The Pragmatic Bookshelf

PragPub

The First Iteration

IN THIS ISSUE

* Adam Goucher reveals the heuristics of automated testing
* Jeff Cohen says community development needs a handbook
* Venkat Subramaniam talks about Scala's Sensible Typing
* Tim Ottinger and Jeff Langr take on distributed development
* Dan Wohlbruck explores the invention of the adding machine

Issue #28
October 2011

Contents

FEATURES

DEPARTMENTS

Published monthly in PDF, mobi, and epub formats by The Pragmatic Programmers, LLC, Dallas, TX, and Raleigh, NC. E-Mail support@pragprog.com, phone +1-800-699-7764. The editor is Michael Swaine (michael@pragprog.com). Visit us at http://pragprog.com for the lowdown on our books, screencasts, training, forums, and more.

ISSN: 1948-3562

Up Front

It's PragProWriMo Time!

by Michael Swaine

We're counting down to our annual orgy of authorship, when we invite you to write that book.

Lots of good stuff this month. Adam Goucher on automation, Jeff Cohen on developer community values, Tim Ottinger and Jeff Langr on distributed computing, and Dan Wohlbruck on tech history. Venkat Subramaniam continues his series on the Scala language. John Shade has a little list. And in the latest installment of our series of staff profiles, you get to meet another member of our team.

You'll find something of interest in this issue, I'm sure. But now let me tell you about a little program we've got going on *next* month.

We call it PragProWriMo, short for Pragmatic Programmers Writing Month, and this November will be our third PragProWriMo. We were inspired by (we ripped off) NaNoWriMo [U1], National Novel Writing Month.

PragProWriMo is all about helping you write that technical book you'd really like to write but for some reason haven't been able to get started.

We'll provide supporting materials and encouragement and all you have to do is to write 60 pages toward that book during the month of November.

To help you along, we're setting up a forum and a Twitter account. Follow us on Twitter at @pragprowrimo [U2] to stay up to date. Join the forum at forums.pragprog.com/forums/190 [U3] for more detailed writing advice, answers to your writing questions, and progress reports from participants. And when you finish your 60 pages, you might even get some special recognition from us.

And of course we'd love for you to submit a proposal for that book to us. But it's your work. You can publish it for free, you can do print on demand, you can hide it from the world and keep it to yourself, or you can take it to another publisher. What we're really trying to do is to help you write the book you've always wanted to write.

External resources referenced in this article:

[U1] http://www.nanowrimo.org/

[U2] http://twitter.com/#!/pragprowrimo

[U3] http://forums.pragprog.com/forums/235

But We Have These Distributed Folks

Can Distributed Teams Be Effective

by Tim Ottinger, Jeff Langr

Have you worked on a distributed team where management apparently thought it should hobble local members to make everybody equally frustrated and ineffective?

Last month [U1], we discussed appropriate reasons for acquiring an agile project management tool. One common reason is to help consolidate project information for environments with distributed team members. Our controversial take is that you'd have one less reason to invest in an agile PM tool if you had no distributed team members.

Quoting the Agile Manifesto [U2] again this month, we reiterate that its very first value is of individuals and interactions over tools and processes, with supporting principles [U3] that tell us what the agile signatories really meant:

1. Business people and developers must work together daily throughout the project.
2. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

There have been attempts, possibly misguided, to create a "distributed agile" manifesto. The problem is, the proposed new principles or values contradict this value and the corresponding principles. A revision that violates the intent of the original is a poor extension, indeed.

Developing in distributed teams is an issue close to our hearts. Both of us have worked as remote developers, including being full-time remote pair-programmers. We have worked with scores of teams that included distributed members. We have seen distributed development work, and have seen how it can fail, and we are devoting this month's article to an eyes-open look at agile tooling and distributed teams.

Distributed Teams Are a Choice

The choice to create a team that is not co-located will challenge the team's ability to follow the above values and principles. This departure from agile in turn detracts from the team's ability to be successful—unless they can find a way to return the emphasis to individuals communicating face-to-face on a daily basis.

Distributed teams are a choice, not a circumstance forced upon a company by chance. A company may find that they can attract professionals of a higher calibre if they don't insist that those professionals pull up roots and move. Likewise, a company may have chosen an implementation technology that does not have a rich user base in their local area. These are decisions that value people and interactions.

In other cases, executives buy into the promise of reducing costs by employing lower-priced and perhaps lower-capability programmers through off-shoring. *CIO*'s article "The Hidden Costs of Offshoring [U4]," by Stephanie Overby

makes it clear that off-shoring must be viewed as "a long-term investment with long-term payback." It's not as simple a matter as finding talent elsewhere—it takes a long time and continual effort to succeed with distributed teams. Executives must be willing to invest in additional travel and technology. They must also demonstrate patience in the early stages, and create a culture that accommodates many hiccups.

