risk in issuing mandates to do it on just one person's word. When you're one against many, it looks like there might be good reason to say no. When you are many, it looks like your idea has merit.

Explain Why You Need a Mandate

For the most part, management is not going to be thrilled to see you bringing a technical issue like this to them. Let's face it—you're complaining about a problem. You then transform that problem, making it

their problem, and you ask them to solve it. They're going to want to know why they have to get involved.

As in previous situations, the answer is disclosure. Tell them about your campaign. Tell them you have tried to convince co-workers and enlisted converts. Let them know that you exhausted your own resources before calling on them.

26.3 Now What?

With any luck, management agreed and has issued a mandate. Problem solved. Right?

Not quite yet. Just because compliance is mandatory doesn't mean that compliance is compelled. You have to monitor, track, and ultimately report compliance. Why you? It's because this whole thing has been your baby from the start. Without that vigilance, the mandate is useless. Now, I'm not suggesting you have to go all Secret Police on your co-workers, but you do need to encourage people to comply.

To be completely clear here, this step is optional and not always called for. You may very well be successful in converting people through the previous three methods. Additionally, your solution might not require 100 percent compliance, and you're cool leaving the stragglers behind.

Be aware that getting a management mandate is the atomic bomb of this type of conflict. Sure, it has tremendous impact and might achieve your aims, but people might not care much for you after you use it, and it does not guarantee success. Convincing people is better than compelling them, but that route is not always open to you.

Chapter 27

Final Thoughts

You've made it. You ignored the Irrational. You lined up your skeptics with tactics and made converts. You and your converts either swayed everyone to use your tool or technique or swayed management to mandate it. Your solution is in use, and you are done. Is that it? Of course not.

27.1 Cautionary Tales

Even when you are successful, there are pitfalls. It's human nature to assume that the end of a process results in happiness, peace, and light. Sadly, that isn't the case. Here are some ways success can be a downer.

Too Successful

Stored Procedures Again

A few years back, I worked at a web development shop that had a problem. The web application servers kept intermittently crashing. It was traced back to the connection between the application servers and the database.

Upon review, it became clear that the cause was the occasional poorly written database operation. Our team of web developers were talented with client-side code and business logic but were not DBAs. The solution was the dreaded *stored procedure rule*: all SQL had to be in stored procedures, and those stored procedures were all reviewed by DBAs before being put into production.

It was one of my first clear successes with some of these techniques. The DBAs convinced many developers to participate. The rest were forced by a mandate from management.

That solved the problem. Database-related crashes went away. The application servers became much more stable.

Flash-forward five years, and I'm having drinks with a recent hire at the web development shop.

"I'm so frustrated. I want to bring in ORM frameworks, but all of them require inline SQL. The DBAs won't allow it. Despite that, the frameworks that I want to use write better SQL than I can. I want to murder whoever set up that policy in the first place."

I shifted uncomfortably and made sure I picked up the drink tab.

It turns out that the DBAs were now enforcing the stored procedure rule, without knowing the reasons behind them. The technique and mandate had outlived the reason for them.

Yes, it is possible that once you sell something, it cannot be undone. This becomes increasingly difficult to deal with when technology shifts render it obsolete.

In the previous example, I made a mistake. I drummed the policy and technique into people's heads without drumming the reasons. Would the DBAs have been willing to lighten up if the reasoning was made clearer to everyone? Probably, but we'll never know.

Make sure you explain the why and not just the what.

You Don't Always Reap the Rewards

Finally

I left an old employer of mine feeling like a failure. I had pushed frameworks, code generation, and unit testing to no mostly no avail. I had heard pretty much the same old tired Time Crunched skepticism for them. After years of trying, I figured they were unmovable.

Again, flash-forward a few years, and I'm talking to a former co-worker. I had pushed to get him hired because he was an awesome developer. Part of the reason I thought he was a great developer was his enthusiasm for the very things I was advocating.

He started complaining about the group:

"So, they didn't like this framework because it's a little complex, and they didn't like this other one because development has slowed down on it. So about five months ago, we went with plan C because it allows for... "

Let me get this straight. The people who refused to even consider frameworks were using one, after arguing about which one to use at

length? Oh, and by the way, they're also using a unit testing framework, automatically generating starter unit test, and adding their own.

So, despite my best efforts, the group wasn't sold, at least not while I was there. More than two years had passed since I had been working with the group, and finally there were some changes.

It's possible that I had softened the ground. It's possible that I had alienated some people, who were able to say yes to the new guy when they couldn't say it to me. It's possible that slow change is the only possible change with this group. Whatever the case, change did happen, just slowly.

You Can Be Wrong

Right Answer, Wrong Answerer

It was another success story. I took a group with no source control and sold them on Subversion. They migrated, and we were enjoying sleeping at night without a panic that we would lose the whole site because of an errant `rm *`.

There was one problem. I didn't know squat about setting up an SVN server. The architecture I built up for our repository structure was a nightmare. It worked for a small number of projects, but once we got in the double digits, performance lagged, checkouts were confusing, and people started to not participate.

I delayed progress. I screwed up. I ended up creating Burned skeptics.

No one is perfect. No one can see unforeseen consequences. That's why they're unforeseen. Experience helps, but you're going to screw up. Accept it.

More importantly, own your failures. Admit them. Advertise them when they happen. Most importantly, explain them.

