Scrum: A Pattern Language for Hyperproductive Software Development

Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland

Can a repeatable and defined process really exist for software development? Some think this is not only possible but necessary, for example, those who favor the CMM (Capability Maturity Model) approach to software development [71].

However, many of us doing work in the trenches have found over time that the *repeatable* or *defined* process approach makes many incorrect assumptions, such as the following:

- ***Repeatable/defined problem.*** A repeatable/defined process assumes that there is a step or steps to capture requirements , but in most cases, it is not possible to define the requirements of an application like that because they are either not well defined or they keep changing.
- ***Repeatable/defined solution.*** A repeatable/defined process assume that an architecture can be fully specified, but in reality it is evolved, partly because of missing or changing requirements (as described above), and partly because of the creative process involved in designing new software structures.
- ***Repeatable/defined developers.*** The capabilities of a software developer vary widely, so that a process that works for one developer may not work for another one.
- ***Repeatable/defined organizational Environment.*** Schedule pressure, priorities (e.g. quality vs. price vs. manpower), client behavior, and so on are never repeatable, and because they are highly subjective, they are very hard to define.

The problem with these assumptions is that these variables do have large variances. In real life projects there are always large dynamic variations that can have a great deal of impact on the overall project. For example, newly found changes in the requirements during an application implementation—a typical occurrence—may affect drastically a project's schedule that assumed that *all* the requirements would be captured up front. However, removing this uncertainty is nearly impossible because of the nearly universal sources of requirements change: business requirements driven changes, usability driven changes, re-prioritization driven changes, testing driven changes, and so forth. This issue cannot be solved through improved methods for identifying the user requirements. Instead it calls for a more complex process of generating fundamentally new operating alternatives.

In other words, once we accept that these dynamic variabilities do exist, we clearly need more adaptive ways to build software. However, what we fear is that most current methods do not allow us to build *soft enough* software: present methods and design paradigms seem to inhibit adaptability. Therefore the majority of software practitioners nowadays tend to become experts at what they can *specify in advance*, working with the unstated belief that there exists an optimal solution that can be planned a priori. Once technology is adopted by an organization, it oten becomes a constraining structure that in part shapes the action space of the user. Thus we build software too much like we build hardware—as if it were difficult to change, as if it has to be difficult to change. In many organizations, "The system requires it" or the "System does not allow it" have become accepted (and often unavoidable) justifications for human behavior before and after the system is released to production.

In contrast, Scrum allows us to build *softer* software, so there is no need to write full requirements up fronts. Since the users do not know what is possible, they will ask for the pre-tech-paper solution that they perceive to be possible ("looking at the rearview mirror"). But in truth, not even the software developers know fully what can be built beforehand. Therefore, the user has no concept of what is possible before he or she can feel it or touch it. As such, The Scrum patterns presented here offer a collection of empirical techniques that assume up front the existence of uncertainty but that provide practical and specific techniques to tame it. These techniques are rooted in complexity management, that is, in self-organization, management of empirical processes, and knowledge creation.

In that sense, Scrum is not only a "parallel iterative and incremental" development method, it is also an "adaptive" software development method.

How does Scrum Work?

Scrum's goal is to deliver as much quality software as possible within a series (three to eight) of short time boxes (fixed-time intervals) called Sprints that typical last about a month.

Each stage in the development cycle (Requirements, Analysis, Design, Evolution, and Delivery) is now mapped to a Sprint or series of Sprints. The traditional software development stages are retained primarily for convenience tracking milestones. So, for example, the Requirements stage may use one Sprint, including the delivery of a prototype. The Analysis and Design stage may take one Sprint each, while the Evolution stage may take anywhere from three to five Sprints.

Editors Note: In recent years, release cycles have shortened to three months or less for most software products. Requirements are specified just enough and just in time to be ready at the start of the Sprint cycle. Sprints produce working software for review at the end of every Sprint. As a result, Analysis, Design, and Evolution occur in every Sprint. Sprint cycles in many companies have been shortened to two weeks or less. In the best companies, delivery is included in every Sprint [40].

Unlike a repeatable and defined process approach, in Scrum there is no predefined process within a Sprint. Instead, Scrum meetings drive the completion of the allocated activities.

Each sprint operates on a number of work items called a Backlog. As a rule, no more items are externally added into the Backlog within a Sprint. Internal items resulting form the original pre-allocated Backlog can be added to it. The goal of a sprint is to complete as much quality software as possible, but typically less software is delivered in practice.

The end result is that there are non-perfect releases delivered every Sprint.

During a Sprint, Scrum Meetings are held daily to determine the following:

- Items completed since the last Scrum meeting..
- Issues or blocks that need to be resolved. (The ScrumMaster is a team leader role responsible for resolving the blocks.)
- New assignments the team should complete before the next Scrum meeting.

Scrum Meetings allow the development team to “socialize the team members’ knowledge” as well as produce a deep cultural transcendence. This “knowledge socialization” promotes a self-organized team structure within which the development process evolves on a daily basis.

At the end of each Sprint there is a demonstration to:

- Show the customer what’s going on.
- Give the developer a sense of accomplishment.
- Integrate and test a reasonable portion of the software being developed.
- Ensure *real progress*, that is, the reduction of Backlog, not just the production of more paper/hours spent.

