Why Beautiful Teams?

We've been on a lot of different teams over the years, in a lot of different companies and building a lot of different kinds of projects. And over the course of writing books, articles, and blog posts about how to make projects run better, we were always nagged by a question. It always seemed like it was our job to come up with prescriptive, "best" ways to run software projects: how to plan the projects, how to build the software, and how to make sure that it doesn't break. But the more we wrote and the more we talked to people, the more we questioned this idea.

We started down that path after writing our first book, which we thought of as a practical recipe book for running successful software projects. We'd done a lot of research, experimentation, and real-world project work over the years to find practices that worked for us: ways to plan software projects, techniques for developers to write better code, and ways to test the software. We took the ones that worked best for us and packaged them up into as lightweight a book as we could come up with. We've gotten a lot of great, overwhelmingly positive feedback about it over the years, and a lot of people have told us that they use it every day.

And that's where things started to go wrong for us.

Ironically, we fell into a trap that we actually wrote about in that book: we started to get a kind of tunnel vision. We saw this particular set of practices as the "successful" practices. We never intended to say that there was only one way to plan out your project, or to estimate it, or to do a code review, or to test the software. But what we found, as we started getting out into the world and talking to more and more software people, is that we were getting pigeonholed. People would say, "Oh, Stellman and Greene—you're the Wideband Delphi people? I always use that to estimate!" (Seriously, people actually said that to us!) Or, much worse, they'd tell us, "Your book talks about requirements specifications, and I never use them. But I develop software just fine. You must be wrong!"

Practices are cold. Practices are theoretical. You can sit and talk about the virtues of one or another, and have hypothetical arguments about whether they'll work in one situation or another. We've done our own share of arguing about that ourselves, talking until we're all blue in the face about when is the correct time to do requirements gathering, how to make a project more or less agile. Those kinds of things are really where people draw battle lines.

But it's not where the meat of the work happens, honestly. How you make those decisions has an impact on the project, certainly, and sometimes a big one. But it's not nearly as important as *who* you have on your team: how skilled they are, and how well they work together. That's when we realized that practices are only one aspect of building better software. And although it's the aspect we've spent the most time studying and dealing with, it took us a long time to realize that it's usually not the most important aspect.

The more we opened up and let other people talk about their own experiences without trying to impose our own practices and views on them, the fewer arguments we got into. And the more we talked this over, among ourselves and with other people, the more we started thinking about four distinct "buckets" (for lack of a better word) that these conversations fell into: *people* (who's on the team), *goals* (what brings them together), *practices* (how they build the software), and *obstacles* (what gets in their way).

So we started looking for ways to get people to open up about their own projects, and to listen to each other about what they've learned. We put together a talk about best practices—because that's what we wrote about and what we knew best—and tried to explain, in as general and non-prescriptive terms as we could think of, how to use them to make your projects run better. We naively thought that people would be excited to hear about what we learned, both in writing our book and in running our own projects. We'd talk about the kinds of pitfalls they'd avoid, and we'd give them the tools to avoid them ("Look, here's one way you can run a productive estimation session!").

It was a disaster. One talk comes to mind as particularly bad, although not too much worse than many others that we gave. We were invited to give a presentation for a brown bag series at a major Internet company's New York office. We started in on the talk, going through the kinds of problems that software teams typically face, and outlining the practices that we found to be useful to prevent them. We could see the mounting boredom and even frustration on the faces of the developers. They were mostly young, the majority under 30, and by the time we were halfway through the talk most were checking their PDAs and laptops. Some people even got up and left. And when we got to the Q&A session at the end, we found out where their heads were. One programmer, a large guy with a long gray beard, challenged us when we mentioned agile project planning: "Agile means that you don't write anything down, you just start coding immediately." Everyone nodded their heads. We tried to redirect that—"No, that's not what agile necessarily says!"—but it didn't matter. Our message wasn't getting through, and we were clearly wasting everyone's time.

