Primavera Gets Agile:
A Successful Transition to Agile Development

Bob Schatz and Ibrahim Abdelshafi, *Primavera Systems*

Primavera's development team is a model for others looking to adopt agile processes. Learn how to deliver high-quality software while improving your team's quality of life.

Primavera Systems provides enterprise project portfolio management solutions that help customers manage their projects, programs, and resources. Our development organization employs 90 business analysts, programmers, testers, documenters, and project managers. When we decided to improve how we build software and increase the quality of life for everyone on the team, we found our answer in agile software development.

Adopting agile practices is a process of continuous learning and improvement. The transition required hard work, intense focus, and strict discipline. As a team, we discovered many things about ourselves, our process, and our products. Every discovery revealed a better understanding of what it takes to successfully deliver software at a sustainable pace.

By telling our story, we hope to help others in the software industry successfully transition to agile development. The time is right for the software development community to improve its historically low success rate, and agile development might provide the answer.

Starting down the agile path

People assume that a company that designs and builds project management software has all the knowledge and tools necessary to continuously perform software development flawlessly. The truth is, we struggle just like everyone else.

Primavera has its roots in construction and engineering. Our culture supported the commonly used waterfall development approach, and the development team worked hard over the years to distribute high-quality releases to a growing, diverse market as quickly as possible. The team followed the typical cycle, which usually resulted in working late nights and weekends in an attempt to finish projects on time.

Primavera's project managers typically used a command-and-control philosophy—a few people, usually those farthest from the action, made the decisions without directly involving the developers doing the actual work. Furthermore, relationships between the development team and other departments began to deteriorate because expectations were seldom ful-

0740-7459/05/$20.00 © 2005 IEEE

filled. So, in conjunction with a leadership change in 2001, we took a hard look at the many issues we had to resolve with respect to the organization, its people, and its processes. In particular, we had to improve the company's perceptions of its own development team, and the team itself had to reduce its deep-rooted dependency on the organizational hierarchy and break down functional silos within the department.

We started by identifying and then synchronizing the vision and values driving our initiatives. We set out to be a world-class development team, focused on our customers and recognized within industry not only for our products but also for how we developed them. We also hoped to create an environment where we could learn and grow, explore our creativity, and have fun.

Changes made in the first year positively affected the team and started to repair relationships in the company. We also moved into a new office space, which for many symbolized a new beginning. We began working on team building, learning how to work better across functional teams. We also enforced a set of shared values using the Fish! Philosophy to help establish desired behaviors.[1]

Everything we did moved us one step closer to our vision, but several nagging questions remained: Why do software development professionals repeat the same ritual over and over again with the same painful results? Why do we push ourselves to exhaustion, working 18-hour days, seven days a week, to deliver software that many users don't really appreciate? Why do we repeatedly lead people down that same path?

Despite the changes we'd made, following the development of a software release in March 2003, we realized it was time for a more significant change. After working to exhaustion, we celebrated our victory, thinking we had delivered a great release. Then we discovered a different perception outside of development—others thought the release lacked features and had quality issues and usability problems. This resulted in feelings of frustration, disappointment, and apathy for the development team. We needed to be able to update priorities and features on the basis of shifting market needs, and we needed to get more people involved in the development process. We needed to be agile.

Discovering and implementing Scrum

Adopting an agile process began with our willingness to take risks. We knew we might fail, but nobody wanted to take on another project using the current development process.

We started reading about iterative development to get some ideas and found an overview of a process called *Scrum*. It comprised a set of project management principles based on small cross-functional self-managed teams (*Scrum teams*), 30-day iterations (*sprints*), and 40-hour work weeks.

The process involves having each Scrum team and *product owner* work together at the beginning of each sprint to define the sprint's goals. The product owner, typically someone from the marketing department, acts as the product manager. He or she determines which features must be implemented in a release to satisfy the market needs. Additionally, each team has a *Scrum master* who coaches the team through the process and removes any obstacles. Each day, the teams hold a 15-minute stand-up meeting and each team member states his or her accomplishments for the previous day and plans for the current day as well as any obstacles preventing the team from doing its work. The Scrum master's primary role is to remove such obstacles, enabling the team to remain focused.

