The Scrum Software Development Process for Small Teams

In today's software development environment, requirements often change during the product development life cycle to meet shifting business demands, creating endless headaches for development teams. We discuss our experience in implementing the Scrum software development process to address these concerns.

Linda Rising and Norman S. Janoff, *AG Communication Systems*

At AG Communication Systems, software development teams range in size from two to several hundred individuals. Intuitively, the development process that's appropriate for very large teams won't work well for tiny teams and vice versa. In our organization, process diversity means adopting a flexible approach to development processes so that each team can apply what works best. In experimenting with the Scrum software development process, we found that small teams

can be flexible and adaptable in defining and applying an appropriate variant of Scrum. This article describes our experience implementing this process.

Why Scrum?

As members of the Software Technology Group, our group is responsible for introducing new technologies and processes into our organization at AG Communication Systems in Phoenix, Arizona. We research new approaches and sponsor their introduction and growth. We also conduct development project checkups for ongoing projects and postmortems for completed ones.

In our periodic postmortems and checkups, we noticed some recurring problems for our New Business Opportunity (NBO) projects. Figure 1 lists some comments aris-

ing from projects at our organization that faced significant challenges.

In the new telecommunications market where our company operates, change is overwhelming. Software developers have always complained about changing requirements, but in traditional approaches they assumed they would understand the requirements before moving on to the next phase. In the current environment, however, project requirements might be unclear or unknown even as the project gets underway. Indeed, the market might not be defined—it might even be that no one clearly understands the product under development.

Most development teams respond with, "Make the chaos go away! Give us better requirements!" Unfortunately or not, chaos is the reality in this new business environ-

ment—as software developers we must learn how to meet customer needs and turn this chaos to our advantage.

In seeing that some of our NBO projects were more successful than others, we struggled to examine the postmortem data to learn the secrets of those successful projects. Here are some comments from successful teams:

- We did the first piece and then re-estimated—learn as you go!
- We held a short, daily meeting. Only those who had a need attended.
- The requirements document was high-level and open to interpretation, but we could always meet with the systems engineer when we needed help.

About this time, we discovered the Scrum Web sites (www.controlchaos.com/safe and www.jeffsutherland.org) and read a paper presenting patterns for using the Scrum software development process from the 1998 Pattern Language of Programming Conference.[1] From the Scrum Web sites we learned that Scrum is a process for incrementally building software in complex environments. Scrum provides empirical controls that allow the development to occur as close to the edge of chaos as the developing organization can tolerate.

The Scrum software development process described in this article developed in a collaboration between Advanced Development Methods and VMARK Software. Both companies were reporting breakthrough productivity. This approach overlapped significantly with what we saw in postmortems of successful projects. Our organization has long used patterns successfully, not just for design but for organization and process as well.[2,3] Many existing patterns supported what we learned about Scrum and saw in the postmortem data (see the "What's a Pattern?" sidebar). For example, Scrum advocates the use of small teams—no more than 10 team members. Jim Coplien's "Size the Organization" pattern recommends 10-person teams,[4] while Fred Brooks argues that "the small sharp team, which by common consensus shouldn't exceed 10 people."[5] Because Brooks, Coplien, and postmortems of successful teams all support this and many other Scrum tenets, we felt confident about undertaking a pilot project.

Although most of our NBO teams were small, some had grown to exceed the limit of 10. We thought initially that we could simply divide larger teams into collections of smaller subteams, each no larger than 10. We found that when the subteams are independent and the interfaces well defined, this works. When the overlap is considerable and the interfaces poorly understood, the

What's a Pattern?

In the late 1970s, two books appeared, written by Christopher Alexander and his building architect colleagues. *The Timeless Way of Building* and *A Pattern Language* presented Alexander's description of recurring problems in creating cities, towns, neighborhoods, and buildings, and the solutions to these problems.[1,2] He used the pattern form to document the problems and their solutions.

Each pattern describes a problem that occurs over and over again in our environment and then describes the core of the solution to that problem in such a way that you can use this solution a million times over without ever doing it the same way twice.[2]

Software professionals have also observed recurring problems and solutions in software engineering. There is evidence of patterns at all levels of software development, from high-level architecture to implementation, testing, and deployment. There is considerable work going on to apply this technology to software engineering. Patterns go a long way toward capturing what experts know, letting them share that knowledge with others.

On the surface, a pattern is simply a form of documentation. Pattern authors document solutions they have observed across many software projects. Experienced designers read these patterns and remark, "Sure, I've done that—many times!"

This kind of documentation captures knowledge, previously found only in the heads of experienced developers, in a form that is easily shared.

