Cutter IT Journal

The Journal of
Information Technology Management

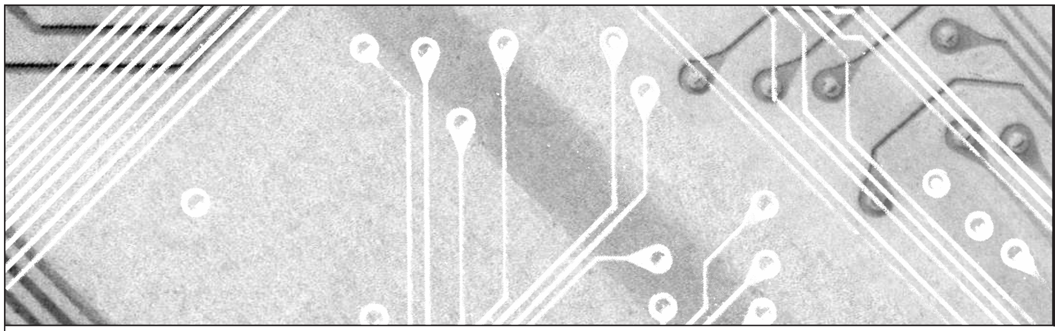
Vol. 27, No. 10
October 2014

"As a coach, it can be frustrating to hear someone want to stop the conversation, which is what invoking the 'real world' often does. Yet it's also an indication that the person or group has other issues that are causing them to lose sleep."

— Dave Rooney,
Guest Editor

Agile in the Real World

CUTTER
CONSORTIUM



Cutter
IT Journal

About Cutter IT Journal

Part of Cutter Consortium's mission is to foster debate and dialogue on the business technology issues challenging enterprises today, helping organizations leverage IT for competitive advantage and business success. Cutter's philosophy is that most of the issues that managers face are complex enough to merit examination that goes beyond simple pronouncements. Founded in 1987 as *American Programmer* by Ed Yourdon, *Cutter IT Journal* is one of Cutter's key venues for debate.

The monthly *Cutter IT Journal* and its companion *Cutter IT Advisor* offer a variety of perspectives on the issues you're dealing with today. Armed with opinion, data, and advice, you'll be able to make the best decisions, employ the best practices, and choose the right strategies for your organization.

Unlike academic journals, *Cutter IT Journal* doesn't water down or delay its coverage of timely issues with lengthy peer reviews. Each month, our expert Guest Editor delivers articles by internationally known IT practitioners that include case studies, research findings, and experience-based opinion on the IT topics enterprises face today — not issues you were dealing with six months ago, or those that are so esoteric you might not ever need to learn from others' experiences. No other journal brings together so many cutting-edge thinkers or lets them speak so bluntly.

Cutter IT Journal subscribers consider the *Journal* a "consultancy in print" and liken each month's issue to the impassioned debates they participate in at the end of a day at a conference.

Every facet of IT — application integration, security, portfolio management, and testing, to name a few — plays a role in the success or failure of your organization's IT efforts. Only *Cutter IT Journal* and *Cutter IT Advisor* deliver a comprehensive treatment of these critical issues and help you make informed decisions about the strategies that can improve IT's performance.

Cutter IT Journal is unique in that it is written by IT professionals — people like you who face the same challenges and are under the same pressures to get the job done. *Cutter IT Journal* brings you frank, honest accounts of what works, what doesn't, and why.

Put your IT concerns in a business context. Discover the best ways to pitch new ideas to executive management. Ensure the success of your IT organization in an economy that encourages outsourcing and intense international competition. Avoid the common pitfalls and work smarter while under tighter constraints. You'll learn how to do all this and more when you subscribe to *Cutter IT Journal*.

Cutter IT Journal®

*Cutter Business Technology Council:
Rob Austin, Ron Blitstein, Tom DeMarco, Lynne Ellyn, Israel Gat, Vince Kellen, Tim Lister, Lou Mazzucchelli, Ken Orr, and Robert D. Scott*

*Editor Emeritus: Ed Yourdon
Publisher: Karen Fine Coburn
Group Publisher: Chris Generali
Managing Editor: Karen Pasley
Production Editor: Linda M. Dias
Client Services: service@cutter.com*

Cutter IT Journal® is published 12 times a year by Cutter Information LLC, 37 Broadway, Suite 1, Arlington, MA 02474-5552, USA (Tel: +1 781 648 8700; Fax: +1 781 648 8707; Email: citjeditorial@cutter.com; Website: www.cutter.com; Twitter: @cuttertweets; Facebook: Cutter Consortium). Print ISSN: 1522-7383; online/electronic ISSN: 1554-5946.

Opening Statement

by Dave Rooney, Guest Editor

As a consultant and an Agile coach, I've had the opportunity to work with many different clients and speak to many people about Agile methods. From my earliest Agile experiences in 2000 to the present day, I've encountered a common statement made by those who haven't been part of teams working in an Agile manner, and even from some who have. The phrasing always contains the words "in the real world."

For example, "Agile is great in theory, but I can't see it working in the real world."

Or how about, "Test-driven development sounds great, but it's impractical in the real world."

Then there's, "Having each team member dedicated 100% to the team would be wonderful, but in the real world, we have to live with partial allocations of time."

The "real world." What does that really mean? In the cases where I've heard the term, I think that you could substitute the phrase "in my experience" or "in this organization in its current state." As a coach, it can be frustrating to hear someone want to stop the conversation, which is what invoking the "real world" often does. Yet it's also an indication that the person or group has other issues that are causing them to lose sleep.

When I first connected with others in the Agile space over a decade ago, there was another common thread among most of us. We almost all saw the practices of Scrum and Extreme Programming (XP) and recognized them from previous experience as "things that worked well." In 1992, I personally worked on a team that was colocated with its customer, worked in small increments, and solicited feedback on the work early and often. I would consider that group successful, meeting customer needs in a timely manner even when those needs changed. The group was able to consistently do this over the long term, which certainly fits a loose definition of Agile.

Since Agile didn't exist then, and we didn't call our process anything or have any certifications, how could it possibly have been successful? Joking aside, this is the crux of the matter:

Teams and organizations are successfully delivering software systems and products in this so-called real world.

In early 2001, Cutter Senior Consultant Kent Beck was interviewed by InformIT for his book with Martin Fowler, *Planning Extreme Programming*. In the interview, Beck shed some light on the origins of XP, one of the various "flavors" of Agile methods:

> The first time I was asked to lead a team, I asked them to do a little bit of the things I thought were sensible, like testing and reviews. The second time there was a lot more on the line. I thought, "Damn the torpedoes, at least this will make a good article," [and] asked the team to crank up all the knobs to 10 on the things I thought were essential and leave out everything else.[1]

When I first connected with others in the Agile space over a decade ago, there was another common thread among most of us. We almost all saw the practices of Scrum and XP and recognized them from previous experience as "things that worked well."

Did you notice how he phrased that? "I asked them to do a little bit of the things I thought were sensible." He became the lead of the Chrysler Comprehensive Compensation (C3) team in March 1996 and leveraged practices that had worked well for him in the past. His only "tweak" was to determine ways to take those practices to the "extreme" in order to maximize their benefit.

Obviously, then, people were building software effectively before XP and Scrum and other Agile methods arrived on the scene, and they are still doing so today.

IN THIS ISSUE

In our opening article, Glenn Waters describes "the best project ever" from 1987 — a testament to the fact that agility existed long before there was a term for it. Waters points out how his team had all the necessary ingredients as well as the proper environment in which to have a real team come together and build a system that delighted its customers. Their work was visible, which allowed management to take a lighter approach than normal, and they often had to use their own software in practice, prompting empathy with their customers. Not only was the project successful, the product that this small team created became the basis for its own startup company.

People were building software effectively before XP and Scrum and other Agile methods arrived on the scene, and they are still doing so today.

Next, Steffan Surdek describes several similar situations in his article, "Near-Agile Software Development Before Agile." Surdek illustrates how the teams with which he worked agreed on using certain practices (and sticking with them), which contributed to their success. He also relates these pre-Agile practices to specific principles from the Agile Manifesto, showing how they paved the way to agility.

In "Agile Team 0: The Journey," Charles Rodriguez tells us the story of a group within an organization that was forced by circumstances to embrace agility in order to have any success delivering their work. Not only did this group not have any Agile training, they also hadn't read any of the literature. Nor did they aspire to do anything other than stem the tide of defects in their system and eventually provide enhancements. But through the combination of a clear vision, committed people, and colocation, this small group "accomplished the impossible." Rodriguez shows how, when faced with dire situations like this, many organizations move toward agility out of necessity without even realizing it.

"Context is the key!" is the theme of Santiago Matalonga's article, in which he discusses how a team's and an organization's context affects the way Agile practices are implemented. Matalonga argues that agility is a continuum rather than a discrete state and suggests that groups can be Agile without living in the "sweet spot" of colocated teams composed of generalists who are dedicated to one project or product for 100% of their time. In five real-world case studies, he details various techniques used to overcome what are considered barriers to agility and that allowed those teams to effectively deliver software.

Our final article, by Jim Benson, brings in a central aspect of Lean thinking: *inspect and adapt*. Benson demonstrates how that one particular tenet, which is so critical, is often ignored by teams using a so-called Agile approach. In one of his three real-world stories, Benson examines the common Agile metric of velocity, showing how it can be misused when it is considered the one-and-only metric for Agile teams when, in fact, others that are more telling of a team's true state are available. He concludes with the advice that you must "adapt or die." Any process that doesn't provide a means to inspect and adapt — and any team that doesn't have the willingness to do so — will not be successful in the long term.

AGILE METHODS ARE A STARTING POINT

The recurring theme in this issue is that any predefined method such as Scrum, XP, or Kanban is simply a convenient place to start your team's or your organization's journey. All of the stories told in these articles highlight what Benson says — you must adapt or "die." Those adaptations may or may not be part of what's generally accepted to be Agile, but as Matalonga tells us, that's how adapting to one's context comes into play. In their articles, Waters, Surdek, and Rodriguez all describe how the teams they were a part of spent time and effort to adjust how they worked in order to improve.

This issue also highlights that effective teams have existed long before the Agile Manifesto was written in 2001. In addition, teams are delivering software effectively without using stated Agile methods right now! In the vast majority of those cases, those teams exhibit the characteristics of what we know as Agile methods.

The difference is that they're doing so "in the real world."

ENDNOTE

[1]"Interview with Kent Beck and Martin Fowler." InformIT, 23 March 2001 (www.informit.com/articles/article.aspx?p=20972).

Dave Rooney is a Senior Consultant with Cutter Consortium's Agile Product & Project Management *practice. He is a veteran Agile coach and software developer with more than 30 years' experience. As a well-recognized member of the global Agile community since 2000, Mr. Rooney helps a wide variety of organizations — large and small, private and public sector, startups and* Fortune *15 companies — improve their software delivery process. He has a deep interest in the human factors involved in the effective delivery of software that delights customers and follows practices that make the work environment not just more humane, but also fun. Mr. Rooney is an active writer, speaker, and advocate of Agile methods. He can be reached at drooney@cutter.com.*

The Best Project Ever

by Glenn Waters

This is a story about developing software, getting things done, and delighting customers. It turns out that the heart of the story is one that embodies Agile principles.

CONTEXT

In 1987, when the Internet was in its infancy, the company I was working for needed a way to manage its own network. By "manage," I mean monitor its real-time state (up, down, troubled, etc.) and gather data for performance reports.

What made this an interesting challenge is that it was, by all accounts, the largest company-run internet in the world. At that time, network management tools were quite new and not up to the job of managing such a large network.

Also unique was the number of Network Operations Centers (NOCs), the centers where network management and operational functions took place. There were more than 20 NOCs, in various locations around the world, and each one required access to the same network management tools and network status as the other NOCs.

Our approach to solving these unique challenges was to form a small software development team. The team consisted of four "developers." I say "developers" in quotes because even though we were all software developers by training, each of us performed many tasks on the project, from requirements gathering to software design, testing, customer documentation, distribution, deployment, and support.

Fortunately, our customer and user base were highly accessible. They were all internal to the company and keen to help us understand their needs in operating a network of this size.

OUR "PROCESS"

Understanding Needs in Context

Access to our customers was not nearly enough to gather requirements, however. This is because customers will ask for the most interesting features that solve their immediate problem in their context. As Henry Ford once said:

> If I'd asked my customers what they wanted, they would have said a faster horse.