We shouldn't have been surprised that we were met with a lukewarm (at best) reception. After all, we'd sat through our own share of networking activities (project management group meetings, architect SIG meetings, software process network meetings…). And for every one that featured a really memorable talk or discussion, there were literally a dozen that boiled down to an advertisement for a consulting company, or a book, or a professional product or service of some kind, and left us completely cold.

So we reflected on the criticism about agile we'd gotten at that talk. And to our surprise, we realized that he wasn't wrong to criticize us, even if we disagreed with him on the facts. Yes, the criticism stung at first, and we really had to take some time to figure out why the talk ended up the way it did. But in the end, his criticism ultimately showed us what was wrong with our approach. *There is no "best" practice*—at least, none that we know of that will guarantee success every time. Whether or not a particular tool or technique works depends on the circumstances: the project, the people, and all sorts of mushy, messy stuff that you just can't quantify or prescribe. Now, that's not to say that we don't care about process. We're not distancing ourselves from the practices that we use, and that we write about. In fact, just the opposite: we still use them every day, we still write about them, and we still train other people to build better software using them, because they really do work—in the right situation, for the right project, and with the right people. But sometimes the way to get something done is to do anything *but* the "right" thing. Sometimes you need to just do whatever works.

Growth for us came from realizing that there really are a lot of ways to run a project. And even though we found some practices that we're very comfortable with, there are times when they're not appropriate. We've had to get a lot more flexible over the years with our own approaches to building software, and how we work with our own teams. The reason we had to do that is because some of our projects failed. Not many, but some. And when they failed, that's when we learned the most.

That's what set us down a new path. We tried to figure out what made some projects work while others crashed and burned. It was hard to figure out one particular cause at first, and we suspected that there was no single answer. And when we started talking to people around us, people we knew, we started to hear the same thing over and over: people talked fondly about one project or another that really stuck with them, and they wanted to share their "war stories" about the terrible manager they had, or the horrible situation they faced, or another problem they overcame. In fact, simply asking people about teams they'd been on often got them to reflect on their own careers, and on their own impact on the people around them. And to our surprise, this seemed to be consistent with everyone involved in every level of software development: from the junior programmer who'd been out of school for only a year or two to the CTO who had worked his way up through the ranks from being a junior programmer years before.

Why These Contributors?

We started this project with one core assumption: the more you know about how different people run their projects, the better equipped you are to run your own. We started out by going to people who we knew personally, from our own teams and people we'd met at conferences and from being around the software development world. We were surprised at the diversity of opinions just among the people who we knew. We were also surprised that we didn't agree with everything they said—and that when we did, we were still able to see the truth in it.

That's when we came up with a cohesive goal for the book: to draw as many different opinions from as many different areas of software development as we could, regardless of whether or not they meshed cleanly with our own or even with each other. Yes, this would almost certainly mean that there would be contradictory ideas. But we decided that not only is that OK, but in fact it's a benefit. (It's a feature, not a bug.) It showed us that even though ideas are contradictory, they still work in different situations. All projects are unique, all teams are different. It would be very odd if there really were a single answer. There's room for all of these opinions, and there are situations in which any of them can be the right way to go.

Now, that's not to say that there are no wrong answers. In fact, just the opposite is true, which is why we included stories in this book about the teams that are *not* successful. Just as the right decision at the right time can make all the difference in a successful project, sometimes people can be truly misguided when they build software. They choose the wrong practice or path: for example, good documentation can turn bad when it's onerous and not used. But projects can fall off the other side of the spectrum, too, where people on the team don't plan at all, and everyone goes in a different direction. And that's what the cautionary stories in this book demonstrate: teams that started out with the wrong goal, included the wrong people, applied the wrong practices, or simply hit an insurmountable obstacle.

We could never have written a book that does that by ourselves. We're very limited in what we know. Everybody is. And that's why we cast as wide a net as possible, talking to people whose opinions we respect. The many people who contributed to this book represent the real spectrum of projects that people across the whole industry take on. And while many of the opinions and ideas that you'll read about differ from our own—in some cases, they're completely opposite—we learned a lot from each and every one of these contributors. We think that you will, too.