Each iteration concludes with a *sprint review*, in which the team demonstrates its accomplishments to the stakeholders, product owners, customers, and other department representatives. The review provides an opportunity to receive constructive feedback, which helps shape and refine the product backlog.

Because Scrum acts as a wrapper around existing development processes, it can work with any existing methodologies you follow. We felt it would complement rather than clash with our culture, so the management team in development met to discuss our chances of success with agile and Scrum. We then went a step further and called Ken Schwaber, one of Scrum's creators.[2] After visiting our office and learning more about our culture, Ken was not only confident that Scrum could work for us—he agreed to be our coach.

One of the first things we *didn't* do was start telling everyone that we planned to use a new process. We didn't want to make people nervous or apprehensive, and we wanted to give them time to adjust to the changes. Plus,

Because Scrum acts as a wrapper around existing development processes, it can work with any existing methodologies you follow.

You need a sponsor—someone who's willing to put everything on the line and is committed to moving to agile.

when you run around announcing your new process and all its benefits, you can quickly set unrealistic expectations.

Instead, Ken started by holding a workshop for the entire development team, talking about agile development and Scrum's principles. Then we held a training session to certify 15 Scrum masters.

Although we presented Scrum as the solution to our problems, reaction was mixed. People were uncomfortable with a change of this magnitude, because they worried how their roles and responsibilities would change. They also questioned who would provide direction; they wanted creativity and freedom but weren't necessarily ready to take on such responsibility. Regardless of these concerns, we knew this was the right thing to do and thus were committed to making the necessary changes.

Tips for moving to agile

One of the first discoveries we made is that you need a sponsor—someone who's willing to put everything on the line and is committed to moving to agile. Someone has to stand up to the critics, encourage the leaders, and communicate the team's vision.

For anyone leading the transition to agile, here are some tips we learned from our experience.

Use objective coaching. When undertaking a significant transition, it's extremely helpful to get honest, objective feedback from an outside source. Coaching helped enforce a learning culture—we would iteratively learn new techniques, try them, and see where we could improve. When you "live" in the environment every day, you can miss little things that make a big difference.

Focus on teamwork. Teamwork and team building are critical to establishing self-managing teams. Team colocation is a real boost to productivity. As managers learn to properly delegate to teams, they should shift their focus from tasks and assignments to team dynamics. Teamwork is what makes agile work so well, but getting teams to perform their best requires a lot of attention.

Use the established agile language. Many organizations, in their efforts to adopt new techniques, try to soften the blow by adapting the language to something more familiar in the culture. This could be a mistake when implementing agile. Its language differs from what we know and recognize in traditional development, but using the new language forces people to think in new ways and helps foster creativity. If you change the agile language to match what was previously known, people tend to slip back into their old ways.

Get executive support. Unlike other initiatives, such as ISO and CMM (which often come in a top-down directive from management), agile is growing in a bottom-up fashion. Adopting agile, or implementing any significant change, requires an executive's sincere support. It can be a bumpy ride until things settle down, and having executive support lets the learning take hold despite any problems or failures.

Don't work overtime. One of the key motivations for moving to agile is its ability to create a sustainable pace for the development team. The death marches must stop! We need to quickly and firmly deal with software teams' addiction to overtime. We can consistently deliver high-quality, complex systems without working nights and weekends.

Learn to negotiate and set expectations. In agile, negotiation and expectation-setting are ongoing activities, because the team gathers feedback and demonstrates the product after each iteration. Negotiation skills become critical as team members learn to work with each other and with developers outside the team.

Watch for trouble. Watch for subtle trouble signs early in the adoption. As with any transition, people will always look for reasons to return to what they know. Managers and leaders must give positive reinforcement to keep people in a learning mode. By continuously trying new things, creativity will take hold and team performance will rise.