We helped our customers with the broader context so that we would not end up building them "faster horses." This process involved working through problems by bringing differing perspectives to the discussion. These many perspectives, collaboratively gathered, helped everyone to get to the true core of the problem we needed to solve and helped us to focus on simple, but innovative, solutions.

Another approach we used to deeply understand where to take the product was to spend time working as network operators. An experienced network operator would shadow us as we took on their role for a day. This immersion allowed us to experience their "normal" day and to internalize the types of challenges they faced. We experienced using our software in their operational environment — which we found painful at times.

Prioritization Plus

Understanding our customers' needs was only the first step in the process, albeit one of the more challenging. As with almost any software development project, there is more work to do than can possibly be completed. We needed a way to organize our requirements.

We could simply keep a prioritized list, which we did, but it was much more complex than that. We wanted to deliver features to our customers every couple of weeks:

- **To shorten the feedback loop.** The best feedback came from having our customers use working software.
- **To receive a return on the software development investment** as soon as possible.
- **To see our code working!** Progress is a great motivator.[1]

In order to achieve the above goals, a simple prioritized list was not enough. Many features were bigger than our “every couple of weeks” delivery goal. We had to break features down into chunks that were goal-sized.

A feature would often be split into many pieces. Each of those pieces would then be prioritized against all other feature pieces. Parts of features would often end up lower in our priority list, sometimes never getting implemented.

After each delivery cycle, we repeated the above process, reprioritizing, splitting, and deleting items from our feature list. This approach to delivering features enabled us to maximize the amount of value delivered every delivery cycle and to receive constant feedback.

OUR DEVELOPMENT TEAM

A Real Team

One of the biggest factors in our success was that we had a “real” team, not just a group of people who reported to a manager. It was real in the sense that we supported each other in getting stuff done, we collaborated well, and we had trust and respect for each other.

There were four people on the team. As a small team, communication was quick and easy. However, the team was also (just) large enough to have a wide variety of skills and a diverse set of opinions. Having divergent, conflicting, opinions forced us to explore ideas deeply and ultimately drove innovation.

Rarely did one developer own a part of the system. As individuals, we had our specialty skills, but we often performed various roles and regularly worked on different parts of the system. This enabled us to:

- **Work on the highest-priority features** (rather than work on features based on available skills), thus allowing us to maximize value delivery
- **Continuously improve the skills of team members** by having people work on parts of the system unfamiliar to them
- **Improve our “lottery factor,”** so that any team member could support our system when others were away on vacation, out due to illness, or — as the phrase would have it — should happen to win the lottery and quit

Colocated

Office space consisted of the standard cube farm. We had our own office cubicles, which afforded each team member a level of privacy. Yet the cubes were located next to one another — within “chair swivel” distance — which put us into eye contact with the rest of the team members.

On the one hand, our close proximity allowed for easy collaboration. On the other hand, it made it too easy to disturb someone who was in the middle of a deep thought process. To avoid losing focus on our work, we developed hand signals that allowed us to defer and manage disruptions. While some might have viewed the signals as rude, our team did not, as we all agreed on how and when we would use such gestures.

One of the biggest factors in our success was that we had a “real” team, not just a group of people who reported to a manager.

Dedicated

Our team was dedicated to creating the network management system; we did not have other projects — well, mostly. We did have a support function for other legacy products. The support function was not significant, however, so our approach was to rotate one person on the team through the support function each week. This allowed three of the team members to be fully focused on our project, while the fourth person took on noncritical project tasks in case they were pulled away for support needs.

LIGHT-TOUCH MANAGEMENT

The project team had a traditional management structure (i.e., a manager, senior manager, director, and VP), but it was difficult for us to think of them as management since they acted more like mentors, giving us guidance and keeping us in tune with the larger corporate goals. Their management approach provided sufficient boundaries for our work while allowing our team enough freedom and sense of ownership over the project.

I recall a number of occasions when a director or VP would come around just to check up on how the team was doing, offer support, or chat about their interest in the project. While I did not recognize it at the time, this informal approach to management produced an environment with a strong sense of trust that created transparency and encouraged truthfulness.

TECHNICAL PRACTICES

For a software development project in the early 1990s, we were reasonably advanced in our technical practices. We often paired while designing, coding, and testing. This occurred perhaps a couple of hours a day, maybe every two days. Our design and code benefited greatly due to the multiple perspectives that went into the system.

Our testing was mostly manual. We did have scripts to test a couple of key processes and procedures in the system, but the scripts were not comprehensive. We would certainly have benefited from a more complete automated test suite.

Our builds were fully automated — push one button to start a build. All code was checked out from the source code control system, built, packaged, and — upon a successful build — moved to our distribution server as a "daily." Occasionally, when we had adequate new functionality and had tested it enough, we promoted a build to beta or production.

Installation was simple. Generally, one command would upgrade the system. An upgrade would only require one or two seconds of system downtime, minimizing impact to operations. Backing out of builds was just as simple and quick.

HOW AGILE WAS THAT?

I am sure you have noticed that I have not mentioned Agile terms at any point up to now. That is because there was no Agile back then — at least as a known process. However, nearly everything that our team did was Agile as it is known today. It was "real" Agile in the sense of continuous improvement and amplified learning.

Our development was incremental and iterative. We delivered something every couple of weeks — *sprints*.

We had a *product backlog*. It was prioritized, generally well split into sprint-sized chunks, and value-focused. We frequently worked with our customers to determine if we needed to add, change, or delete *backlog items*.

We had a team. We had each other's backs. We worked closely together — pairing, sharing, and learning together. We were T-shaped — each of us had our specialties, but we all did many types of work. We were colocated, dedicated to the one project, and self-organizing. We were a real team!

We limited *work in progress* (WIP), which allowed us to focus on getting few pieces of work done and done well. We followed the Lean pull principles, only starting new work when capacity existed. Limiting WIP helped us:

- Improve quality through better focus
- Increase ROI by reducing the amount of incomplete software residing in the development team
- Shorten the very important feedback loop between developer and customer

What Were We Missing?

There were a couple of practices that Agile today considers important, at least in some contexts.

We did not have a dedicated *product owner*. We all played the role of product owner, which worked well.

We did not have a *ScrumMaster* role. Our management, in a servant leadership[2] type of way, could have been considered our ScrumMaster. Management did help "clear the path" (remove impediments) to getting work done, which is a key function in the ScrumMaster role.

We didn't have the activities that Scrum defines, at least not in a formal way. In terms of the standard Scrum activities:

- Our *sprint planning* was ad hoc. So was our *backlog grooming*.
- We did not have *daily Scrums*. Instead, we all sat within earshot of each other, synchronizing and communicating frequently during the day.
- *Sprint review* (aka the "demo") took place whenever a feature was complete. Sometimes this happened on our local machines. More often, we installed the new software in a live NOC, doing a five-minute walkthrough with the operators and learning from it in a live environment.
- We didn't have *sprint retrospectives* as a defined activity, but we talked all the time about improvements we wanted to make to our process. Better still, we tried new ways of working together frequently — just "micro" changes. If we liked a change, we kept it. If not, we improved it further or discarded the change.
- We did not do *burn-up* or *burn-down charts*. Actually, we rarely estimated other than crudely for the purpose of determining where the most value could be generated from the items in our product backlog. We certainly had no formal notion of *velocity*.
- We did not use *test-driven development* (TDD) practices. From a developer perspective, TDD would have helped us. There were enough times that the system regressed, and we felt the pain. Some of those times the regressions made their way into our customer environment. Then both sides felt the pain.

- As for *acceptance test-driven development* (A-TDD), those practices would not have been of much value to us. A-TDD helps to bridge the communication gap between customer, product owner, and development team. We didn't have much of a gap, as we were all product owners.

WHY "BEST PROJECT EVER"?

This was the "best project ever" because we had fun. Lots of fun. "Work" was enjoyable. We were regularly solving problems for our customers.

In his book *Drive*,[3] Daniel Pink talks about *autonomy*, *mastery*, and *purpose* as key factors in a successful knowledge-based workplace. He writes, "People need autonomy over task (what they do), time (when they do it), team (who they do it with), and technique (how they do it)."

Autonomy drives a sense of ownership and intrinsic motivation. Our network management project had high levels of autonomy in all the senses Pink talks about.

Mastery is the ability to become better at something that matters. Mastery requires problems that are "neither too easy, nor too hard."[4] Our project was on the leading edge of a brand-new technology space and in a uniquely large environment where we were challenged to improve. We constantly strove to become better in many areas, including:

- Customer interactions
- Development tools
- Software distribution and deployment
- How we interacted with management
- How we worked as a team

Finally, Pink says, "Humans, by their nature, seek purpose, a cause greater and more enduring than themselves." Our purpose was to delight our customers. Every day we set out to make their jobs easier through using our software. Over time, our purpose grew even further, to outside of the company. We contributed our unique knowledge about scalable network management to the Internet Engineering Task Force (IETF), an international body that oversees Internet standards.

Autonomy, mastery, and purpose — we had them all. I believe that at the core of our success were these three principles, which allowed us to innovate, delight, and have fun while doing so.

EPILOGUE

For nearly 10 years we continued to develop the software. However, as time went by, commercial network management software was maturing. Eventually, the commercial software was at a state where it made more sense to purchase a system that was supported by far more people than the four developers building our system. However, that was not the end of the story for the software itself, nor for my Agile journey.

A couple of people in the company recognized the valuable software asset that we had created. They negotiated a contract with my employer to take the software to a newly formed startup. The startup continued enhancing the software and was successful in marketing it for a number of years.

Personally, I have continued working in an Agile fashion throughout my career. This has included continued success as a developer on other projects as well as in "managing" and "directing" teams. From "the best project ever" onward, the essence of the Agile Manifesto has always been in my practice and in my heart.

ENDNOTES

[1]Amabile, Teresa, and Steven Kramer. *The Progress Principle: Using Small Wins to Ignite Joy, Engagement, and Creativity at Work*. Harvard Business Review Press, 2011.

[2]Greenleaf, Robert K. *Servant Leadership: A Journey into the Nature of Legitimate Power and Greatness*. Paulist Press, 1977.

[3]Pink, Daniel H. *Drive: The Surprising Truth About What Motivates Us*. Riverhead Books, 2011.

[4]Pink (see 3).

Glenn Waters is a seasoned technology executive with over 25 years of experience in many aspects of the software industry. Mr. Waters has served as an Agile coach, helping to lead large corporations and government organizations as they adopt Agile. He has successfully worked with Agile at all levels of an organization, from helping teams improve technical and collaborative practices up to introducing the Agile mindset to CxO offices.

Mr. Waters regularly speaks at Agile gatherings, has helped organize a number of Agile conferences, and is a cofounder of the Agile Ottawa Meetup group. His training and coaching have received praise from organizations around the world. When not helping organizations, Mr. Waters is often found in the kitchen planning and preparing gourmet meals. He can be reached at glenn@westborosystems.com, Twitter: @gwww, and Linkedin: http://ca.linkedin.com/in/glennwaters/.

Near-Agile Software Development Before Agile

by Steffan Surdek

I have had many conversations with people over the last five years that have told me there is nothing really new in adopting Agile practices. These people often tell me that Agile is either merely common sense or a set of good practices people bundled together and put under an umbrella. I am here to tell you that in many ways I agree, and there is some truth to this!

In this article, I take a stroll down memory lane, looking back on many of the traditional and Agile projects that I worked on and trying to determine what made them successful. I have boiled it down to some of the key practices we used on these projects and made links between these best practices and what I teach teams every day working as an Agile coach.

By way of background, I wrote my first piece of commercial software back in the early 1990s. It was a shareware application to manage the file areas of an electronic bulletin board system. I wrote other shareware applications back then, but I have a special place in my heart for that specific program. There is nothing like fully owning an application end to end — from the visioning to the coding, testing, packaging, and marketing of it — as a way to learn about developing quality software.

Throughout my career, I have worked in small companies with relatively few developers and in much larger companies on commercial software development projects. In many of the small companies, we did not talk about “waterfall” or “Agile”; we mainly jumped in and coded whatever we needed to build. In some of the larger companies, I saw very formal waterfall development processes as well; I cannot say that these processes always helped us develop better software.

WORKING CLOSELY WITH CLIENTS

Back in my shareware development days, my main “client” was also one of my closest friends. I had fun exploring new things, and we both toyed around with the results of my work. Throughout our informal development cycles, he would regularly use, and do real-life tests of, the various applications I was working on. We had regular conversations about the glitches and annoyances he discovered, and I kept him aware of what was coming next.