This book includes stories and interviews with veteran team leaders from all around the software industry. We recruited contributors from as many different industries and areas of interest as possible: from defense to social organizing, from academic research to video game development, from aerospace and defense to search engines, and from project managers to "boots-on-the-ground" programmers and system admins. There are people who we met over the course of our educations and work lives. There are contributors from a wide range of companies, including people who worked (and, in some cases, still work) at NASA, Google, IBM, and Microsoft. We were especially surprised and pleased when we got contributions from people like Grady Booch, Barry Boehm, Steve McConnell, and Karl Wiegers, whose writing and teaching were central to our own understanding of how software is built. We felt honored to work with them and other people in this book who you may not know as well, but who are also doing amazing and innovative things for software development. We're especially grateful for contributions from Tim O'Reilly (whose publishing company printed this and our other books), Scott Berkun (who we've known for years and who not only contributed a great essay but also interviewed Steve McConnell), and Tony Visconti (a legendary music producer who took the time to talk to us about his own process).

Frankly, we're still amazed that we were able to get such a wide-ranging, knowledgeable, distinguished, and talented group of contributors. But even more surprisingly, not a single contributor asked to be paid. Instead of dividing the royalties from this book among the contributors, we have the privilege of donating them to PlayPumps International, an innovative charity that digs wells to deliver clean drinking water to schools and villages in rural sub-Saharan Africa. PlayPumps is more than a charity; they've had to do their own share of engineering and innovation. You'll learn more about them and their mission (and something about teams, as well!) in our interview with Trevor Field, the founder of PlayPumps.

Every one of these contributors has something interesting, important, and, most significantly, *useful* to say about teams: how they work, how to build them, and how they break down. Each of them is a veteran team leader in his or her own right, with successes and failures under his or her belt. In some cases, we were surprised and even shocked by the stories they had to tell. And every single one is entertaining. The sum of all of these parts amounts to an experience that no single person could cobble together.

Before we delve into the four areas we chose to examine—people, goals, practices, and obstacles—we'd like to start with our interview with Tim O'Reilly because he touches on themes that you'll see over and over again throughout the book. As you're reading it, keep an eye out for each of those four things. When we talked to Tim, he was able to make a lot of these concepts concrete for us, putting words to a lot of the ideas that had been swirling around in our heads after editing the stories and interviews in the book.

We weren't really sure what to expect when we interviewed Tim O'Reilly. Between the two of us, Jenny had met him only once, at the Amazon Developer Days conference (where she'd been invited to speak), but we'd read interviews with him, and his writing on O'Reilly Radar. And having worked with O'Reilly for years as authors, we've had a lot of contact with the culture that he created there and the teams that he's built. Everyone who's been around the industry for any time knows that he's had much more of an impact on the field than someone who simply publishes books. By most accounts, he coined the phrase "Web 2.0," and he was a seminal figure in the early days of the open source movement. We were particularly interested in what he had to say about leadership and how to direct groups of people without micromanaging them. Both of us were surprised—pleasantly—to find that he told us things about teams and management that neither of us had heard before, and we found ourselves thinking about them a little differently after we spoke with him.

So if you picked up this book hoping to find the *One Correct Way*™ to run a beautiful team, we're really sorry, because that's not what this book is about. But if you're looking to gain some insight into what makes a good team tick, and what you can do to take a mediocre team and make them better—or take a great team and make them crash and burn—you're going to get a lot out of this book. We think that it's a good read, and at times a gripping one. It represents a wealth of experience, from a very wide range of people all around the industry (and a few who aren't in our industry at all). It attempts to find the general things that hang teams together, without giving you prescription, dogma, theory, or overt advice. We hope you enjoy reading *Beautiful Teams* as much as we've enjoyed bringing it together.

ANDREW STELLMAN AND JENNIFER GREENE
MARCH 2009