Patterns are not theoretical constructs created in an ivory tower; they are artifacts that have been discovered in multiple systems. In patterns, the solution is one that has been applied more than twice. This "rule of three" ensures that the pattern is documenting tried and true applications, not just a good idea without real use behind it.

The approach calls to mind the notions of cohesion and coupling Ed Yourdon and Larry Constantine developed during many late Friday afternoon postmortem sessions, discussing lessons learned from past projects.[3] The cohesion and coupling ideas captured qualities of real systems. The guidelines were not theoretical musings, but rested on observations of system capabilities that made life easier for developers and maintainers.

This is true for all patterns.

References
1. C.A. Alexander, The Timeless Way of Building, Oxford Univ. Press, New York, 1979.
2. C.A. Alexander et al., A Pattern Language, Oxford Univ. Press, New York, 1977.
3. E. Yourdon and L. Constantine, Structured Design, Prentice-Hall, Englewood Cliffs, N.J., 1978

The following reports summarize the experience with Scrum of three diverse software development teams at AG Communication Systems. The A-Team developed a new multiplatform simulator for the internal use of our GTD-5 EAX switching system software developers. The B-Team's focus was a new product in the small call-center market. The C-team developed a new feature for the GTD-5 EAX switching system.

A-Team

A-Team's new leader called for a project checkup. As a result of some problems that surfaced during the checkup, we offered to give a short Scrum presentation. The team decided to pilot a one-month sprint. To provide some on-the-job training in the approach and also to learn how this team would adapt the approach to its application, we sat in on the Scrum meetings.

At first, the team was uneasy about spending a lot of time in daily meetings and proposed holding sprint meetings every other day. The schedule of Monday, Wednesday, and Friday one week, followed by Tuesday and Thursday of the following week seemed to work best. During these meetings, the team began to grow together and display increasing involvement in and delight with others' successes. The team completed a successful sprint, with the planned components delivered on time.

The team began to cooperate almost immediately. The changes happened before our eyes. One plausible explanation is that our developers are superb engineers; they love to solve problems. When one team member shares an obstacle in the Scrum meeting, the entire team's resources come together to bear on that problem. Because the team is working together toward a shared goal, every team member must cooperate to reach that goal. The entire team immediately owns any one individual's problems.

Detailed problem solving does not happen in the meeting, of course—only an offer of help and an agreement to meet after the Scrum meeting. For example, a team member would say, "I ran into a problem." Typical responses would be, "I had that problem a couple of weeks ago. I can help with that. Let's talk offline;" "I know who is working in that area. I'll help you get in touch with them;" or "I'm having the same problem myself. Let's get together after the meeting and talk about it." We immediately noticed an increase in volunteerism in the team.

The celebration of small successes was also evident. At every meeting, as small tasks were completed and the team could see progress toward the sprint's goal, everyone rejoiced. On the sprint's last day, we could feel the excitement as the responses around the table were, "I finished my task! I finished my task! I finished my task!" They did it. They reached their goal together.

Figure A lists comments from the team leader at the end of the sprint.

Our biggest outcome from this pilot came in the definition of the Scrum master's task. We originally thought that this

---

Give it time to get started before expecting big results. It gets better as the team gains experience.

Tasks for a sprint must be well quantified and achievable within the sprint time period. Determine the sprint time by considering the tasks it contains.

Tasks for sprints must be assigned to one individual. If the task is shared, give one person the primary responsibility.

Sprint tasks might include all design-cycle phases. We set goals related to future product releases in addition to current development activity.

Scrum meetings need not be daily. Two or three times a week works for us.

The Scrum master must have the skill to run a short, tightly focused meeting.

Stick to the topics of the sprint. It's very easy to get off topic and extend what was supposed to be a 10 to 15 minute meeting into a half-hour or more.

Some people are not very good at planning their workload. Sprint goals are an effective tool for keeping people on track and aware of expectations.

I've noticed an increase in volunteerism within the team. They're taking an interest in each other's tasks and are more ready to help each other out.

The best part of Scrum meetings has been the problem resolution and clearing of obstacles. The meetings let the team take advantage of the group's experience and ideas.

Figure A. A-Team's team leader comments.

---

benefits are not as great. Clearly, this is not an approach for large, complex team structures, but we found that even small, isolated teams on a large project could make use of some elements of Scrum. This is true process diversity.