In the mid-to-late 1990s, when I was developing software in small companies or working on my own projects, my team members and I were often forced to put ourselves in the shoes of our end users. Before coding, we needed to build a high-level design around the purpose of our projects. To do this, we usually brainstormed about what the applications needed to do and why. We also discussed the potential users and the general look and feel of the application. In an Agile project, the parallel would be building a vision and identifying the business objectives. In our case, putting ourselves in the shoes of our end users also meant playing with our software and asking if we would use these applications ourselves. Were they simple and user-friendly enough? Would we use them every day, or would they annoy us?

In an Agile world, we often talk about the importance of involving the client (or key stakeholders) throughout the development cycle. My work experience mainly allowed me to work with two types of clients: internal clients (as part of the IT department in the mid-to-late 1990s) and with product management teams (from 2000 to 2010).

Internal Clients

Internal clients were always happy to contribute and were interesting to collaborate with because they were as close to an end user as we could get. When the team needed information or wanted to see how our application would fit with how the clients worked, we could meet with them and show them our progress. We spent 20-40 minutes with some internal clients each day, while we only scheduled a weekly meeting with others.

The great thing about meeting these internal clients regularly was that we pretty much always delivered what they needed in the end. We had to manage scope creep and continuous requests for changes at first, but we usually found a space where we aligned around a common vision and the problem faded away.

When I compare those experiences with our internal clients to some of my experiences now in an Agile world, the main difference I see is in the formality of the role brought about by naming someone the product owner. The way the business sees the role and the impact of assigning someone full-time to the role can create resistance in some organizations. Although the Agile mindset encourages building a new partnership between the business and development sides of an organization, the hard truth is that, in some places, the negative history and distrust are difficult to overcome. In a business world constantly changing to become leaner and more competitive, freeing someone up to play the product owner role full-time is a challenge for many organizations.

External Clients

External clients were another source of feedback. When these clients needed help resolving technical issues, we would sometimes send someone to help them onsite. These trips were always great opportunities to see people using our software. Were they working around usability issues that we did not even know existed? Were they getting frustrated with certain features? Would a small change in functionality or a report make a big difference to them? In an Agile world, teams typically receive these types of comments through end-of-sprint reviews or user acceptance testing.

My experiences with product management teams in the commercial software arena varied greatly. Often there was a lack of a clear product roadmap or a general lack of vision and strategy, which made it difficult for teams to deliver the right solutions. These issues affect all software development teams, whether they are Agile or not. The main difference, though, is that an Agile project using Scrum has regular inspection points that would raise such issues much faster.

CONTINUOUS INTEGRATION

On some of my earlier projects in the 1990s, we had no real reason for continuous integration, while on others, we had no source control at all. We either compressed the source code for each release in a single file we archived somewhere on a server, or we created separate folders on a shared server for each release. Ideally, in such cases, our practice was to fix bugs in the following release and never go back to update the code of a previous version. In small projects, this worked fine, but in larger commercial software packages with larger customer bases, it did not make sense anymore. The teams I worked with used a variety of approaches to deal with this issue, as I will outline in this section.

In Agile projects, we encourage teams to do continuous delivery of software. As mentioned earlier, working closely with clients meant we needed to be able to show them our work regularly as the project progressed. When working alone, making a build is easy, but as soon as a small team begins working together on a project, some form of continuous integration needs to happen.

Although the Agile mindset encourages building a new partnership between the business and development sides of an organization, the hard truth is that, in some places, the negative history and distrust are difficult to overcome.

Early in 2000 was the first time I worked with multiple team members on a big project where we needed to integrate our code in the same code base. We had a basic agreement among team members to check in our latest functional code in the version control system every Friday. Overall this worked well, but whenever team members changed either common pieces of code or sensitive pieces of code, we would schedule conversations about the changes before integrating them.

On a life-critical application in another company, we had an integration team that manually merged in changes for each release. This implied that developers needed to isolate their changes for any bug fix or feature in branches created from the main integration stream. Developers also needed to document their changes and get them peer reviewed before the integration team integrated them into the official builds. The integration process was painful because of the overhead involved for everyone from the developers to the team integrating the work. Because of the code complexity and various governance requirements, though, the team accepted the process as an added layer of quality assurance that created accountability and provided tracking on the content of a release.

From 2007 to 2010, I had my last encounter with continuous integration as a developer with a technically gifted team. We were not using any Agile software project management practices, but we were the proverbial geeks looking for easier ways to deliver high-quality software. The version control system ran validation rules against the code that was being checked in to ensure each developer followed coding standards. We also put in place a build server that would check for changes and compile the code in the integration branch every hour. The build server automatically ran a basic suite of automated tests and would send an e-mail notice when someone broke the build with a change. The benefit of using this approach was that it allowed the team to know very quickly if any changes delivered to the integration branch broke the build, and thus we could take action to correct the problem(s). At the time, we were also working with a distributed team in India, and the automated process helped provide a form of sanity check of the build for both teams at the end of their respective workdays.

By delivering code regularly to our testers, we also avoided another trap for new Agile teams, which is delivering working code to the testers only in the dying days of a sprint.

In an Agile world, automating processes and testing allows teams to have a repeatable process and build a safety net for the quality of the product. One interesting note is that although people associate these practices with Agile teams, they are equally as effective in traditional software projects. Just as there are traditional teams that use these practices but nothing else, many Agile teams only do Agile planning and do not use technical practices such as automated testing and continuous integration.

TESTING AND FIXING ALL THROUGH THE RELEASE

Because of my early experience developing shareware, doing thorough and continuous testing as well as "eating my own dog food" have always been key practices for me in software development. Back then, I did the unit and usability testing myself and provided regular builds to one of my friends to get his comments and find more defects.

When I worked in larger companies with internal clients or testers, the two keys to testing during the development process were putting in place some form of continuous integration and properly guiding testers toward what they should be testing. For example, if we had a screen to manage users, we might deliver everything they needed to add a new user, but they may not be able to delete or edit them right away. Delivering functionality this way allowed our testers to start their work in parallel to us and enabled us to get feedback on usability and defects earlier as well.

How often we delivered testing builds to our testers depended on the project size and the nearness of the testers. For some projects we delivered weekly builds, but on others we delivered every other day.

In an Agile world, the "definition of done" guarantees the quality of the deliverables by helping teams better understand everything they need to do to deliver completed increments of software to a production environment. You can see it as a checklist the team can use to make sure they did everything necessary before making the feature available to users. Although we did not have that on the teams I led, at a minimum we always tried to ensure that we handled and corrected defects as quickly as we could. This allowed us to keep the list of defects short and easy to manage in a spreadsheet. We rarely had defects stuck on the list for more than a couple of weeks, except for minor ones.

By delivering code regularly to our testers, we also avoided another trap for new Agile teams, which is delivering working code to the testers only in the dying days of a sprint. When this happens, some developers believe their work for the sprint is complete, but if testing is incomplete, the feature will not count as "done" at the end of the sprint.

REUSING CODE AND KEEPING IT SIMPLE

One of my favorite projects was a timesheet entry application that I worked on from 2000 to 2005. This project was a great showcase for many of the coding best practices that I had seen. In one of the years I was there, we redesigned the application from scratch to use a distributed architecture, which we also designed and developed. We were under pressure to develop this new version as quickly as possible, and this caused big challenges for the development team. We chose the approach of developing a couple of object-oriented frameworks to support our work: a server-side framework to handle the requests and a client-side one to manage the user interface.

In Agile projects, we often encourage teams to develop features an iterative and incremental way. This means we expect teams to deliver smaller but functional and useful slices of software that build on top of one another. In this project, that is how we built our framework, but we called it developing "testable activity flows." We started with a few basic abstract classes (the base of every screen), which drove an empty screen with add/remove/edit/save/undo operations. We created one of the easiest screens and wrote the code to add and save a new item. This allowed us to start populating the database, and our next increment allowed us to load existing items, select one item, edit, undo our changes, or save it back to the database.

While we were building each increment, we gave builds to our tester to start playing with so we could discover if we had defects we did not know about yet. Each increment taught us a bit more about our distributed architecture, the pieces we needed in the framework, and the hooks we needed to set up in the code to keep the code generic for each new screen.

Usually, we encourage Agile teams that are developing new features to architect and code only what they need, as they need it. That is what we instinctively did on this project. Once we got our first screen fully working, we took another screen with similar attributes and used it to confirm our design. Eventually, we selected a screen with a slightly different set of challenges and implemented it to discover how the framework could support it. Working in this way reduced our code maintenance in the long run and allowed us to evolve our early framework.

The great thing about this design was how much easier it made testing and code maintenance in the long run. Testers initially panicked because some defects appeared on many screens, but for the developers, a single fix in the framework resolved many defects.

One thing I often hear teams new to Agile complain about is the potential need for continuous refactoring, because they have a strong need to code and deliver something only once. For example, they are uncomfortable with the idea of developing only part of a screen and then cycling back to it in another sprint in order to add more functionality. Their ingrained belief is they need to do it all in one fell swoop.

For us, the framework forced discipline on the team. We tried to keep the code in the base classes as completely generic as possible, and we would call abstract or virtual functions implemented in the descendant classes to fill in the blanks when required. This approach encouraged us to refactor the code continuously to make sure we were not repeating code that either belonged in a base class or that needed to be made generic in a separate library.

Two other good Agile practices around coding are clean code and minimizing documentation. While these may not have been conscious goals of ours at the time, the project teams I worked with nevertheless used to ensure that we had clear class and method names in our code. We used design patterns to bring consistency to the code and simplify code reuse, and we also worked hard at writing methods with a single purpose that were, more often than not, no longer than 50 lines of code. These practices allowed us to make the code self-documenting, as we isolated everything and our clear method names made reading the code akin to reading a book.

Many teams I worked with also had formal code review practices in place to ensure the quality of the code. While some distributed teams used collaboration tools to help keep track of necessary changes, others did not; they only met face-to-face to discuss the needed changes. Face-to-face meetings were more effective, but they required us to have webcams for the distributed team members. At a minimum, it was important for distributed team members participating in a code review to at least be on a phone call, because using only the collaboration tool sometimes caused communication issues and delays in addressing the concerns highlighted in the tool.

CONCLUSION

When writing this article, I realized the biggest factor in our teams' successful delivery of software was that we upheld a high level of discipline in following our development practices. Without discipline and structure, your teams will struggle whether you are using Agile practices or not.

I'm always amused when I hear the phrase "Agile says [insert statement here]," because when you think about it, Agile is born from a manifesto that contains just four values and 12 principles. Neither the values nor the principles tell you exactly what to do; they simply offer some high-level guidelines to follow.

When you look at Agile practices from that point of view, it is easy to see why some people have the impression there is nothing new here, because essentially there isn't. You do not need to reinvent the wheel; you can simply determine which of the best practices you have used before will help you align your team with the principles and values contained in the Agile Manifesto.

Steffan Surdek is an Agile coach and trainer at Pyxis Technologies. As a coach, he strives to create engaged teams led by inspiring and empowering leaders for the sake of making the software development workplace fun again. Mr. Surdek has worked in IT for over 20 years in collaboration with many distributed teams around the world. He speaks at many conferences and user groups about leadership and agility with distributed teams. Mr. Surdek is coauthor of the book A Practical Guide to Distributed Scrum*, written in collaboration with the IBM Scrum Community. He blogs about life and agility at Surdek Solutions (www.surdek.ca) and about authentic leadership at Provoking Leadership (www.provokingleadership.com). He can be reached at steffan@surdek.ca.*

Agile Team 0: The Journey

by Charles C. Rodriguez

Why do some organizations lose focus on true agility and distract themselves with "going Agile"? Agile product development, along with its various frameworks, has matured over the last decade. These frameworks have offered different interpretations of the same message: the Agile Manifesto. Each framework introduces its own artifacts, ceremonies, and deliverables. Almost inevitably, these items become the obsession of teams attempting to adopt Agile. Organizations, teams, and individuals lose themselves in the mechanics of "being Agile" instead of taking the time to realize true agility.

Unknowingly, most organizations already exhibit signs of agility. Consider what happens when a company or team goes through an emergency or "fire." Employees and managers drop titles and processes, establish a collaborative zone, and focus on accomplishing their mission. For some reason, the more people attempt to interpret a simple message, like the manifesto, the more complex they make it. I have had the opportunity to work on several teams over my career. I have witnessed teams with no knowledge of the Agile frameworks flourish and deliver, and I've also witnessed teams with all the information at their fingertips flounder and struggle.

