

CompAct

ELECTRONIC NEWSLETTER



CONTENTS

[Table of Contents](#)

[Letter From The Chair](#)

[Editor's Notes](#)


[Our Experiences With Agile](#)

[Excel 2010—An Analyst's Perspective](#)

[Actuaries Looking To The SCAI For Answers](#)

[R Corner—Memory Management](#)

[Number Puzzle](#) 

[Last Issue's Number Puzzle Solved](#) 

[The 9th Speculative Fiction Contest](#)

[Articles Needed](#)

QUICK LINKS

[Technology Section Web site](#)



Share



Print Article



Search Back issues

OUR EXPERIENCES WITH AGILE

By Robert Ream and Justin Bozonier



MG-ALFA is financial projection software developed and supported by Milliman and used by leading life insurance and

financial firms worldwide to perform financial projections. Among other things, our product is known for consistently delivering quarterly software releases.

Historically these regular, content-rich releases were delivered with a small team of very talented developers. We were an agile development team, but it was by accident. We were not aware of why we were so effective, so as we tried to expand, our team struggled to maintain the agility. We decided to evaluate our development process from the ground up, and allow for the possibility of radical change. After several team members and our stakeholders read *The Art of Agile Development* by James Shore, we decided to bring James in to help us understand how to regain our effectiveness with our larger team. Often, moving to an Agile approach is difficult to sell to the stakeholders, but because the core principles and expected outcomes of Agile software development were consistent with the lost expectations of the stakeholders, it was an easy decision.

In today's software development world there are few buzz words that are more overloaded and overused than "Agile." Coined by a group of 17 thought leaders in "lightweight" software development methodologies, this "Agile Alliance" authored what became known as

Council

[Links of Interest](#)

[Fiction Contest](#)

[Howard Callif, Editor](#)

SOA Staff
[Meg Weber, Staff Partner](#)

[Sue Martz,](#)
Section Specialist

[Sam Phillips, Staff Editor](#)

the "The Agile Manifesto." It is a very succinct statement, so we have included it here:

"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."

—The Agile Alliance

Our Solution

Before James would commit to coming in, he talked with every person individually to assess his or her comfort level with changing core processes and communication styles. Taking on this challenge took a tremendous act of faith, courage and hard work on everyone's part, from the engineers, eager to provide the required flow of software, all the way up to our stakeholders, anxious to regain our stride and take advantage of emerging market opportunities.

The solution we committed to as a team was to restructure how we were working and communicating together. James helped us learn how to rapidly respond to change and keep our software in a constantly releasable state, and how to communicate as a single indivisible unit.

The first step in our transformative process was to construct a vision with our stakeholders. Vision statements are often broad and banal, and consequently are also often ignored. The vision we created was tangible and focused on the next six months of product development. It also focused on the actual challenges we were facing; it was specific. Simply writing down this direct and honest statement did something our team hadn't experienced in too long: It aligned our individual goals and gave us a common purpose.

Now we all knew where we were heading over the next six months, but how were we going to get there? The vision that we laid out was concrete enough to ensure we all knew what would make the next six months a success, but was not specific enough to be

independently actionable.

The vision was further divided up into Minimally Marketable Features (MMF). These are the smallest features we could possibly deliver to our customers while still providing value. These features formed the road map to our destination. As you can imagine, there is a large list of MMFs, but with our vision clearly articulated, we can more easily prioritize them and identify how each will contribute to achieving our vision. While this seems simple, we realized that prior to implementing this process we had been assuming a common vision and understanding. That had been true when our team was small and had a long history, but we had lost that during our growth without realizing it had happened, or the consequences.

Even though we now had features clearly defined and their value established, they were still rather large chunks of work. How would we know when these were done? How could we give our stakeholders visibility into our progress while we worked on these features that might take six to eight weeks to complete? Even more important, how can we ensure that we are delivering what our customers want?

For this James Shore introduced us to "Stories" and "embedded customers." Stories are ideally the smallest possible body of work that has any real value. It acts as a sort of visible checkpoint so that your customers have a real and testable mark for where you are. In an ideal world, every Story would be an MMF. To ensure we are delivering what our customers want, our team includes proxies for our customers—embedded customers. Having users included in the team room with the programmers provides subject matter expertise and feedback in real time. This has significantly reduced rework and developer downtime related to waiting for input from the business experts.

Many times we found that we had created more stories than we really needed to finish an MMF. This allowed us to "Maximize Work Not Done" and focus like a laser on the value we were actually delivering to our customer base. Previous to our transition to Agile development, our checkpoints had such a long interval between them and access to business experts was less convenient, so decisions on value were often made by the software engineers. The issue here, of course, is that developers don't always have the same context as the business customers, and therefore may make suboptimal decisions.