Expect hard work. There is no silver bullet—saying you're agile doesn't make you agile. Despite what many might say, there's no easy answer to adopting agile. It requires a change in how we work as an industry and will manifest itself in different ways in every organization. The best we can do is to learn from—and continuously improve upon—our experiences.

A tool for managing the Scrum project

Despite what people might think about agile, it requires discipline and strong project management. In many organizations, project management tools provide insight into a project's progress. Primavera provides a comprehensive suite for project portfolio management, which we used effectively to manage our first Scrum project. The suite's Project Management module captured the release backlog and represented each of its features as a top-level *Work Breakdown Structure* element. Then we created a *resource team* to represent each Scrum team on the project. Figure 1 illustrates the release backlog, captured in the WBS View, along with a list of teams assigned to each feature on the backlog.

At the beginning of a sprint, each Scrum team selected individual features from the release backlog by assigning itself to the corresponding WBS element. The Scrum team then broke down each feature into detailed requirements and added them as lower-level WBS elements. Team members then broke down each requirement into multiple 8- to 16-hour tasks and assigned themselves to these tasks.

At the end of each day, team members updated the remaining work for each of their backlog tasks using the lightweight, Web-based Team Member module that we created (see Figure 2). They also used the tool to create and modify assignments for additional features on the release backlog. The tool fed the updated remaining durations back into the release backlog. As the team completed a feature or requirement, the tool marked the corresponding WBS element as complete in the release view.

The Scrum masters used the updated infor-

Layout: Sprint Layout

WBS Name	Planned Sprint	Resource Team
SCRUM Module	0	
Individual Contributor Module	0	
Display SPRINT Backlog	0	
display all functional requirements and their related SPRINT tasks	1	Agile Express
display Actual Work and Remaining Durations for each task for each day	2	Agile Express
display the entire sprint log with Durations for the entire sprint duration	3	Agile Express
The list shall be sortable	3	Agile Express
The list shall be filterable	4	
The system shall indicate the current day	4	Kaizen Kids
Initial Setup to start project	4	Kaizen Kids
Status Tasks	0	
user to assign a RD to a SPRINT task for the current day	2	Kaizen Kids
user to assign an AD to a SPRINT task for the current day	2	Kaizen Kids
user to assign a STATUS to a SPRINT task	3	
Update Release Backlog	0	
allow user to Update the release backlog	2	Krafty Contractors
allow user to CLOSE OUT the SPRINT	3	Krafty Contractors
allow user to TERMINATE a SPRINT	4	Karfty Contractors
update the RELEASE backlog with SPRINT updates	5	Krafty Contractors
Scrum Master Module	0	

Figure 1. A project's release backlog, as displayed using Primavera's Project Management module.