In 2006, I enjoyed the amazing experience of working as part of a team that exercised agility without reading a book or researching a framework. We didn't have a creed, process, or manifesto at our disposal, but we did have a vision; our primary purpose was to see the success of that vision. The simplicity of that goal allowed us to accomplish what seemed impossible. While we were ignorant of the Agile Manifesto and the various Agile frameworks, in our story we naturally implemented some very familiar activities.

THE TEAM

In the summer of 2006, a small development team formed for the sole purpose of maintaining and enhancing our company's invoicing engine. The application's portfolio contained over 200 tickets, consisting of defect reports and change requests. Along with two other developers, I accepted the assignment, knowing it would provide a challenge like no other in my career. In addition to the developers, two analysts and a project manager were assigned to the project. Interestingly, the assembled group did not receive its priorities from the primary stakeholders responsible for invoicing. As we will see, this disconnect between the stakeholders and the delivery team would result in poor productivity and perceived quality.

A ROCKY BEGINNING

In its first few months, the team struggled and lost ground. Our IT department ran both the development and project management team; there was little connection to our company's growing needs, and the communication was fragmented. Our priorities were set from the top down, with little input from the personnel most familiar with the application's strengths and weaknesses. Frustrated with the inability to attack the technical debt, our best developer departed in the fall of 2006. Our ticket count continued to grow, and we were not releasing code quick enough. If we were going to make any progress, we needed a change fast.

ORGANIZATIONAL CHANGE

Tired of the lack of progress, the organization made a drastic adjustment to the team's structure. An operations team that reported directly to the business formed. This team absorbed the project manager's and analysts' positions. Their goal was simple: ensure top quality of the invoices generated while maintaining production levels. The development team remained within IT, but from that point on, we would take prioritization directly from the operations team. The company hired a new operations manager to lead the movement. This new manager brought a servant-leader approach to both teams. He understood that the operations analysts and developers knew the system best, and he would balance our recommendations for prioritization along with those of the stakeholders. These two moves would serve as the catalysts for our team's long-term success.

FORMING

Both teams were given the same long-range vision: eliminate all major bugs, guarantee invoice accuracy, and support new products. One afternoon, the operations manager and I were reviewing the 200-plus tickets; we both wondered how in the world we were going to pull off a miracle and accomplish our goal. Everything was working against us: long release cycles, QA bottlenecks, and constant new product rollouts. In this meeting, we joked how our situation resembled a historical moment, the Battle of Thermopylae. The movie *300* had just been released, and though we were joking at the time, it felt like an appropriate analogy for our situation. There stood a small, skilled team facing a seemingly endless onslaught of tickets. Rushing in and attacking large chunks over a long release cycle would prove disastrous. Similar to the Spartans, who used their knowledge of the terrain to fend off waves of attackers, we decided to dig in our heels, rely on our expertise with the system, and knock out tickets in quick, controlled bursts.

STORMING

In order to succeed, we knew we had to make communication across both teams a priority. We acknowledged that open, honest communication would be the glue for collaborating on fires and designing new products. Like the Spartans, we would work as one single unit. Colocation was key; the ability to collaborate in our "war room" effectively built trust within the teams.

Our team consisted of outstanding professionals, and we knew we could count on them to follow through with the level of communication required. First, we faced the traditional release cycle; it was far too long and hindered the kind of progress we sought. The teams negotiated a break from the traditional four- to six-month release cycle; instead, we would deploy a small number of tickets weekly. In exchange for this schedule, the development and operations teams agreed to accept full responsibility for the entire development lifecycle. Everyone was responsible for all aspects of the product, which created a sense of ownership for the product. Because the operations team owned the application, they would provide the sign-off on releases. The sense of ownership drove our commitment to code quality and care for the application.

Our weekly cadence consisted of three major short — 30-minute — meetings. On Monday, the operations team would present the highest-priority tickets, and our development team would pick those we felt could be completed by week's end. Sometimes we chose tickets that would carry over into the next release because of sizing issues, but that was an acceptable risk. Wednesdays consisted of a quick status update. By then, we would know what was in danger of falling by the wayside and what would release on Friday. We also used this meeting as an opportunity to make any changes in scope to our tickets. By Friday, we were in full release mode. We tried to avoid last-second changes, but they happened often. However, working closely with the operations team guaranteed that we did not introduce unnecessary risk; they were excellent at rejecting dangerous, 11th-hour changes. By the end of the day, we were deploying some tickets and prepping others for the following week.

NORMING

We started following our strategy in November 2006, and we consistently followed it until February 2008. Operations prioritized the work, the developers coded, and we all agreed on the tickets for release at the end of every week. For the first time in the team's history, we gained ground, completing over 200 tickets in a year. The application was at its most stable, and it met the company's performance demands. In less than two years' time, our small group had accomplished the impossible. Although the amount of new work coming in didn't stop, the backlog now numbered fewer than 50 tickets.

The pace we established was brutal. It was common to work 10-plus-hour days at the office, weekends, and evenings. Despite the tenacious pace, the entire team remained steadfast in accomplishing the goal of stability and maintainability for the system. A high level of commitment existed at the heart of the team — if one team member worked late, we all stayed with him or her. On new product release weekends, team members would volunteer to pull an overnight shift to watch for any possible fallout.

There was a clear understanding of how important this application was to the organization. As a result of our success, our team was allowed to operate outside of the IT department's established rules and processes; many other teams did not enjoy such freedom. Unfortunately, that freedom didn't sit well with some, and eventually things had to change again.

THE END OF AN ERA

Like the Spartans at Thermopylae, our team could only hold out for so long. Eventually, the development team was moved away from the operations team. It's amazing how sitting in close proximity breaks down communication barriers, but now we no longer had the benefit of colocation. The frenetic pace drove two of the developers to seek new opportunities. Thus, we had to start all over again with a growing backlog and fresh faces.

With a new team came new imposed policies. We would once again follow the standard four- to six-month release cycle. Instead of one source for prioritization, the requests would come from a variety of stakeholders across the organization, and of course every request from the stakeholders was marked as highest priority. The newly assembled development team was just as skilled as the previous one, but all the factors that led the previous team to success had been removed. Momentum was lost, and our team fell into the dark void of waterfall software development.

LOOKING BACK

Eight years later, the operations manager and I were enjoying my farewell dinner with some members of the original team. Over the last few years, agility had become something the organization aspired to achieve. However, teams equipped with all the information they could possibly want about the Agile framework were struggling with mere fundamentals and debating trivial definitions rather than focusing on producing. How, in contrast, did a team with no knowledge of Agile frameworks unknowingly stumble upon so many of Agile's core values? I believe I can attribute our success to three core factors:

1. We had a clear vision.
2. We had the right personnel who were fully committed.
3. We took full advantage of being colocated.

The sense of urgency created a bond within our team that helped us avoid silly arguments over process and rules. Ironically, that sense of urgency also exposed our inability to establish a sustainable pace; our success increased the demand for results, and we didn't have the tools available to regulate that pressure.

On the night of that farewell dinner, we toasted to all we had accomplished as a small team facing a monolithic challenge. We did not win the war, but we came together and won an amazing battle. Though our team was long forgotten in the years that followed, we had pioneered agility in the organization.

CONCLUSION

Conversations with several colleagues in the software industry have yielded the same frustrations and difficulties in adopting an Agile framework. Most of the conversations revolved around switching away from a sequential product delivery model and their organization's attempts to translate the prior processes and practices into Agile frameworks. In these cases the "process" became the central focus, which completely contradicts the core of Agile, which is the manifesto:

> We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> **Individuals and interactions** over processes and tools
>
> **Working software** over comprehensive documentation
>
> **Customer collaboration** over contract negotiation
>
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more.[1]

My experience with the team in 2006 and undergoing a complete enterprise adoption of Agile years later has provided an interesting perspective. To me, the intent of Agile and its various frameworks is simple and unified: "Get people together so they can collaborate in an attempt to reach a goal." I am not advocating that we burn all the books and delete all the information available. However, why not view these resources as guidelines rather than doctrine?

As human beings, we naturally want to introduce order and process into something we don't quite understand. Yet in times of crisis, we don't give a second thought to kicking all the formalities to the curb and finding a way to bond together. If we can introduce that same sense of urgency into product development, we'll realize the true agility we already possess and abandon the false notion of "being Agile."

ENDNOTE

[1]Agile Manifesto (http://agilemanifesto.org).

Charles C. Rodriguez, cofounder of Synergetic Management Solutions, works with organizations that are attempting to make the transition from sequential to iterative product development. At Agile West 2013*, it became apparent how many organizations struggle with the adoption of Agile frameworks. After successfully leading a movement at his previous employer to switch from waterfall to Scrum, Mr. Rodriguez decided to work alongside companies to help guide them through the twists and turns of an enterprise Agile adoption. He takes a hands-on approach to learn about a company's background, current processes, and where they wish to be in the future. Along with his work in software product delivery, Mr. Rodriguez is also expanding the application of Scrum to physical fitness and wellness. In a society where clients want instant gratification, most individuals become discouraged with their attempts to achieve weight loss and become healthier. The iterative philosophy behind Agile frameworks provides an excellent medium for helping clients see their progress one "sprint" at a time. He can be reached at ccrodriguez@agilescrumdaddy.com.*

Five Process Tweaks That Won't Prevent You from Being Agile

by Santiago Matalonga

Agile methodologies have proven their worth in their almost two decades of existence. Yet just like many other technologies, they have also been misused and abused. Fervent Agile crusaders advocate their use in almost any context. These fanatics will deny teams and projects the title of "Agile" if the practices dictated by their methodology are not followed. In this article, I will argue that so long as the values of the Agile Manifesto are respected, your process can still be Agile even if it does not follow a standard Agile methodology to the letter.

WHAT IS AGILITY?

The question of "being Agile" has troubled many software developers over the last decade. In the years that have passed since the signing of the Agile Manifesto, Agile methods have gained mainstream adoption in the software industry.[1] Such success has brought crusaders who argue that Agile methods and practices are *the* way to develop software. As a result, for the best part of the last decade, we have witnessed trench warfare between agilists and traditionalists.

What was missed in these disputes is the key role that *context* plays in the lifecycle of a project. Each project within each software development organization is subject to constraints imposed by the project's reality. For instance, a problem my colleagues and I have studied is the impact of Agile practices in global software development.[2] In global software development environments, teams are distributed, cultures are different, and the customer can usually only be at one location at a time. Does this mean that these projects cannot be Agile?

This same research revealed a flaw in the widespread understanding of agile; namely, that there is no formal definition of agility. There have been a few attempts at defining the term in academia, [3-5] but each definition brings its own vision to the term. Likewise, all case studies that claim success are based on a self-proclaimed definition or understanding of agility. In theory, this is a problem for advancing empirical knowledge, since results from different case studies cannot be aggregated. In practice, it is also a problem since it is not possible to separate successful Agile cases from false positives.

Our view is that agility is not a binary concept, but rather a continuum of possibilities constrained by the context in which a software development project must operate. This view is currently supported by several academic[6, 7] and commercial[8] attempts to measure the "degree of agility." The practical implications for this outlook are that practitioners should be free to select the solutions that best fit their problems, regardless of whether those solutions come from Agile or traditional settings.

This article presents five case studies in which the project context posed some restrictions that prevented the out-of-the-box application of an Agile methodology. In each of these cases, my associates and I implemented context-sensitive tweaks to the development practices being followed in order to solve a pressing project problem while striving to maintain the values held by each organization.

FIVE REAL-WORLD AGILE ADOPTIONS

In the real-world Agile adoptions recounted here, my colleagues and I had to introduce practices that are not strictly recommended by the standard Agile methodologies. In the following case studies, I will show how we introduced these practices to mitigate a risk or constraint imposed by the project's context and argue that they did not have an impact on the project's capacity to "be Agile."

Case 1: No Onsite Customer

Context

This Uruguay-based organization tailored e-commerce solutions to clients. They had succeeded at implementing a Scrum-based Agile process for projects when the clients were located in Uruguay. Though they never achieved the practice of an onsite customer, the team had become accustomed to receiving the client's representative at key Scrum events, such as reviews and most planning sessions. Unfortunately, Uruguayan clients represented only a fraction of the organization's projects. They also had projects for nearshore clients who were left outside the scope of the initial Agile adoption. The organization's strategy for dealing with the geographic distance was to invest in improving their telecommunications infrastructure. They envisioned involving the nearshore client representative at the Scrum events via a Web-based meeting space.