Whether the draw is greater talent or lower cost, distributed development incurs some intangible costs. One significant difference with distributed development is that remote folks are usually considered peripheral to the team. Since they're not physically present, they miss out on important ad-hoc conversations. Teams may view them as an annoyance:

Jeff: "Nuts! We forgot to start the phone bridge to include Curt."

Tim: "Oh! ... Oh well."

Jeff: "Oh well."

There is no question that the potential impact of Curt is diminished because he's remote. Curt knows this too, and often feels distanced from the team as much emotionally and mentally as physically.

We worked with a team that had a business analyst in Sydney, a project manager in Dallas, two developers and a tester in Dallas, two more developers and two testers in Krakow, and another developer in Bangalore. The time zone issues alone meant that it was impossible to hold a meeting with all team members without encroaching on the sleep habits of at least one participant. When there was a question, it could take 12 to 24 hours to get an answer.

If you are in a sizable organization with distributed teams, the question to ask is, "What can we do to minimize the isolation of team members and maximize face-to-face conversation?"

The best answer? Don't do that. In larger organizations, it's usually possible to create entire project teams in many of your offshore locations, particularly if you're willing to send out your expertise to help remotely grow the capabilities and culture you need to succeed. We hear "we can't do that" far more often than we actually believe it.

Employing remote developers in order to save on costs is not a necessary condition of development, but instead a choice. This choice not only diminishes from an optimal communication environment, but also emphasizes negotiation and contracts instead of incremental and iterative exploration.

Employing remote developers in order to retain a stellar performer is also a choice. Our take: We'd rather have an enthusiastic, co-located team with an average performer in place of the headaches associated with dealing with a remote "rock star."

We produced a fairly popular card [U5] to help organizations cope with having distributed teams, presented here for your consideration.

Rules for Distributed Teams

- Don't
- Don't treat remotes as if they were local
- Don't treat locals as if they were remote
- Latitude hurts, but longitude kills
- Don't always be remote

The discussion that accompanies the original card stands alone, and is worth a quick read, but since we've gotten to this discussion through a discussion of tool purchases, let's see how these principles guide us in the selection and deployment of agile project management tools.

Don't

Try pushing distributed management tool purchases off to the last responsible moment. First consider all other tools: development tools, distributed version control, continuous integration tools, remote desktop sharing, voice-and-video tools, shared whiteboards, telepresence devices, conference systems, testing hardware, and so on. If the need for an agile management tool outweighs all of these, then buying an agile management tool is reasonable. Be sure that the need is immediate and significant, the use of the tool is a good long-term decision, and that it does not further burden the remotes.

As the zen of Python [U6] says, "never is often better than right now."

Don't Treat Remotes As If They Were Local

Beware how you firewall in your teams. If the remotes can't access email, intranet resources, file shares, version control, messaging, and the like then you won't have much fun trying to collaborate with them. Locking out tools like Skype may be a bad idea.

If you choose any collaboration or documenation tools that present themselves as file systems or shares, will the remotes have ready access to them?

Remember that VPN is "the next best thing to being there" but also that it isn't the most efficient way to communicate—it's a distant second place. Often even simple screen sharing apps will have a half-second or more of lag, so that typing a word or editing a line of code can be tedious.

Don't Treat Locals As If They Were Remote

Well-meaning companies use various communication and coordination tools to help the remotes, and then make the locals use those tools as well. Local developers don't need such tools because they can more easily and fluidly communicate face-to-face. The goal of tools (management, bug tracking, or

whatever) is not to give all members of the team common hardships, but to empower them to do work well.

Remote developers know that their team experience is degraded, and expect some sacrifice to be a part of the bargain. They learn to communicate with the local developers better, to keep back channels of communication open, and to deal with lag and network droppage. Constraining locals to the hindrances of being remote only serves to diminish productivity.

Latitude Hurts, But Longitude Kills

Most agile management tools are not affected by time zones, but a communication among widely-distributed team members may routinely be an overnight thing. Expect that your emails will be answered tomorrow, and that any status updates to the management system will not be immediate like it will with local people. Expect this delay to be a part of the daily routine.

For God's sake, don't check in code that breaks the build right before going home. Knowing the time zone difference, be sure that you leave your work day with a clean build that is ready for the remote team to use. Failure to observe this warning has caused teams to develop disdainful views of each other. Be wise, be warned.

Don't Always Be Remote

A team whose members understand each other and can work together well is a successful team.

If you are just starting an agile effort, start with everyone in one place. If you have remote team members, fly them in, and insist everyone situate in a single room to feel each other out. Not literally of course—that's the stuff of lawsuits—but ensure that they come to understand each others' motivations, cultures, and quirks. Only then should you fling them back to remote corners of the earth.