While conducting a project checkup, we thought that Scrum might address some of the issues facing the particular team. After we described the approach as we then understood it, the team decided to try Scrum and became our first pilot project. We didn't understand a lot about the approach, so we learned as we went along, as did the

role would be hard to fill: we needed someone who could facilitate a tight meeting, keep everyone on track and solve problems on the fly. The A-Team experience taught us that the team solves the problems—or most of them. The team still needs a capable Scrum master's guidance, but the team addresses and resolves most issues.

After this pilot project, we continued to present the Scrum overview at team meetings and to leaders of various business units: we took the Scrum story to both the grassroots level and to management. Whenever we detected a flicker of interest, we volunteered to work with the team to implement Scrum. Once the team began to use the process, we attended the Scrum meetings in the first sprint to provide guidance, lend moral support, and share successful experience across teams. We saw first hand what worked well for us and what teams learned about things that we could adapt for our environment. This was process diversity in action. Teams could modify Scrum to meet the team's needs. We could help them tailor the process based on what we had learned.

B-Team

Today's chaotic telecommunications business environment reflects back on development teams. A team's software developers might need to simultaneously support the present release in the field, develop new features for the next release, and estimate features for a future release. Management might also tap some of the developers for other projects.

The B-Team found itself in this situation. One week into their first sprint, the company made a major strategic change that significantly affected the backlog items committed for delivery. Should the team violate a cardinal Scrum principle by letting outside fac-

tors affect the sprint? Painful as it was, the team decided to modify the backlog to reflect the new strategy. They considered the outside influence a global change event, not the typical occurrence of a new or modified feature. We learned that rules for any approach must be considered in light of the overall business context. Any rule is just a rule, and if there are compelling business reasons, no process should stand in the way of making the best decision for the overall project.

Scrum works best when each developer focuses exclusively on the sprint. B-Team developers had numerous responsibilities in addition to the sprint's backlog items. As a consequence, the team overcommitted itself. In Scrum, overcommitment becomes visible very quickly. Rapid feedback helps keep developer commitment aligned with the goals of a time-boxed sprint.

B-Team's Scrum master used an Excel spreadsheet to track the backlog items, sorting the backlog by person and by priority, for each sprint. Some team members reported backlog completions in every Scrum meeting, while others could only report partial progress. To address the partial-progress reporting, the team tried to define task granularity to allow completion of at least two tasks per week. If tracking the backlog becomes too complicated, though, it suggests that perhaps the team is trying to make the process too difficult.

During the daily meetings, the Scrum master would call attention to backlog-item priority. This was especially helpful for new team members, who might have gone off in another direction. The B-Team completed a series of incremental sprints tht led to a product presently in market trials with a number of customers.

C-Team

We tried applying some of the Scrum development process for preliminary testing and bug fixing for a C-Team feature. The team was scheduled for four testing times in eight days, so they planned a half-hour Scrum meeting before and between each test time.

During the brief team meetings, team members could ensure that the proper load was used for the next scheduled test time. They redefined some testing procedures to make things go faster. The meetings let all testers hear what was planned and volunteer to work the next test time. The meeting also served to remind everyone of current load problems and events planned for the next couple of days.

The group as a whole decided the kind of testing to perform in the next test time, not just the tester who worked that test time. Added fixes were announced to the group.

The Scrum master announced the latest load schedule and informed everyone of information from outside the group that might have come in over e-mail but was missed by someone on the team. During one meeting, the Scrum master made a phone call to check the load status to help decide whether to go ahead with the next scheduled test time.

We saw that the regularly scheduled meeting gave the team an efficient way for sharing information and tracking progress. The team met its goal. The feature was ready for integration testing at the end of the sprint.

Even though all three teams implemented different instantiations of Scrum, they were all enthusiastic about the approach and felt that the experience had been a successful one. They were all willing to continue using it on their next release or project.

team. The pilot project and each succeeding project taught us more about what worked for each group and what components would address particular needs.

The "Experience Reports" sidebar describes three of our experiences in implementing the Scrum approach.

Scrum

A scrum is a team of eight individuals in Rugby. Everyone in the pack acts together with everyone else to move the ball down the field.

For those who know rugby, the image is

clear. Teams work as tight, integrated units with each team member playing a well-defined role and the whole team focusing on a single goal. In development teams, each team member must understand his or her role and the tasks for each increment. The entire team must have a single focus. The priorities must be clear. As we now describe, the Scrum development process facilitates this team focus.

How does Scrum work?

Scrum is a software development process for small teams. As Coplien and Brooks have shown, small teams that work independently are more effective..[4,5] Well-seasoned research in social dynamics supports this view. The phenomenon called *social loafing*, identified as early as 1927, empirically measured individual contribution to a group effort.[6] Using a gauge attached to a rope, the study found that individuals pulling the rope averaged 138.6 pounds of pressure. However, groups of three averaged only 2.5 times the individual rate, and groups of eight averaged less than four times the individual rate.