Problem Statement and Risk Evaluation

The organization's first warning sign that the strategy was off course was the amount of rework coming out of review meetings. This included a fair amount of defects, change requests, and even whole stories getting rejected. This had seldom happened on projects with Uruguayan clients. The situation was decreasing the clients' satisfaction levels and reducing their trust in the development process.

Implemented Solution

After discussing alternatives, my associates and I suggested implementing a new role within the Scrum process. The "proxy product owner" (PPO) role was assigned the responsibility of looking after the customer's best interests in the Scrum events where the customer was not able to attend.

Results

The introduction of the PPO helped the flow of those Scrum events. At Scrum planning, the team had an interlocutor who could provide a view of the business. During the sprints, the PPO provided the necessary feedback on the progress of user stories. The PPO also established more fluent communication with the client representative.

Discussion

The PPO solution to the lack of an onsite customer had already been reported in the literature,[9, 10] and according to our research,[11] it is the second most-used strategy to mitigate problems associated with geographical distance. In this case study, a key factor in ensuring the success of the strategy was the selection and training of the appointed PPO. At each project, we chose individuals who had experience doing business analysis to appoint a PPO. In four one-hour sessions, we coached them on the responsibilities of this new role and its importance in the success of the project. During the coaching, which took place over a one-month period, we also determined that they needed training on how to write user stories with their acceptance criteria.

It is worth noting that the introduction of a PPO tends to weaken the bond between the team and the product owner. In a textbook Scrum implementation, a trust relationship is usually generated between the product owner and the team. The product owner trusts that the team is able to deliver the stories he or she selects to give value to the business, while the team trusts that the product owner has the ability to make that choice. The introduction of a PPO comes with the risk that he or she won't understand what value is to the product owner. In this case, though, it was a necessary tradeoff to mitigate the constraint of not having an onsite customer.

Case 2: Specialists over Generalists

Context

This organization specialized in B2B solutions using J2EE technologies. They had failed at implementing Scrum-based Agile methods in the past, reverting to previous methods of project management. Nonetheless, senior managers were confident that Agile should work for their organization, and my associates and I were asked to lead the transition.

Internally, the organization was divided into service areas. An independent testing team was in charge of providing functional testing services to the projects. Technical experts and usability/graphic design groups were two other groups the teams had to interface with. Development teams were formed on demand to serve new customers. The organization provided training for personnel within each service area, and there were virtually no cross-training events.

Problem Statement and Risk Evaluation

The problem was whether Agile project management practices were a good fit for the organization. Having failed at previous attempts to adopt Agile methods, there was certain disbelief among the organization members that they would succeed this time.

Implemented Solution

Our working hypothesis was that previous consultants were pushing Agile solutions without regard for the organization's context. So we started by evaluating the values and practices of the organization against those stated in the Agile Manifesto. To do this, we adapted a collaborative scheme we use for process appraisals,[12] tailoring it so we could appraise values and practices instead of process requirements. The results showed that the organization held no values that came into conflict with those of the Agile Manifesto, but they did have some practices that were not recommended by — and even conflicted with — some of the manifesto's 12 principles. Not surprisingly, their reliance on specialists was one of them.

Results

The aforementioned activity helped reduce resistance in the subsequent change toward Scrum-based project management. It also made clear that since some skills (e.g., testing, usability design) would remain outside the development teams' skill set, they would have to implement communication interfaces to those service providers. We designed this interface as a kanban board through which teams could signal work requests to the service areas. When we left this organization, they had successfully adopted a Scrum-based project management process.

Discussion

The appraisal process helped to explicitly uncover the organization's value of specialists. We agreed with senior management that this value was so rooted in the organizational culture that it would require considerable effort to change it. Furthermore, to the best of our knowledge, there is little or no empirical evidence in the software development realm to convince them otherwise. So the best course of action was to give them the tools to evaluate this belief for themselves.

In retrospect, we believe the success of the Agile transformation in this setting was due to our respect of the organization's values. Practices are a lower level of abstraction; they are easier to change when they conflict with each other. In contrast, when different values come into conflict, individuals must make hard choices in order to prioritize one over the other.[13] We removed this hard choice by introducing Agile practices that did not go against the organization's values.

Case 3: Interfacing with Service Areas

Context

This organization was producing software using a Scrum methodology. As in the previous case, project teams had to interface with service areas from the organization in order to access specialized skills such as quality assurance (QA) and technological support. In contrast to the previous organization, these teams were used to the Scrum process, and their main concern was to integrate the skill set from the service areas into their planning cycle.

Problem Statement and Risk Evaluation

Most of the teams we met had identified a problem of delayed feedback when interfacing with the organization's service areas. This was especially evident with QA, as it could not guarantee the allocation of personnel to test the user stories that were being finished toward the end of sprints. This resulted in unfinished stories, defects carrying over to the following sprint, and a fluctuation in the teams' velocity. Figure 1 shows an example project with initial unsustainable velocity values, due to the team's not being able to gauge their defect injection rate.

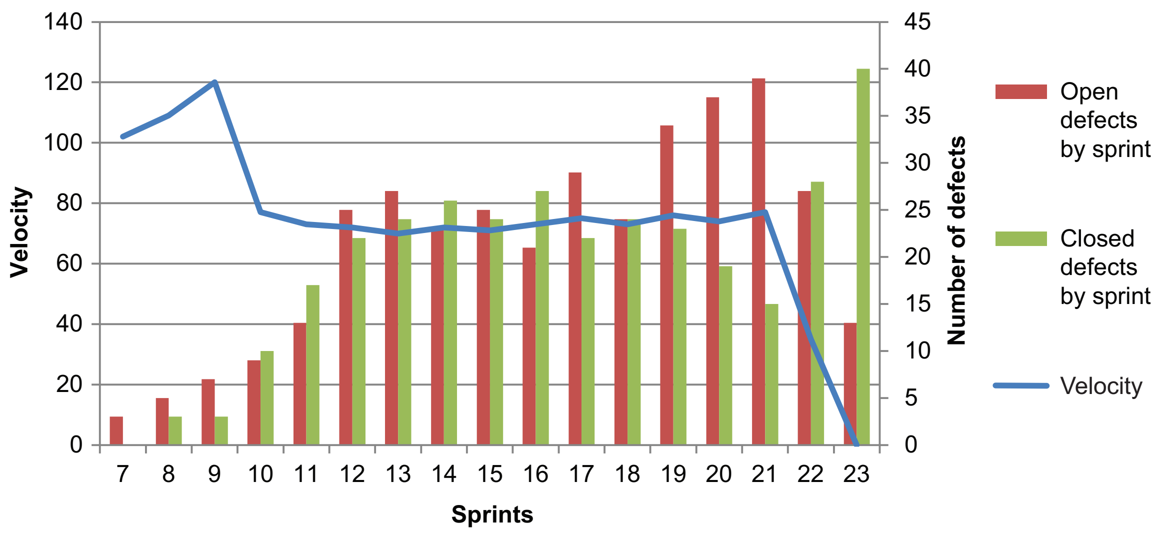


Figure 1 — Effect of defects carryover on sprint velocity.

Implemented Solution

We "gamified"[14] several retrospectives to guide the teams into discussing the effects of the problems they had identified. The emerging solution was to introduce two practices to the development process:

1. Sprint goal
2. User story prioritization

Teams were only using a velocity goal as their goal for a sprint. To distinguish their practice from the sprint goal as defined in the Scrum Guide,[15] we introduced the term "qualitative sprint goal" (QSG) to describe a product owner's high-level expectation of the value of the sprint. Achieving the QSG required the prioritization of user stories. A two-level MoSCoW[16] approach was introduced to signal which stories were key to the product owner and which ones could be carried over to the following sprint.

Results

The tweaks had the effect of stabilizing sprint velocity, thereby making the team's output more predictably aligned to the product owner's expectations.

Discussion

Though the problem of this and the previous case study seemed similar, the latter organization's maturity was a key differentiating factor. This organization had a strong Agile practice and had already been able to identify the effects of the problem with tangible data. However, they were not using retrospectives efficiently. Pushing for a kanban solution would probably have resulted in a more thorough description of the symptoms of the current problem (delayed feedback, fluctuating velocity, etc.) but done little to resolve the underlying cause (the ineffectiveness of the retrospectives).

In Scrum, retrospectives are events where process improvements can be defined and implemented. A great deal of Scrum's effectiveness is lost when teams cannot achieve a strong reflective practice during their retrospectives. Therefore, we first put focus on using the games that would train team members in performing causal analysis.[17] It was only then that we hinted at a solution that would correct the current problem; namely, clarifying the interpretation of the "sprint goal" so the team could achieve what the product owner considered to be the value of the sprint. Requirements prioritization was a natural follow-up to achieving the sprint goal practice.

Case 4: Introducing Capacity Management for Multitasking Teams

Context

This organization was building Ruby on Rails solutions for a variety of clients. Projects were short, with very few of them lasting longer than two months. Teams were usually pairs; a four-developer team was hardly ever seen at this organization. Team members were assigned to multiple projects at the same time, and some even had projects in maintenance to attend to. All teams were using variations of the Scrum practices: they organized work increments in sprints, had one planning session per sprint, and usually asked customers to sign off on working software at the end of the sprint. There was no onsite customer and hardly any retrospectives.

Problem Statement and Risk Evaluation

The team was losing faith in their Agile methodology because they were consistently failing to meet commitments. Team members were sure that the culprit was multitasking. In contrast, management thought that assigning teams to multiple projects was the only way to make the organization viable.

Implemented Solution

We worked this problem at two levels; with senior management to obtain commitment on project assignment caps, and at team level to scale Scrum event durations based on the total effort in a sprint. On the management front, we obtained commitment for a maximum of three projects per developer. On the project level, we set a maximum duration per Scrum event by using "rule of three" cross-multiplication.[18] We also implemented a *velocity by effort* metric as a measure of productivity. Team members now had to agree on which stories to work on according to the number of story points involved and the effort required to realize them. Commitments would be based on both effort and story points.

Results

The introduction of the three-project limit provided a reasonable starting point for defusing the conflict. It allowed for a healthy flow of the Scrum process within this organization. Furthermore, though we don't have sufficient data to be certain, the tweaks in the process appear to have reduced the variation of sprint velocity (see Figure 2).

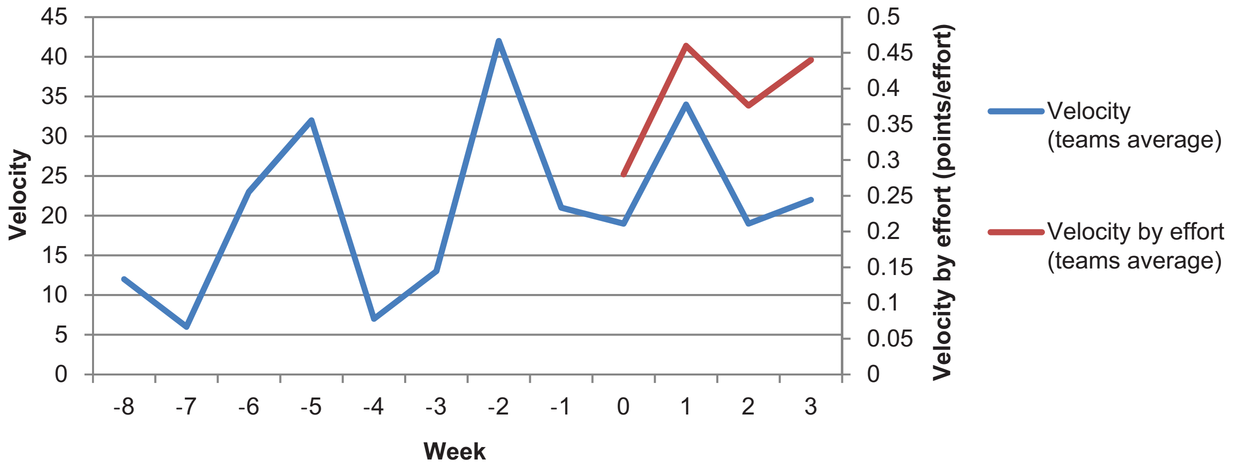


Figure 2 — Comparison of velocity and velocity by effort variation.