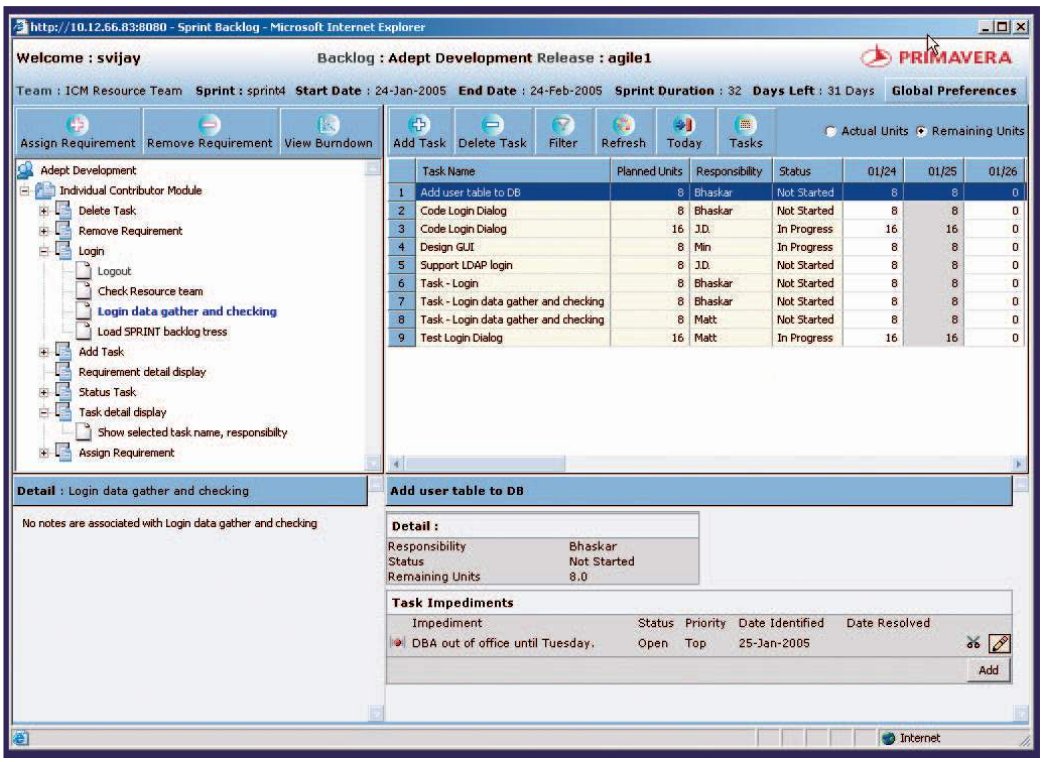
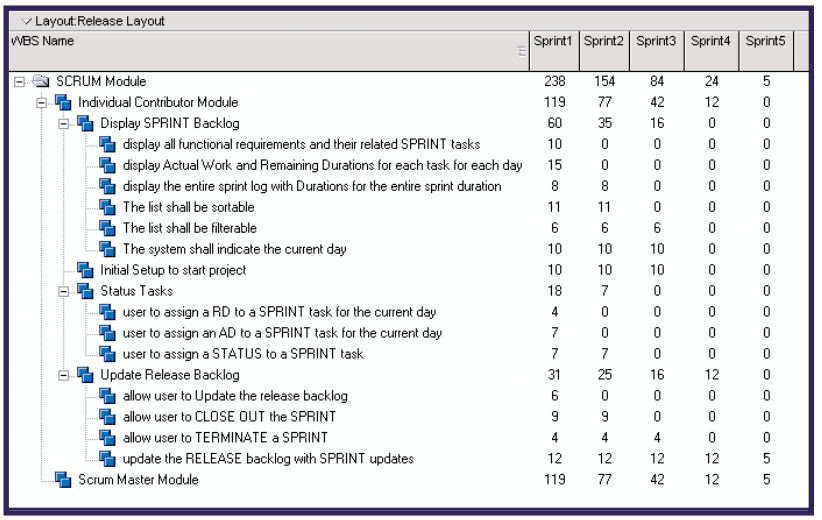


Figure 2. The Team Member module, showing the original duration for each task on the sprint backlog, the developer responsible for completing the task, and the task's remaining duration at the end of each day of the sprint.



Layout:Release Layout

WBS Name	Sprint1	Sprint2	Sprint3	Sprint4	Sprint5
SCRUM Module	238	154	84	24	5
Individual Contributor Module	119	77	42	12	0
Display SPRINT Backlog	60	35	16	0	0
display all functional requirements and their related SPRINT tasks	10	0	0	0	0
display Actual Work and Remaining Durations for each task for each day	15	0	0	0	0
display the entire sprint log with Durations for the entire sprint duration	8	8	0	0	0
The list shall be sortable	11	11	0	0	0
The list shall be filterable	6	6	6	0	0
The system shall indicate the current day	10	10	10	0	0
Initial Setup to start project	10	10	10	0	0
Status Tasks	18	7	0	0	0
user to assign a RD to a SPRINT task for the current day	4	0	0	0	0
user to assign an AD to a SPRINT task for the current day	7	0	0	0	0
user to assign a STATUS to a SPRINT task	7	7	0	0	0
Update Release Backlog	31	25	16	12	0
allow user to Update the release backlog	6	0	0	0	0
allow user to CLOSE OUT the SPRINT	9	9	0	0	0
allow user to TERMINATE a SPRINT	4	4	4	0	0
update the RELEASE backlog with SPRINT updates	12	12	12	12	5
Scrum Master Module	119	77	42	12	5

Figure 3. A release burndown report for the project's first five sprints.

mation in the project to generate daily *burndown* charts for their teams. They also created release burndown charts for the project, which provided an up-to-the-day view of the project's process and its completed features. Figure 3 shows the release burndown report for the first five sprints, captured inside the WBS view.