As in all projects, there must be an initial planning phase. During this phase, the project team must develop an architecture and identify a chief architect. During development, the team should be prepared to make changes to this architecture, but they need a plan, an architecture, and a chief architect at the start. The chief architect defines the development project's vision based on this architecture and ensures that vision's consistency throughout all the development phases. Coplien's pattern "Architect Controls Product" supports this understanding: "The Architect role should advise and control the Developer roles and communicate closely with the developers."[4]

After initial planning, a series of short development phases, or *sprints,* deliver the product incrementally. A sprint typically lasts one to four weeks. A closure phase usually completes product development.

The team tracks all currently identified tasks, capturing them in a list called the *backlog*. The backlog drives team activity. Before each sprint, the team updates the backlog and reprioritizes the tasks—each team signs up for a number of tasks and then executes a sprint. Individual team

> **The team can reduce delivered functionality during the sprint, but the delivery date cannot change.**

members agree to complete tasks they believe are feasible during each sprint. The team defines task granularity as appropriate, commonly specifying that a task must be completed in less than a week—larger tasks are more difficult to define and report.

During a sprint, no changes are allowe from outside the team.

What happens during a sprint?

A sprint produces a visible, usable, deliverable product that implements one or more user interactions with the system. The key idea behind each sprint is to deliver valuable functionality.

Each product increment builds on previous increments. The goal is to complete tasks by the sprint's delivery date. A sprint is time-boxed development, meaning that the end date for a sprint does not change. The team can reduce delivered functionality during the sprint, but the delivery date cannot change.

During the sprint, the team holds frequent (usually daily) Scrum meetings. These meetings address the observation made by Brooks: "How does a project get to be a year late? One day at a time."[5] When the team comes together for a short, daily meeting, any slip is immediately obvious to everyone. The meetings involve all team members, including those who telecommute. The meetings serve a team-building purpose and bring in even remote contributors, making them feel a part of the group and making their work visible to the rest of the team.

How do you plan and estimate?

Often, the marketing group or the customer will set the release date. With a release date set, the development and marketing groups must work together to provide the features with the highest value for the product's first release. Marketing should prioritize the features, while the product development group provides estimates for the effort. Marketing and product development must agree on the target set of features. If the product development group cannot deliver the requested features, the groups must negotiate a reduced set of features.

In negotiating features in the release, management must identify available developers for feature development. The num-

ber of developers for each team must meet the Scrum recommendation: no more than 10. Of course, you can have several teams. To complete negotiations with the marketing department, the product development group must have the required developers committed to the project for the identified time period.

Once negotiations between the marketing and product development groups are complete, the backlog is allocated to sprints in priority order. The product development group also establishes the target development environment and determines the risks associated with it. While the marketing department understands what the customer needs and the value to the customer, product development members understand the risk of new technologies, tools, or software processes. They identify and address these risks in the decision-making that leads to the feature allocation to sprints. In general, they address risky items in early sprints to allow time to recover if technical difficulties arise.

Planning proceeds relatively quickly because the initial assumptions will surely change as sprints deliver incremental functionality.

Who leads the team?

The *Scrum master* leads the Scrum meetings, identifies the initial backlog to be completed in the sprint, and empirically measures progress toward the goal of delivering this incremental set of product functionality. The Scrum master ensures that everyone makes progress, records the decisions made at the meeting and tracks action items, and keeps the Scrum meetings short and focused.

The Scrum master works constantly to reduce product risk through the incremental delivery of features, rapid response to development obstacles, and continual tracking of the delivery of backlog items.

What happens during a Scrum meeting?

Each team member must answer three questions:

1. What have you completed, relative to the backlog, since the last Scrum meeting?
2. What obstacles got in your way of completing this work?

> **The team can trim functionality but must meet the promised delivery date.**

3. What specific things do you plan to accomplish, relative to the backlog, between now and the next Scrum meeting?

The Scrum meeting should last 15 to 30 minutes, which provides enough time to address obstacles, but does not allow time to brainstorm a solution. All discussion other than responses to the three questions is deferred to a later meeting involving only those actually affected by the discussion. The goals of the Scrum meeting include

- focusing the effort of developers on the backlog items,
- communicating the priorities of backlog items to team members,
- keeping everyone informed of team progress and obstacles,
- resolving obstacles as quickly as possible,
- tracking progress in delivering the backlog functionality, and
- addressing and minimizing project risk.

What happens at the end of a sprint?