Discussion

At first, the issue of multitasking appeared to be an unresolvable conflict between the developers and senior management. However, the three-project cap proved to be a reasonable limit for the organization. Senior management agreed that it should not have an impact on the bottom line; it allowed them to try WIP limits and provided a good basis for bargaining with the developers. On the other side, introducing capacity management practices at the developer level helped developers realize the value of the measurement practice. They related more to velocity by effort in a sprint than they did to absolute velocity.

This case shows how a restriction that prevented the smooth flow of an Agile project was successfully circumvented. There is not a single statement in the Agile Manifesto — values or principles — that asserts only full-time developers can do Agile. Multitasking has its costs, of course, but if it is deemed necessary, this case shows that there are still ways to use Agile processes in these settings.

Case 5: The Traveling Salesman in a Distributed Scrum of Scrums

Context

This was a large organization trying a multisite Scrum of Scrums process for one of its projects. The outer Scrum was onsite with the US-based customer. That team performed QA, defect correction, and deployment of solutions that the other two Scrum teams delivered on the previous sprint. The latter development teams were based in Montevideo, three time zones away from the US team. Each Scrum team had about six developers each. Even though the process worked for a reasonable time, conflicts were starting to appear, specifically between the US team and the Uruguay-based development teams.

Problem Statement and Risk Evaluation

The inter-Scrum rivalry was not evident in the project measurements, but the organization leaders were afraid poor morale would eventually cause staff turnover and reduce customer satisfaction.

Implemented Solution

In this setting, we introduced the concept of the traveling salesman. Every other sprint, one team member would change projects so as to experience the different work environments. We hoped that traveling (especially international travel) would boost morale, while meeting face-to-face would bring the team members together and foster collaboration.

Results

The solution had the intended effect of boosting morale and getting the team members acquainted.

Discussion

The traveling salesman approach to reducing geographic distance is also one of the mitigation strategies we identified in our research.[19] In this case, in order to reduce the overhead cost of international travel, we also rotated within the colocated teams. This meant that one team member made one international trip every other week.

In this setting, communication was suffering from the effects of distance in the teams. The traveling salesman approach had the effect of getting everybody acquainted face-to-face, and most importantly, of familiarizing everyone with the problems of each team. It provided a fresh outlook on one of the manifesto's key values: to collaborate, not compete.

CONCLUSIONS AND LESSONS LEARNED

In this article, I have tried to reinforce the view that "agility" is not a state but a continuum. I believe that Agile values, practices, and methodologies have provided value to the discipline of software engineering, and some of these practices are widely applicable in a wide variety of cases (e.g., automated testing). In contrast, I am also a firm believer that each context presents its own restrictions on a software project, and these must be understood before imposing out-of-the box solutions. As the above case studies show, with an understanding of the desired outcome and a view to the context, Agile methodologies can be adapted to fit your organization's constraints. Of course, such adaptations will often have cons as well as pros that organizations will have to weigh. For example, the traveling salesman approach will enhance face-to-face communication, but it will also increase costs. It's up to the practitioners either to assume the cost of the tweak to reap the intended benefits or work to remove the restriction.

ENDNOTES

[1]Begel, Andrew, and Nachiappan Nagappan. "Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study." *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE Computer Society, 2007.

[2]Matalonga, Santiago, Martín Solari, and Gerardo Matturro. "Factors Affecting Distributed Agile Projects: A Systematic Review." *International Journal of Software Engineering and Knowledge Engineering*, Vol. 23, No. 9, November 2013.

[3]Pikkarainen, Minna. "Agile Assessment Framework." Agile VTT, Version_1.0.

[4]Boehm, Barry, and Richard Turner. *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley/Pearson Education, 2003.

[5]Leffingwell, Dean. *Scaling Software Agility*. Addison-Wesley Professional, 2007.

[6]Pikkarainen, Minna, and Ula Passoja. "An Approach for Assessing Suitability of Agile Solutions: A Case Study." *Proceedings of the International Conference of Extreme Programming and Agile Processes in Software Engineering*. Springer-Verlag, 2005.

[7]Schweigert, Tomas, et al. "Agile Maturity Model: Analysing Agile Maturity Characteristics from the SPICE Perspective." *Journal of Software: Evolution and Process*, Vol. 26, No. 5, May 2014.

[8]Kroll, Per, and William Krebs. "Introducing IBM Rational Self Check for Software Teams." *The Rational Edge*, 3 June 2008.

[9]Lee, Seiyoung, and Hwan-Seung Yong. "Distributed Agile: Project Management in a Global Environment." *Empirical Software Engineering*, Vol. 15, No. 2, April 2010.

[10]Jalali, Samireh, and Claes Wohlin. "Agile Practices in Global Software Engineering: A Systematic Map." *Proceedings of the 5th International Conference on Global Software Engineering (ICGSE '10)*. IEEE Computer Society, 2010.

[11]Matalonga et al. (see 2).

[12]Alvarez, Amalia, and Santiago Matalonga. "Process Appraisals: From Group Interviews to Socratic Circles, Ten Years of Results." Paper presented to the *6th World Conference on Software Quality*, London, UK, July 2014.

[13]Glazer, Hillel. *High Performance Operations: Leverage Compliance to Lower Costs, Increase Profits, and Gain Competitive Advantage*. FT Press, 2012.

[14]For inspiration for games to use in retrospectives, we use TastyCupcakes.org.

[15]Schwaber, Ken, and Jeff Sutherland. "The Scrum Guide." ScrumGuides.org, 2013.

[16]MoSCOW stands for "Must have," "Should have," "Could have," and "Would like to have." It is used to classify requirements according to their value or priority. In this case, user stories were classified into "Must haves" and "Should haves."

[17]Kalinowski, Marcos, David N. Card, and Guilherme H. Travassos. "Evidence-Based Guidelines to Defect Causal Analysis." *IEEE Software*, Vol. 29, No. 4, July-August 2012.

[18]"Cross-multiplication (rule of three)" (Wikipedia).

[19]Matalonga et al. (see 2).

Santiago Matalonga is a researcher at Universidad ORT Uruguay. Dr. Matalonga has a PhD in software and systems from the Universidad Politécnica de Madrid. He has several publications in international conference proceedings and journals on the subject of the ROI of software process improvement. Dr. Matalonga was a member of the team that translated CMMI v1.3 to the Spanish language. Since 2010, he has been a graded researcher at the Uruguayan national research system (SNI) and at the Uruguayan program for the development of basic science (PEDECIBA).

Dr. Matalonga has strong ties to the Uruguayan software development industry, where he has been working since 2002. In 2006, he was part of the process group team that achieved the first CMMI Level 3 rating for a Uruguayan organization. Since then, he has been involved in several process improvement initiatives in the software and service industry. Dr. Matalonga is currently providing services as an independent consultant, helping organizations achieve tangible business results by introducing process improvement practices. Some of his current projects include coaching in Agile application lifecycle management, coaching Agile teams, and assisting organizations in adopting Agile and traditional process models. He can be reached at smatalonga@uni.ort.edu.uy, Twitter: @santimatalonga.

If It's Broke, Fix It: Inspect and Adapt Is *Real* Agile

by Jim Benson

THERE IS NO "RIGHT" WAY TO BE AGILE

When the Scrum trainers of the world got together to decide how to create a Certified Scrum Trainer course, they figured they would be in and out of the planning room in one day. Instead, they found themselves engaged in months of grueling discussions. Even though there have been books written and techniques employed in the service of creating an Agile toolkit, it seems that understanding what goes into *being* Agile is actually quite complicated.

Or is it?

Whether we want to say we are "Agile" or "Lean" or "agnostic," the heart of any good process is that it truly delivers working software in a way that doesn't disappoint us. We'd like it on time, we'd like it on budget, and we'd like to not get status meeting surprises like "Oh, we were about to launch and realized the whole platform is built of confusing and brittle code" or "We were about to launch, but it turns out our security model is utterly illegal in the EU."

Agile rollouts tend to focus on how to do things: how to release faster, how to use planning poker cards, how to turn your product manager into a product owner, or how to turn your project manager into a ScrumMaster. We tend to send people out to be certified in Agile, which we believe teaches them to do Agile "right."

The bad news: there is no one right way to be Agile.

INSPECT AND ADAPT: LEAN'S CENTRAL TENET, AGILE'S FORGOTTEN GIFT

We have access to a wide range of good practices, all of which can be used in an Agile team, and all of which can be *ignored* by an Agile team. What's important to remember is that all practices are suggestions, and all practices have a range of implementation patterns. We apply these practices when we need them and in ways that they will help.

We *inspect* our current state. We *plan* a preferred future state. We *adapt* our processes to transition from our current state to our future state. That is Agile; that is Lean. Everything else is a suggestion.

Lean uses a construct from W. Edwards Deming called the plan-do-check-act (PDCA) or plan-do-study-act (PDSA) cycle. Agile teams that take on this cycle reorient toward evolving and improving their processes rather than relying on someone to tell them how to do their work. Teams that actively inspect and adapt are paying attention to their processes, their product, and their customer. Figure 1 shows how it works.

This is all a Lean consultant way of saying something very simple:

> Pay attention to what you are doing and change your processes to make work flow more smoothly.

In the end, Agile is a family of potential business processes. Business processes exist for one reason only — to facilitate the realization of value. Value takes many forms: user satisfaction, sales, team satisfaction, market acceptance, profit, and so on. If our business processes are not meeting the value demands of our business, they will fail and take us with them.

On the other hand, if our processes are constantly making us inspect and adapt, we will make better decisions, deal with change elegantly, and suffer fewer catastrophic failures.

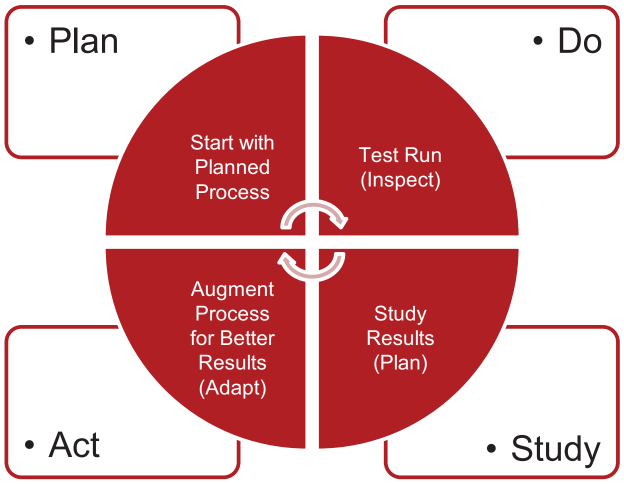

• Plan
• Do
• Act
• Study
Start with Planned Process
Test Run (Inspect)
Augment Process for Better Results (Adapt)
Study Results (Plan)

Figure 1 — The plan-do-study-act cycle.

WE ARE ALL UNIQUE!

There are many aspects of software development that are fairly universal. We have created a large number of good practices to respond to them. Tool vendors have even created fairly good tools to implement those good practices.

Good practices are not best practices.

There is only one best practice in software: *treat nothing as a best practice*.

If we treat nothing as a best practice, suddenly we have to do something uncomfortable. We need to pay attention to our projects and use our processes as controls to steer our teams and our organization to success. We can no longer buy "a box of Agile" from a cadre of coaches and expect to make our goals.

Agile is not a taxi, it's a steering wheel.

ENOUGH RANTING; LET'S TELL SOME STORIES

The following stories are about teams that were doing prescriptive Agile and then inspected and adapted their ways to better states or became stuck when inspection and adaption broke down. Any Agile purist (myself included) can — and will — say that these examples show people *doing Agile wrong*. That's true, but it's also completely false. It's true and false for the same reason: there is no way to do Agile "right" that doesn't involve inspecting and adapting.

Story 1: The Case of the Vile Velocity

In a large global firm, I visited a group of developers. These were brilliant IT professionals who had done amazing work in the previous months. I was consistently impressed with not only how they were building their software, but also how they were visualizing their work and communicating throughout their organization. These were clearly top-notch people.

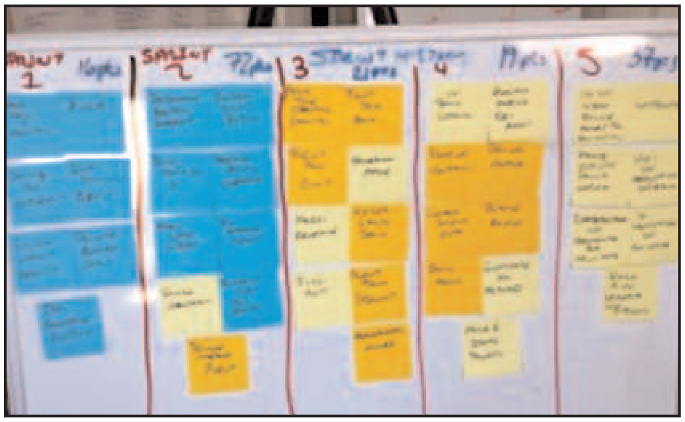


Figure 2 — The team's kanban board.