At the end of each sprint, the product owners reviewed the release backlog's priorities using the WBS view and adjusted priorities and requirements as needed. The Scrum teams began their next planning session, assigning themselves to the revised release backlog's highest-priority items.

Measuring our success

In a Scrum environment, it's often difficult to measure your success against a predetermined plan because Scrum lets business owners adapt and change their plan every sprint. However, measuring Scrum's impact based on product quality and time to market was easy.

We experienced a 30 percent increase in quality as measured by the number of customer-reported defects in the first nine months following the release—0.36 defects per KLOC for this release versus 0.51 in the previous release. In addition to providing our customers with a higher-quality release, the reduced number of reported defects allowed us to focus on the next release rather than addressing existing issues.

Implementing Scrum on this project also improved our time to market. We had planned to deliver two parallel releases in a risk-loaded 14 months. The flexibility that Scrum provides enabled the product owners to adjust to market conditions and consolidate the two releases into one backlog containing the highest-value items from both releases. We delivered this combined release four months early. Had we followed our traditional development process, working on the two releases separately, we couldn't have changed course midstream to deliver one release with the combined feature set.

The true benefits of adopting Scrum on this project, however, go beyond measuring the number of features completed during the release cycle. We gained many intangible benefits as a development team and as an organization.

Benefits to the team

Scrum created a long-term, sustainable pace for the development team while significantly improving the work environment. During the project's entire development cycle, the team never worked overtime or weekends. Additionally, developers got to work outside their roles to help their Scrum team achieve its sprint goals. This teamwork made the work environment more enjoyable for the developers and helped build trust between them, which is fuel for high-performance teams.

Developers took ownership of the features they created and took pride in showing their work to the stakeholders during sprint reviews. The teams also worked closely with the product owners and consequently had a much better understanding of their work's importance and the business value of the features they were implementing. The features' design and implementation ensured delivery of the desired business value. Working closely with the product owners and stakeholders gave developers more influence on the product and what went into the release backlog.

Furthermore, there was no turnover during this project's 10-month release cycle. One developer, two weeks away from resigning to return to his hometown, enjoyed the new work environment so much that he put off leaving for more than a year. The developers came out of this project fresh and ready to work on the next assignment; we started the first iteration of the next release the day after completing this release.

Benefits to the business

Seeing slices implemented during each sprint review made it easier for product owners to focus on the highest-value items. In

many cases, seeing 50 percent of the feature implemented was sufficient to meet the desired business value and, as a result, the product owner could either drop the remaining requirements for that feature or reduce their priority in the release backlog. Also, rather than passing their decisions through layers of management, product owners were able to work closely with developers during the sprint.

Scrum also put the stakeholders much closer to the work, because they saw the product evolve during monthly sprint reviews. This gave them higher confidence in the team's ability to deliver specific value in the desired timeframe. It also made them more aware of the changing requirements on the release backlog. Scrum didn't necessarily give the stakeholders more control of the release backlog, but it let them provide more input to the product owners early in the development process.

Obstacles encountered

Although our first release of Scrum was a success, we did encounter a few problems along the way.

A main principle of Scrum is to show only "potentially shippable" increments at the sprint review. In the earlier sprints, we noticed that the teams were too eager to show their accomplishments during the sprint reviews and, in some cases, showed features that were either not fully tested or still had high-priority bugs pending. Instead of addressing these issues at the beginning of the following sprint, teams were eager to pick new items from the release backlog so they could show them at the following sprint review. Over time, the backlog of bugs grew and many of the features, while having impressive functionality built into them, weren't in shippable condition. We eventually had to dedicate a full sprint to fixing bugs to stabilize the product.