To kick the team off initially, a week is not enough; you'll want them in close quarters for at least a few weeks—ideally a month to six weeks. If you have the budget, then invite spouses as well, since locals don't have to leave their families to go to work and it may be helpful for families to experience the local culture.

Sound costly? It is, but it's less costly than a non-team unable to deliver quality software because they can't communicate and don't understand one another.

You'll want your distributed team to occasionally reacquaint themselves with each other, perhaps for a week or two at a time. Twice a year is a reasonable goal.

Tools That Work

Our tools must lead us toward agile values and principles, not away from them. If a tool detracts from our ability to communicate face-to-face with our teammates on a daily basis, it is not a good agile tool. Conversely, we seek tools that help us make up for any loss in daily face-to-face communication that we might suffer due to being distributed.

"Face-to-face" means just that: When we can converse with a visible individual, observe their facial expressions, hear the tone in their voice, and read their body language, we have the best opportunity for understanding them. Email strips all of these important facets of communication. Text messaging returns only the ability to converse. Voice chat allows us to hear tone of voice, but hides body language and facial expressions that inform our reactions and interpretation. Was the remote person being sarcastic or enthusiastic? Was it a joke? Was the last remark condescending or just ill-phrased?

Audio/video chat begins to return us to the realm of face-to-face. The camera lens is limited in scope, however, and can still prevent us from seeing body language. It's also usually fixed in focus, preventing us from following a teammate into the hallway, or from seeing that someone else just entered the room.

Here are some specific guidelines for communicating within a distributed team:

1. Everyone has a pretty good camera and microphone or headset available on their computer.
2. Everyone is available on a single, standardized team chat channel. At GeoLearning, some developers used Jabber, some used Skype, many didn't log into their chat clients until reminded, and some weren't on chat at all. It was very frustrating when you needed a question answered but couldn't find anyone to ask.
3. Everyone is automatically logged into chat.
4. The team area hosts at least two always-on cameras, positioned to cover as much of the workspace as possible.
5. The team area hosts a high-quality conferencing microphone that is also always on.
6. The team either focuses another always-on camera at the card wall, or uses online distributed card wall software.
7. The team uses a scribe to capture and record important conversations when not everyone can be present (perhaps due to time zone issues). Conversation records must be broadcast to all team members.
8. The team creates and follows a process that makes it clear when and how each distributed conversation gets closed out. Otherwise, distributed conversations can easily drag on without resolution.

Conclusion

There is no reason that a company with distributed membership cannot be agile (Industrial Logic is largely a distributed organization) but it is considerably more work. Distance makes communication both less natural and more important. Misunderstanding is more likely. It is harder for local teams and local team members to appreciate the contributions of remote teams.

You can of course succeed with distributed team members (and we have!), but we ask that you recognize that the loss of free-form conversation compromises the agile ideal. We don't care whether this means you can no longer call yourselves "agile"—that's not what's important. What's important is that you

understand the values you are controverting, so that you can seek remedies and help improve your chances of success.

About Tim

Tim Ottinger is the originator and co-author of Agile in a Flash [U7], a contributor to Clean Code, and a 30-year (plus) software developer. Tim is a senior consultant with Industrial Logic where he helps transform teams and organizations through education, process consulting, and technical practices coaching. He is an incessant blogger and incorrigible punster. He still writes code, and he likes it.

About Jeff

Jeff Langr has been happily building software for three decades. In addition to co-authoring Agile in a Flash [U8] with Tim, he's written over 100 articles on software development and a couple books, *Agile Java* and *Essential Java Style*, and contributed to Uncle Bob's *Clean Code*. Jeff runs the consulting and training company Langr Software Solutions from Colorado Springs.

Send the authors your feedback [U9] or discuss the article in the magazine forum [U10].

External resources referenced in this article:

[U1] http://pragprog.com/magazines/2011-09/the-only-agile-tools-youll-ever-need

[U2] http://agilemanifesto.org/

[U3] http://agileinaflash.blogspot.com/2009/08/12-principles-for-agile-software.html

[U4] http://www.cio.com/article/29654/The_Hidden_Costs_of_Offshore_Outsourcing

[U5] http://agileinaflash.blogspot.com/2011/04/rules-for-distributed-teams.html

[U6] http://www.python.org/dev/peps/pep-0020/

[U7] http://www.pragprog.com/refer/pragpub28/titles/olag/Agile-in-a-flash

[U8] http://www.pragprog.com/refer/pragpub28/titles/olag/Agile-in-a-flash

[U9] mailto:michael@pragprog.com?subject=agile

[U10] http://forums.pragprog.com/forums/134