This group had previously been in a traditional planning environment and had taken it upon themselves to implement various Agile and Lean methods. Their kanban and Personal Kanban boards were impressive. They had many stories about breakdowns, bottlenecks, and recurring failures that they had solved. In short, this was a group of skilled coders and managers. I'd be happy to have any of them on one of my teams.

But their inspection and adaption came to a halt in one key location. They became fixated on an Agile tool and metric. That fixation led them to spend hours trying to fix something that wasn't actually broken.

Here's how it played out. They had a board that showed all their sprints, the story points for each sprint, and the user stories completed. It looked like the board in Figure 2.

Them: We have a problem and we can't figure out how to solve it.

Me: What is your problem?

Them: We can't get our velocity to stabilize. We get our work done, but the story points are all over the map — 16, 72, 21.... What's wrong with us? Why can't we estimate?

Me: They sure are all over the place. Look at that.

Them: If we can't get our velocity to stabilize, we're never going to have meaningful estimates.

Me: Yep. Sure looks that way.

Them: Can you offer any advice?

Me: No.

Them: …

Me: What do you notice when you look at the board?

Them: Our velocity is all over the place.

Me: That's what you see when you look at this board.

Them: Yes. It's clear. Just look at the numbers.

Me: You're looking at the numbers.

Them (confused): Do you have any tips on how to get them more uniform?

Me: You are aware you have no problem here and that your estimation is as close to perfect as it's going to get, right?

Them: What?

Me: These features were in your sprint goals?

Them: Yes.

Me: You see how consistent your cards are? You have a throughput of nine tickets. Make guarantees on six tickets each time and have three or four nice-to-haves.

Them: … but the numbers!

Me: You've proven systematically and empirically that the numbers were bogus from the start. Focus on the work and not the ritual. These tickets are your real work. These numbers are your imagination. The company can't run on your imagination.

Them: But some of those tickets were bigger than others. They aren't right-sized.

Me: Apparently they are. Your work is uniform and unwavering.

Them: But the numbers … our velocity!

Me: Lies. It's all lies, and it's making you accept defeat in the face of victory. You plan well. Velocity isn't the metric you want. Throughput is.

These developers had a beautifully proven and consistent throughput. They knew their release cadence perfectly. Yet they were focused on a faulty metric — they were looking at the proxy when they knew the reality.

In this case, they were fixated on what they erroneously thought to be a "best practice." Story points and velocity were, to them, immutable parts of being an Agile team. Their metric was failing them, but there was a very stable metric sitting right in front of them — the actual number of cards: 7, 9, 9, 9, 7.

There is natural variation in these numbers, but the trend is clear and actually fairly free of real variation. The team didn't have enough experience to promise the full nine or even seven tickets, but six is clearly a low-risk promise for at least the next sprint. Remember, in business we aren't looking for a guarantee, we're looking for an honest estimate.

Understanding that there is a comfort level with six stories with a few other stories that are nice-to-haves changes their conversation with their product owner or customer. It goes from a "We think we can fit this much stuff we don't really understand into these two weeks" to "Our history shows that we can reliably finish six features in two weeks; please give us six things, and we'll get them done."

What is worrying is that these very smart people, who already got most of this, did not see that their delivery was stable and consistent as it was. They were so focused on the proxy metric of velocity that they didn't see the real metric of completed work.

Metric Fetishism and False Alignment

In this case, each time this team entered a planning phase and began to throw down their planning poker cards, they felt lost. This led to long conversations about the upcoming release (not a bad thing) and false alignment (a very bad thing).

Conversations likely led them to come to the conclusion that "We can do those cards" even though the numbers were weird. That is a good thing, but everyone left the table uneasy. *Why was our proposed velocity so different from last time?* They would reach alignment and agree on the set of features to be released, but this was always a false alignment because they didn't understand *why*.

Worse yet, when they would bring their recommendations to their bosses, they had to go to great lengths to defend the upcoming sprint (projected to be 62 story points) over the last sprint (which was 17). They were having extremely long, wasteful conversations because they believed velocity and story points to be the only measures available to them.

What is worrying is that these very smart people, who already got most of this, did not see that their delivery was stable and consistent as it was.

Recommendations for Avoiding the Velocity Fetish

If you'd like to examine whether your team is truly providing predictable value, I give this advice:

- Do what you are doing now.
- Set up a board like theirs that tracks story points and tickets. (You should do this anyway.)
- Watch both the number of tickets and your velocity.
- Track when work is actually started and completed (cycle time).
- Track the number of tickets per sprint (crude deadline-driven throughput).
- When you've done this for five or six sprints, look at the data.

The cards per sprint will give you some vital information:

- Actual work completed every two weeks in number of features or stories
- An idea of the natural variation in work items produced every two weeks
- Work type breakdown (bugs to planned features to unplanned features to emergencies)

By doing this, you'll start to understand your team's work. When that happens, you can do some real estimation. You can guarantee real results. And you can start to focus on what's really important: quality code, fewer escaped defects, and work that relates more closely to customer demands. This can be the basis for real inspection and adaptation.

Every day, these professionals went to work and saw the backlog and felt like failures.

Story 2: Support This!

In America's heartland, surrounded by deer and trees, lies a 40-year-old software company. This software company had a support team that was very, very, *very* angry. There were 24 team members. They supported over 30 products. They would go to work and break things. They would punch holes in the walls, damage their desks, and treat their chairs like bumper cars.

Angry.

Us: Why are you angry?

Them: Management hates us and no one cares about us and we come in every day and try to get our work done but we can't because we're not given any support and we're undertrained and our coworkers are idiots and … (this went on for some time).

Us: That sounds bad.

Them: It's worse than you think. And then there's the backlog ...

Us: The backlog?

Them: Yeah, we have 2,000 tickets in our backlog. It sucks.

Us: How many are you doing a day?

Them: 250

Us: That doesn't sound so bad. How many do you take in per day?

Them (staring death at us): 250

Us: Ahhhhhh …

So there were 24 customer service staff struggling to keep pace with the influx of tickets. Anything that didn't get satisfied right away was dropped into the backlog, which meant a response was long in coming. Every day, these professionals went to work and saw the backlog and felt like failures.

Here's the thing: they weren't failures. Far from it. These people had a reliable Net Promoter Score (NPS) of 95. Time in and time out, they scored in the mid-to-upper 90s. Their customers loved them.

Not what you'd expect from angry vandals.

The System Has Throughput

What was needed here was not for my partner Tonianne and me to give them a new, Agile process. Rather, the point was to employ some Agile or Lean techniques in a way that encouraged these smart, dedicated professionals to solve not only these, but other problems themselves.

The support staff thought that they, personally, were not processing enough tickets. But they clearly weren't the problem. If they were substandard support staff, they would not have high NPS scores. Rather, their processes were inhibiting them from doing their work and from improving their situation.

So we gave them some suggestions:

- All tickets aren't the same size. Try batching tickets according to perceived effort.
- You are all working independently. Form teams that really work together.
- Ask yourselves, "If I were on a real team, what roles would that team have?"
- If your goal wasn't to do as many tickets yourself as possible, but to make sure the number of tickets in the backlog shrunk with no degradation in service, what would you do?
- Every morning, each team should strategize for 10 minutes on how the day will go.
- Every day, make one thing better than it was yesterday.

And then we told them something novel:

> You do whatever you need to do, within reason.

"Within reason" meant that they couldn't rent a new building or move the support staff to Bermuda. Just about anything else was okay.

They did not believe it at first.

Toni and I left for a month, and then we came back to see how things had gone. The results still amaze us. Without outside intervention, these 24 support people came up with:

- **Three teams,** which they labeled red, green, and blue.
- **Two new roles:** the *smooth operator* and the *ticket slayer*. The smooth operator does triage for a team and tackles tickets taking less than five minutes. He or she passes other tickets along to other support staff. The ticket slayer handles tickets that the smooth operator estimates will take 15 minutes or less. This is a high burnout role, so people rotate in and out of this position throughout the day.
- **A display board** showing who the active ticket slayers are. This is important because ticket slayers are quite busy and cannot be interrupted.
- **The Deming Team,** which is made up of representatives of each team and the VP of services to discuss improvement projects and their trajectory.

With this new structure, team members who were not assigned to the new roles did not have to answer calls at all, but instead moved directly on backlog tickets. The Deming Team launched improvement initiatives to clean up the backlog, remove tickets that aged out, merge duplicates, and more.

Within three months, the backlog was half gone; within a year, it was functionally all gone.

Within one month, team morale had improved considerably.

Humans Adapt

For better or for worse, we naturally adapt to our environment. If our processes are static and adaption is impossible, we personally adapt to the inability to improve. When we cannot improve, we react in a very rational way: *we shut down the part of our brain that wants to improve*.

Psychologists call this "learned helplessness." We find ourselves in situations where the system itself won't allow us to improve the system. At that point, we simply look for rules to follow and follow them as best we can to avoid personal harm.

It gets worse. Anyone who has had to pay taxes understands that when we can't change a system, we game the system. We look at the system now in terms of personal risk avoidance. We all do this.

The more rigid the system, the more likely it will be gamed and the greater the toll on your corporate culture that gaming will take. Earlier, this team's management had set specific performance goals and rewarded individual contributors for completing the most tickets in one day. This led to a combative culture where support staff would hoard "easy" tickets, close tickets out early, or create multiple tickets for the same problem simply to increase their numbers.

Once this team was given the agency to improve their own processes, they shed their learned helplessness, ceased gaming the system, and began creating a stronger organization.

For better or for worse, we naturally adapt to our environment. If our processes are static and adaption is impossible, we personally adapt to the inability to improve.

Story 3: That's for Us to Know ... and for Us All to Find Out

In the early 2000s, I ran a software company called Gray Hill Solutions that specialized in software-for-hire for the government sector, specifically in transportation management and traveler information. Among other projects, we built the original 511.org site for the Metropolitan Transportation Commission in the San Francisco Bay Area. We specialized in rescuing government projects that had spent most of their budget and their time, yet had little to show for the effort.

Not surprisingly, most of our work had severe budget and time pressures and high customer anxiety. We were ostensibly an XP shop and took much of what Cutter Senior Consultant Kent Beck wrote to heart. Our teams were tight, distributed, and focused. Project-based work with tight deadlines and a strong will to watch and learn led us to regular deliveries and the ability to immediately spot, communicate, negotiate, and correct problems that arose.

This meant that with every project, indeed with every stand-up meeting, we were evolving our version of XP. In this case, "Agile" became the way we related to each other, developed software, and communicated with our clients.

At one point, we had the opportunity to build a multi-enterprise-scale software package that would allow regional agencies to communicate the status of all roadways in real time and manage regional transportation networks as one cohesive system. This had never been done before. We would need to re-create from the ground up how regional networks were conceived and managed.

Our client asked us for a scope and budget.

We refused, telling them that this type of work had never been attempted before.

They asked again.

We refused again, saying that any firm contract drawn up with this number of unknowns would not be a negotiation of work, but rather a negotiation of risk. We'd write the document to push risk on them; they would write it to push risk on us. No matter who "won" that negotiation, we would both lose in the end by having a contract with a firm scope that wouldn't respond well to the level of change we all knew was coming.

To say that our client was highly skeptical would be an understatement.

Instead, we created an Agile contract that focused on inspecting and adapting. The rules of the contract were simple:

1. Agreed: both parties wanted a good product.
2. Agreed: neither party knew what the final product would look like.
3. The parties would aggressively collaborate:
   a. The client would attend every stand-up meeting.
   b. The client would be involved in every decision made.
   c. The client would have direct access to every developer.
   d. Every developer would have access to the client.
   e. We would do 10-minute code commits. (Yes, we did continuous release of enterprise-scale software in 2006.)
   f. The client could build at any time.
4. The parties would use a digital kanban board with work-in-process limits set up in Groove 2.0:
   a. The client and all staff would have access to the board.
   b. Decisions made in the daily stand-up would be reflected immediately on the board.
   c. No work was to be completed that wasn't encapsulated in a card on the board.
5. We would have regular two-week releases with demos. (This rule was quickly dropped because iterations became unnecessary.)