Prioritizing which MMFs to work on was and still is a team decision

for us. Our embedded customers ultimately have the final say, but the cost estimates that the development team provides are taken into consideration and allow the stakeholders to make fully informed decisions. If a story will take three weeks to be completed and there is only one week left in a release cycle, it is in our best interest to prioritize some smaller stories to fill the gap.

Providing these estimates was a challenge, but we have become more adept and predictable during our transition process. Our developers estimate the stories in "Ideal Pair Days." In doing this we get a consistent idea of how long it would take a pair of engineers pair programming to finish the story at hand. These estimates also enable our business to predict how much work we will get done and enable a good amount of risk management.

To recap: We had a vision, a set of features that would help us achieve our vision upon completion, and a set of embedded customer oriented checkpoints that would help to guide us along our way. The only thing left was for us to plan out the individual tasks for the software engineers on the team. You might think that this is where the embedded customers can check out and go about their day, but that would be dead wrong.

While the customers did work on other tasks while the planning was going, they remained immediately available for questions from our engineers. Where debates would occur regarding importance of UI look and feel, the customers could quickly squash them by making a value decision. Where engineers may fret over concerns about performance, customers can provide guidance for acceptable performance levels with engineer feedback regarding the costs of solutions.

Finally, we got to work. As the weeks of the release cycle ticked by, we noted how much value we had delivered to our embedded customers with what we call "velocity." The value of velocity is driven by the estimates given by the programmers, and how it is used can be somewhat confusing. For example, just because your velocity is steadily increasing does NOT mean your team is improving in any way. One of the key ideas behind these Agile methods is that they are not a silver bullet. James warned us that just following the "rules" by rote memorization wouldn't solve our problems. He encouraged us to always ask "why?" What are some reasons other than team improvement that might cause our velocity to go up? Perhaps we scheduled a number of stories with a high variance in the same iteration. Or maybe we were overly pessimistic in our estimates. On the other hand, what are some of the reasons velocity might

decrease, other than degradation in team performance? Since we are measuring velocity each week, absences for illness or vacations can lead to a drop in velocity.

What did we do when key developers were out sick or away on vacation? We mitigated this risk by having all of our developers pair program on all production code that is written. Pair programming was the one practice about which our stakeholders were most wary. Why would we pay for two people to do the work of one? However, it did not take long for everyone to appreciate the benefits of pair programming—including reduced cost.

Not only does pair programming eliminate the risk of knowledge silos as mentioned above, it also produces better designed and higher quality code. Many of our business experts are pairing with each other as they work on client projects, presentations and other non-development tasks. Everyone has recognized that working collaboratively to solve problems, whether through writing code or writing a report, is productive and ultimately creates a higher quality product.

Another technique we use to ensure a higher quality product is Test Driven Development (TDD). TDD centers around the idea of writing a failing test prior to writing code. In this way we express what the expectations of our next change are, then hypothesize and implement the small isolated changes to the system that we think are necessary to get the test to pass. If our hypothesis is verified and the code passes the test, we refactor the code to make it as clean as possible and then move on to the next test. We made a concerted effort to develop all new code in this manner. Furthermore, we refactored legacy code to allow for this style of development when possible.

In fact, what we found is that programmers who were less familiar in a particular code base asked questions that quickly led to "aha" moments for those more experienced in those areas. You can almost always find that someone with fresh eyes for the problem at hand provides valuable insights. To maximize the advantages of pair programming, we gave our whole team equal code ownership. We have no architect and no software designer, just a team of passionate engineers who are encouraged to discuss new system designs and architectures together to formulate the future direction of our product.

Looking Back

Looking back on our journey until now, it would be best characterized as always questioning what "done" looked like. Our stakeholders had a vision and they communicated what it would like when we achieved it in a real and measurable way. We worked with our business team and described the stories that would need to be done to have a complete feature. Everything, down to each line of code where a failing test suddenly passes, shows us that we've completed yet another step in our journey.

This is how Agile software development works for us. It's a process of continuous change and improvement. We must always make sure we have time to experiment and never assume we should not change the process. And most importantly, we must remain focused on business value. We've taken what James Shore has taught us and made it our own with our own successes to show for it. That is really the heart of the lesson in the end. Find what works best for you; keep searching with the belief that you can always do better, and that focusing on continuous improvement is the best investment you can make for individual growth, team growth and business growth.

Robert Ream is a Senior Software Simian for Milliman's MG-ALFA team, and can be contacted at robert.ream@milliman.com

Justin Bozonier is a Code Samurai for Milliman's MG-ALFA team, and can be contacted at justin.bozonier@milliman.com



475 North Martingale Road, Suite 600 Schaumburg, Illinois 60173
Phone: 847.706.3500 Fax: 847.706.3599 www.soa.org