At the end of a sprint, the team produces an increment that builds on previous increments. The team can trim functionality but must meet the promised delivery date.

After each sprint, all project teams meet with all stakeholders, including high-level management, customers, and customer representatives. All new information from the sprint just completed is reported. At this meeting, *anything* can be changed. Work can be added, eliminated, or reprioritized. Customer input shapes priority-setting activities. Items that are important to the customer have the highest priority.

New plans and estimates are made following the same process discussed under "How do you plan and estimate?" Assignments are then made to teams for the next sprint. As sprints finish, estimates become better as planners see what each team has produced in previous sprints. With the small time increments, the planners must be careful in their estimation, because, as Brooks states, "Extrapolation of times for the hundred-yard dash shows that a man can run a mile in less than three minutes."[5]

Because each sprint produces a visible, usable, increment, product delivery can take place after any sprint. In fact, delivery of a

sort does take place and feedback from the business and marketing side and from customers is a reaction to the current increment's delivery. Typically, final delivery takes place after a wrap-up phase that is run as a final sprint.

The organization can make one very important decision at the end of a sprint: whether to continue product development. Given what was delivered during the last sprint and the current state of the market, the stakeholder meeting should address the question, "Should this project continue?" This is a business decision and must be made after considering all the technical and marketing issues.

As a result of the interaction of small teams in small, focused development cycles,

- the product becomes a series of manageable chunks,
- progress is made, even when requirements are not stable,
- everything is visible to everyone,
- team communication improves,
- the team shares successes along the way and at the end,
- customers see on-time delivery of increments,
- customers obtain frequent feedback on how the product actually works,
- a relationship with the customer develops, trust builds, and knowledge grows, and
- a culture is created where everyone expects the project to succeed.

Most of Scrum's elements are not new. It is basically an incremental, time-boxed development with an added twist: the frequent meetings where the three questions are asked. Barry Boehm's spiral model certainly addresses the element of risk in software development.[7] The essential ideas behind the spiral model are exactly the same as those in Scrum—just speeded up! A sprint typically lasts one to four weeks and an entire project only a few months. During that short time, the original customer might leave and a new one arrive—with new constraints. In addition, technology might change out from under the team.

We've found that Scrum is appropriate for projects where we can't define requirements up front and chaotic conditions are anticipated throughout the product development life cycle. At AG Communication Systems, we continue to evolve the approach as our development groups gain experience with it. ⊛

Acknowledgments

References

1. M. Beedle et al., "SCRUM: An Extension Pattern Language for Hyperproductive Software Development," *Pattern Languages of Program Design 4*, N. Harrison, B. Foote, and H. Rohnert, eds., Addison-Wesley, Reading, Mass., 2000, pp. 637–651.
2. B. Goldfedder and L. Rising, "A Training Experience with Patterns," *Comm. ACM*, Vol. 39, No. 10, Oct. 1996, pp. 60–64.
3. N.S. Janoff, "Organizational Patterns at AG Communication Systems," *The Patterns Handbook*, L. Rising, ed., Cambridge Univ. Press, New York, 1998, pp. 131–138.
4. J.O. Coplien, "A Generative Development-Process Pattern Language," *Pattern Languages of Program Design*, J.O. Coplien and D.C. Schmidt, eds., Addison-Wesley, New York, 1995, pp. 184–237.
5. F.P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, New York, 1995.
6. L. Fried, "When Bigger Is not Better: Productivity and Team Size in Software Development," *Software Engineering Tools, Techniques, Practice*, Vol. 2, No. 1, pp. 15–25.
7. B.W. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer*, Vol. 21, No. 5, May 1988, pp. 61–72.

About the Authors

Linda Rising is the editor of *The Pattern Almanac 2000* and *A Patterns Handbook* (Cambridge Univ. Press) and was also the feature editor for a special issue of *IEEE Communications* on design patterns in communications software, which appeared in April 1999. She is interested in patterns and processes for software development and has worked in the telecommunications, avionics, and strategic weapons systems industries. She has a PhD from Arizona State University in the area of object-based design metrics and is a member of the ACM and IEEE Computer Society.

Norman S. Janoff is a software project engineer at AG Communication Systems in Phoenix, Arizona. He has worked as a software manager on a large telecommunication switching system and as a software engineer. His current research interests include software processes and software metrics. He received a BS in electrical engineering from the University of Michigan, an MS in electrical engineering from the University of Illinois, and an MBA from the University of Chicago.

Contact the authors at AG Communication Systems, 2500 W. Utopia Rd., Phoenix, AZ, 85027; risingl@acm.org, janoffn@agcs.com.