In the previous case, Subversion wasn't the problem—I was. I admitted, advertised, and explained. Because of that, I was afforded the time to migrate to a different Subversion solution. It worked, and I stopped people from getting Burned—but only because I owned the failure before I tried to fix it.

27.2 Success Is Siloed

Once you achieve some success, you can sometimes lose sight of the fact that your co-workers are still skeptics. Sure, they're with you on

what you've sold, but that doesn't make them any easier the next time you want to sell something.

In fact, with the amount of change present in our industry, there are so many thousands of tools and techniques that we could be considered Uninformed skeptics merely because we don't know about them.

The takeaway here is that yes, you do have to start over each time. Multiple successes may raise your credibility, but they don't carry over much more than that. It's frustrating, but it is the way it is.

There are a few things that you can take from attempt to attempt.

- People who you've identified as the Herd, the Cynic, and the Time Crunched are likely to be the same for other attempts.
- The Uninformed is slightly more likely to be Uninformed in other attempts.
- The Boss is usually the Boss time after time.
- The Burned is a crap shoot and has no impact on other attempts.
- The Irrational can also be a crap shoot, but if you've been targeted by them personally, as opposed to technically, then they'll probably give you a repeat performance.

27.3 Problems Always Expand

We often have a delusion that once we get people to use our tool or technique, we'll eliminate a set of our problems. In this mode of thought, problems are finite, and we will reduce our problems. Sadly, problems are infinite. Problems are a gas, not a liquid. They always expand to take up as much room as they can.

Maybe saving a whole bunch of time will result in fewer billable hours and people losing jobs. Maybe fixing one broken area will reveal a weakness in another. Whatever the case, the world won't be perfect when you are done.

27.4 A Journey, Not a Destination

The previous sections have been a bit of a downer. You could come to the conclusion that even if you manage to get people onto your solution, it doesn't matter. Things will go wrong, or bad tools will become

entrenched. Even if you do manage get people over, you'll just have to start over from scratch, and more problems will take their place.

That conclusion is only correct if you can't change your mode of thinking about your workplace environment. Better isn't a place; it's a direction. Acceptance of your technical solution isn't a destination; it's a journey. Between where you are and where you want to go, there are many much better places. Focus on leaving where you are, instead of where you want to go.

So that's it. Get out there and start trying to make your workplace better. It can be done. It has been done. You just need to add yourself to the list of people who try.

Appendix A

Bibliography

[HT00] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, Reading, MA, 2000.

[Mas06] Mike Mason. *Pragmatic Version Control Using Subversion*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, second edition, 2006.

[RTH08] Sam Ruby, David Thomas, and David Heinemeier Hansson. *Agile Web Development with Rails*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, third edition, 2008.

[Rud07] Jason Rudolph. *Getting Started with Grails*. InfoQ, 2007.

[Swi08] Travis Swicegood. *Pragmatic Version Control using Git*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2008.

[TH03] David Thomas and Andrew Hunt. *Pragmatic Version Control Using CVS*. The Pragmatic Programmers, LLC, Raleigh, NC, and Dallas, TX, 2003.

Index

M

N

O

P

Q

R

S

T

U

V

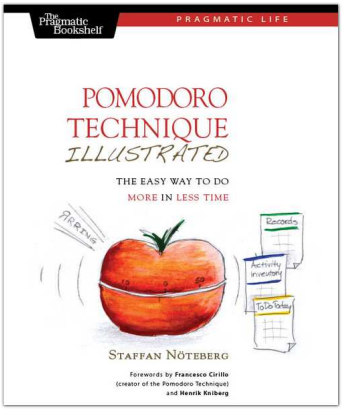
W

Z

More Good Ideas

Pomodoro Technique Illustrated

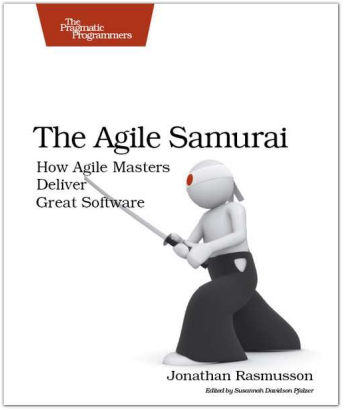
Do you ever look at the clock and wonder where the day went? You spent all this time at work and didn't come close to getting everything done. Tomorrow, try something new. In *Pomodoro Technique Illustrated*, Staffan Nöteberg shows you how to organize your work to accomplish more in less time. There's no need for expensive software or fancy planners. You can get started with nothing more than a piece of paper, a pencil, and a kitchen timer.



Pomodoro Technique Illustrated: The Easy Way to Do More in Less Time
Staffan Nöteberg
(144 pages) ISBN: 9781934356500. $24.95
http://pragprog.com/titles/snfocus

The Agile Samurai

Faced with a software project of epic proportions? Tired of over-committing and under-delivering? Enter the dojo of the agile samurai, where agile expert Jonathan Rasmusson shows you how to kick-start, execute, and deliver your agile projects. You'll see how agile software delivery really works and how to help your team get agile fast, while having fun along the way.



The Agile Samurai: How Agile Masters Deliver Great Software
Jonathan Rasmusson
(275 pages) ISBN: 9781934356586. $34.95
http://pragprog.com/titles/jtrap

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

Home page for Driving Technical Change
http://pragprog.com//titles/trevan
Source code from this book, errata, and other resources. Come give us feedback, too!

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

Join the Community
http://pragprog.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://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

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

Contact Us

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