To say that our client was highly skeptical would be an understatement. For our first delivery, we proposed to have a mock-up of the UI with fake processes running on it in three months. The customer led with a statement about how that would be impossible.

In fact, the team finished that in six weeks, which left us another six weeks in which to include a few real working systems in the UI.

In this case, we were not only inspecting and adapting as we built the system — that was a daily and routine event. What was interesting was that we were also inspecting and adapting the relationship between client and contractor. Ostensibly, we were working with no budget. We had a daily burn rate we could not exceed, but that burn rate was actually greater than the sum total of salaries of everyone working on the project. So we were safe there.

We were, every day, making new decisions about the product based on what we'd learned the day before. Some days, course corrections were small; other days, they were pronounced. None of them were threatening, however, because the client was intimately involved in the project merely by spending 5-15 minutes a day on Skype.

Allowing the client firsthand, unfiltered knowledge of the decisions being made about the software had the following advantages, all of which are part of inspection and necessary to be able to comfortably adapt:

- **Understanding.** By simply being present while discoveries and decisions were being made, the client *never* needed a status briefing, excuses, or a demo. They knew in real time what was being done and what challenges we were facing, and they were making decisions with the team on direction.

- **Empathy.** Engagement led to understanding, which led to questions like "How is that going?" rather than "Why didn't you do this?" Real-time involvement removed opportunities for second guessing.
- **Participation.** The client's direct participation led to their doing things like calling up in the middle of the day and offering suggestions. It also led them to immediately conduct acceptance testing of code, because they were as excited to find a solution to whatever issue was plaguing the team as the team was. In this case, participation didn't merely mean showing up, it meant acting.

With these elements in place, our project had active engagement on both sides. There was no passive party. We were all actively inspecting the processes, relationships, and products of the project and offering hypotheses for improvement.

This did not slow us down, because most suggestions were made at the stand-up. Since stand-ups were daily, suggestions could be small, implementable, and reportable. Because of this, our backlog of work could easily change as the project progressed. We were able to quickly onboard contractors with specific skill sets when we discovered a need, and we were able to test potential solutions and determine whether they worked. If they didn't, we'd back them out and try again. There were no commits of worthless or sketchy code.

CLOSING: ADAPT OR DIE

For me, above all things, *inspect and adapt is the golden rule of process management*.

The goal of all process control or project management should be nothing more or less than making sure that all stakeholders remain involved, invested, and integrated.

Your Agile, Lean, or [fill in buzzword here] method either adapts to change and improves over time, or it remains stagnant and resists opportunities. Every element of change that presents itself to your team or your organization is an opportunity to deliver a better product in a better way.

Make the most of these opportunities.

Jim Benson, CEO of Modus Cooperandi, specializes in Lean project management and the management of knowledge work. He is the creator of Personal Kanban and, with Tonianne DeMaria Barry, coauthored the book Personal Kanban*, which won a Shingo Research Award for Excellence in 2012. He is the 2012 winner of the Brickell Key Award for excellence in Lean thinking.*

For the past two decades, Mr. Benson has worked at uncovering ways for groups to find clarity in unpredictable and amorphous knowledge work environments. Since starting Modus, he has helped the World Bank, NBC Universal, the United Nations, Spotify, Riot Games, Comcast, R.W. Baird, and others improve their kanban systems, implement collaborative solutions, identify and implement improvements, and create more innovative cultures. He can be reached at jim@moduscooperandi.com.

CUTTER CONSORTIUM

●●● ACCESS TO THE EXPERTS

Cutter Membership

The ultimate *Access to the Experts*

"I am amazed that this venue exists; that I can listen and interact with these individuals. I am more amazed that I have not done this before."

— Mark Rubin,
Fidelity Investments,
USA

"Cutter seems to be unique in consistently providing information one can immediately put into action or clarify one's thinking on day-to-day problems. The service provides tremendous value for the money."

— Lloyd Fletcher,
Information Systems Manager,
Institute of Physics Publishing,
Bristol, UK

With unlimited access to Cutter's research, inquiry privileges with Cutter Senior Consultants and Fellows, regular strategy meetings for your team with a Cutter Practice Director, virtual roundtables with Cutter experts and peer-to-peer networking, free or discounted admission to events, and more, your Cutter Membership opens up multiple avenues to interact with Cutter's experts to brainstorm and gain guidance aimed at boosting success.

Like everything IT, one size does not fit all. That's why we encourage you to choose the Membership level that's right for your organization:

Full Membership

Best for organizations seeking personal, real-time guidance on the full gamut of business technology and software engineering issues and dedicated to investing in the career development of their entire staff.

Business Technology Strategies

Best for organizations developing operationally sound strategies and plans that link and align business and IT, and harnessing the latest leadership and management techniques to ensure IT is furthering the mission of the business.

Agile Product & Project Management

Best for software and product development organizations that are seeking to create and capture lasting value through end-to-end Agile initiatives.

Business & Enterprise Architecture

Best for organizations interested in using architectural approaches to manage the complexity and cost of IT and to pave the way for strategic planning and portfolio management.

Data Insight & Social BI

Best for organizations challenged with data collection, analysis, and integration

as well as internal- or external-facing information sharing via social, mobile, or legacy apps.

CIO Add-On

Ongoing, one-to-one relationships with seasoned expert mentors; research and analysis specially curated to support high-level decision making; peer-to-peer networking.

Membership Benefits

- Continuous flow of advice, insight, and answers via enterprise access to written and multimedia research from Cutter's top experts
- Priority Access to the Experts: get near-real-time answers to your questions
- Regular strategy sessions with Cutter Practice Directors or Senior Consultants
- Admission to client-only online Q&A sessions
- Participation in peer-to-peer discussions led by Cutter's experts
- Cutter events
- Add-on options for consulting and training offerings

And more ...

To arrange for a free trial membership or to discuss how Cutter's consultants can help your organization, contact our team today: sales@cutter.com or +1 781 648 8700

What's Unique About Cutter?

- Cutter's internationally recognized expert practitioners provide all of Cutter's research and analysis. You get to tap into this brain trust whose written words have been likened to a "consultancy in print."
- Without exception, every single inquiry is fielded by a Cutter Senior Consultant, Fellow, or Practice Director.
- Cutter approaches every consulting or training assignment as unique, requiring a tailor-made solution, and creates a team for you that includes only its best-in-class experts. We focus on knowledge transfer, so you can leverage our work together and move forward on your own.
- With Cutter, you get cutting-edge thinking from multiple viewpoints so you can determine what's best for your situation.
- Emphasis is on strategies and processes, so you can be sure your success is not dependent on vendor/product detail.
- Cutter is unique in having no ties to vendors. Rest assured that the advice you get is unbiased and in the best interest of your organization alone.
- Focus includes the business management of IT — you're plugged into the research from top business thought leaders.

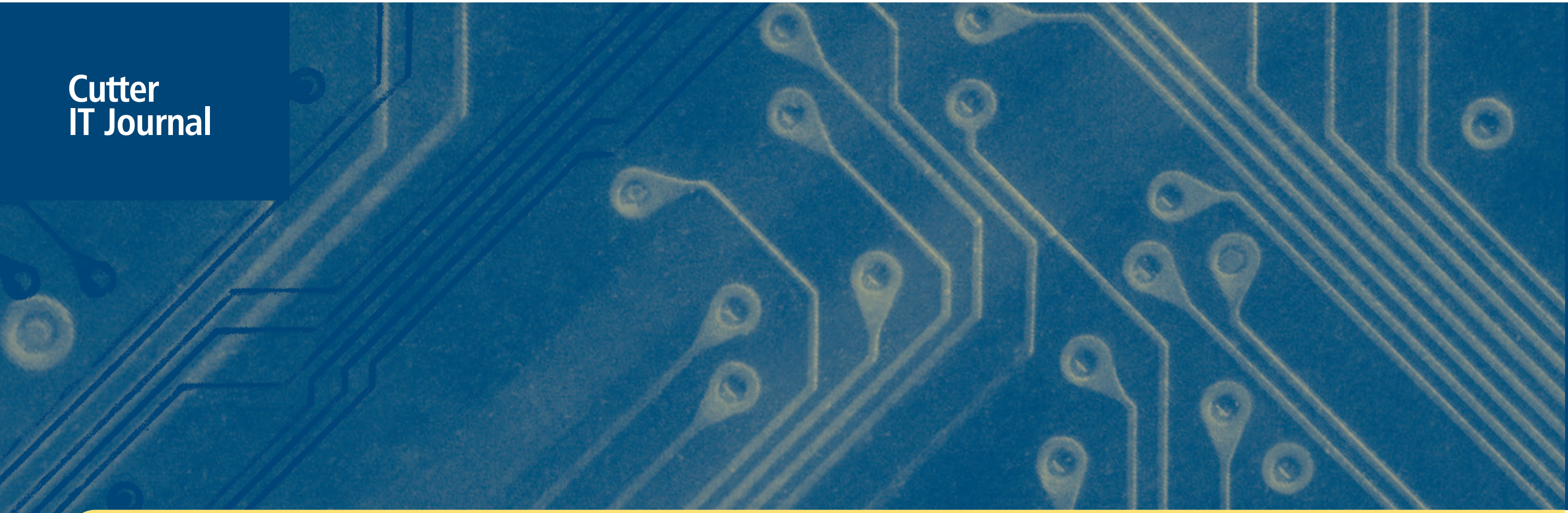
"I have personally been able to leverage Cutter's services since 1999. Among the attributes that differentiate Cutter from other firms, two remain at the top of my list — *Thought Leadership* and *Real Value* — executed in a practical way.

Thought Leadership is driven by Cutter's experts. The advantage is that Cutter doesn't pitch a single best practice for a given area. Instead, Cutter provides multiple good practices/options that come from both academic rigor as well as on-the-ground experience. This provides several benefits for Dairy Farmers of America:

- Exposure and awareness of proven good-practices — particularly for IT, but also for overall business leadership and management
- A finger on the pulse of emerging good practices and IT-impacting trends
- Options for improving our performance
- The opportunity to develop relationships with the experts

The last, 'Access to the Experts,' drives the *Real Value*, letting us go beyond just understanding the options. We can develop relationships with the experts and tailor the options so that they can be quickly and practically executed within our organization, enabling our Business Technology team to continually improve, engage, and contribute to business growth."

— Doug Mikaelian,
VP Business Technology,
Dairy Farmers of America



About Cutter Consortium

Cutter Consortium is a truly unique IT advisory firm, comprising a group of more than 100 internationally recognized experts who have come together to offer content, consulting, and training to our clients. These experts are committed to delivering top-level, critical, and objective advice. They have done, and are doing, groundbreaking work in organizations worldwide, helping companies deal with issues in the core areas of software development and Agile project management, enterprise architecture, business technology trends and strategies, enterprise risk management, metrics, and sourcing.

Cutter offers a different value proposition than other IT research firms: We give you Access to the Experts. You get practitioners' points of view, derived from hands-on experience with the same critical issues you are facing, not the perspective of a desk-bound analyst who can only make predictions and observations on what's happening in the marketplace. With Cutter Consortium, you get the best practices and lessons learned from the world's leading experts, experts who are implementing these techniques at companies like yours right now.

Cutter's clients are able to tap into its expertise in a variety of formats, including content via online advisory services and journals, mentoring, workshops, training, and consulting. And by customizing our information products and training/consulting services, you get the solutions you need, while staying within your budget.

Cutter Consortium's philosophy is that there is no single right solution for all enterprises, or all departments within one enterprise, or even all projects within a department. Cutter believes that the complexity of the business technology issues confronting corporations today demands multiple detailed perspectives from which a company can view its opportunities and risks in order to make the right strategic and tactical decisions. The simplistic pronouncements other analyst firms make do not take into account the unique situation of each organization. This is another reason to present the several sides to each issue: to enable clients to determine the course of action that best fits their unique situation.

For more information, contact Cutter Consortium at +1 781 648 8700 or sales@cutter.com.

The Cutter Business Technology Council

The Cutter Business Technology Council was established by Cutter Consortium to help spot emerging trends in IT, digital technology, and the marketplace. Its members are IT specialists whose ideas have become important building blocks of today's wide-band, digitally connected, global economy. This brain trust includes:

- Rob Austin
- Ron Blitstein
- Tom DeMarco
- Lynne Ellyn
- Israel Gat
- Vince Kellen
- Tim Lister
- Lou Mazzucchelli
- Ken Orr
- Robert D. Scott