Another issue we encountered was focusing on short-term deliverables and, in some cases, losing sight of the code base's technical infrastructure and long-term maintainability. We have a stable and extensible architecture, which helped minimize this risk, but we need to make sure not to neglect this in future releases.

Also, stakeholders were concerned with the lack of metrics regarding the project's projected completion date. Stakeholders could clearly see the progress made from sprint to sprint, but they couldn't tell how much work remained on specific features or how much work remained until the release. Scrum can make it difficult to determine how far you are from release because, by definition, the requirements change from sprint to sprint. However, we could have better collected data from the teams and created reports for the stakeholders. The only report we generated was the release burndown, which didn't sufficiently communicate the remaining work.

Evaluating stakeholder feedback during sprint reviews created additional challenges. The stakeholders attended most sprint reviews and provided valuable feedback, some of which involved suggestions such as, "This feature looks great, but it would really be nice if we could also sort by this field." Developers tended to treat these suggestions as new requirements for the following sprint review, even though stakeholders generally didn't expect them to take the comments literally. In many cases, the product owners weren't even aware that these items were being addressed. The lesson learned was to be more disciplined about adding feedback to the product backlog so that the product owners have final say regarding the priority of these additional requirements.

There aren't many rules in Scrum, but you need to adhere to the ones that exist.

We've taken several steps to address these problem areas. First of all, we're stricter in enforcing all Scrum practices. There aren't many rules in Scrum, but you need to adhere to the ones that exist. We set clear criteria on what constitutes a completed feature, and only features that fit the criteria are shown to stakeholders at sprint reviews. We also assisted teams in doing better planning by helping them break out their work at the beginning of each iteration into truly shippable feature increments.

Furthermore, we gave priority to the code base's maintainability and extensibility by having the product owners add technical maintenance items to our current release backlog. We then worked with them to prioritize these items against the other features on the backlog. This process ensures that infrastructure maintenance items don't get lost in the mix and are addressed every sprint as part of the overall release.

Finally, we're currently reemphasizing the good engineering practices we had in place prior to using Scrum as well as reducing the number of bugs introduced early on in the development cy-

About the Authors

Bob Schatz is the vice president of development for Primavera Systems, where he leads the team that develops Primavera's software solutions for enterprise project, resource, and portfolio management. His research interests are in managing the development of large enterprise software systems. He received his BS in computer science from Temple University and is pursuing his MS in organizational dynamics from the University of Pennsylvania. Contact him at bobschatz@yahoo.com.

Ibrahim Abdelshafi is the director of programming for Primavera Systems. He's responsible for developing the architecture to support a multiplatform, highly scalable product that supports organizations around the world. His research interests include scaling the agile process to an enterprise development environment. He received his MS from Carnegie Mellon University and his MBA from the Wharton School at the University of Pennsylvania. Contact him at iabdelsh@primavera.com.

cle. To that end, we've started adopting some Extreme Programming practices, such as test-driven development and some pair programming. After seven sprints of implementing TDD, our defect count has dropped to less than 10 per team, which represents over a 75 percent improvement in defect rates relative to the previous release. As a result of these improvements, the teams can now be confident that they're always only one month away from a potential release—a foundation of the Scrum development process.

The biggest lesson we learned from Scrum is that, as a team, building software is a continuous learning process. We must have the discipline to honestly assess what we're doing and not be afraid to make changes. By consistently refocusing, we've been able to change our process, step by step, and we continue to make improvements every day.

References

1. S.C. Lundin, H. Paul, and J. Christensen, *Fish!: A Remarkable Way to Boost Morale and Improve Results*, Hyperion, 2000, p. 112.
2. K. Schwaber and M. Beedle, *Agile Software Development with Scrum*, Series in Agile Software Development, Prentice Hall, 2002, p. 158.