After gathering and reprioritizing leftover and new tasks, a new Backlog is formed and a new Sprint starts. Potentially, many other organization and process patterns may be used in combination with the Scrum patterns.

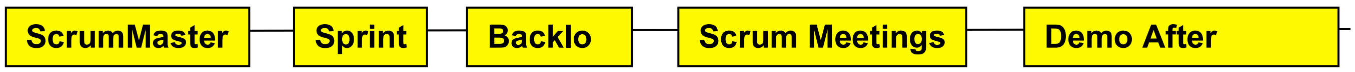

Figure: *The Scrum pattern language*

The Patterns

Sprint

Context

You are a software developer or a coach managing a software development team where there is a high percentage of discovery, creativity, or testing involved.

You are building or expanding systems, which allow partitioning of work, with clean interfacing, components, or objects.

Problem

You want to balance the needs of developers to work undisturbed and the need of management and the customer to see real progress, as well as control the direction of that progress throughout the project.

Forces

- Developers need time to work undisturbed, but they need support for logistics; management and users need to be convinced that real progress is being made.
- Often, by the time systems are delivered, they are obsolete or require major changes. The problem is that input from the environment is collected mostly at the start of the project, while the user learns mostly by using the system or intermediate releases.
- It is often assumed that the development process is a well-understood approach that can be planned and estimated. If a project fails, that is considered proof that the development process needs more rigor. These step by step approaches, however, don't work because they do not cope with the unpredictabilties, both human and technical, in system development. Therefore, at the beginning of a project it is impossible to make a complete, detailed specification, plan, or schedule because of the many uncertainties involved.
- Overhead is often created to prove that a process is on track. Current process automation adds administrative work for managers and developers and often results in marginally used development processes that become disk-ware. (Misfit: Activity is not synonymous with results. More often that not, a project plan shows activities but fails to ensure real progress or results.

Solution

Divide the project in Sprints. A Sprint is a period of approximately 30 days in which an agreed amount of work will be performed to create a deliverable. Each Sprint takes a pre-allocated amount of work from the Backlog, and it is assigned to Sprints by priority and by approximation of what can be accomplished during the Sprint's length. In general, chunks of high cohesion and low coupling are selected—either horizontal or vertical "packets," that is, vertical or horizontal components.

As a rule, nothing is added externally to the allocated Sprint Backlog during the Sprint. External additions are only added to the global Backlog, but blocks (unresolved issues) resulting from the Sprint can be added to the allocated Sprint Backlog. A Sprint end with a demonstration (Demo After Sprint) of new functionality.

This gives the developers space to be creative, to learn by exploring the design space and by doing actual work. Undisturbed by outside interruptions, they are free to adapt their ways of working using opportunities and insights. At the same time, this keeps management and other project stakeholders confident by showing real progress instead of documents and reports produced as *proof* of progress.

The net result is that each sprint produces a visible and usable deliverable that is shown to the users at the demo (Demo After Sprint). An increment can be either intermediate of shippable, but it should stand on its own. The goal of a Sprint is to complete as much quality software as possible and to ensure real progress, not paper milestones as alibis.

Editors Note: This strategy had a huge impact on global software development. Iterations that demonstrate early working software in order to incorporate real-time user feedback have increased project success industry-wide from 16.2% in 1994 to 35% in 2006. This increased the industry-wide return on dollar invested in software from 25 cents in 1998 to 59 cents in 2006 for a compound annual growth rate of 24% [107].

Rationale

- The fact that no items are added to the Backlog externally allows development to progress "full speed ahead," without needing to think about changes in direction.
- The fact that developers are not "tested" during the Sprint is empowering.
- The ability to choose a process per Sprint is empowering and enables adaptation to changing circumstances (different developers, different project phases, more knowledge, etc.)
- Sprints are short; therefore, the problem of completing a Sprint is much simpler that that of completing a project. It is easier to take up this smaller challenge.
- Developers get feedback frequently (at the end of each Sprint). They can therefore feel their successes (and failures) without compromising the whole project.
- Management has full control—it can completely change direction at the end of each Sprint.
- The end users are deeply involved throughout the development of the application through the Demos after the Sprints, but they are not allowed to interfere with the day-to-day activities. Thus ownership and direction still belong to the users but without their constant interference.
- Project status is visible since the Sprint produces working code.

Known Uses

At Argo, the Flemish department of education, we have been using Sprints since January 1997 on a large number of end-user projects and for the development of a framework for database, document management, and workflow. The Backlog is divided in sprints that last about a month. At the end of each Sprint, a working Smalltalk image is delivered with integration of all current applications. The team meets daily in Scrum Meetings, and Backlog is re-prioritized after the Demo in a monthly meeting with the steering committee.

Resulting Context

The result is a high degree of "effective ownership" by the participants, including users who stay involved through the Demos and the prioritizing of the Backlog. "Effective ownership" in this case means both empowerment and the involvement of all the participants.

At the end of a Sprint, we have the best approximation of what was planned at the start. In a review session, the supervisors have the opportunity to change the planning for the future. The project is totally flexible at this point. Target, product, delivery date, and cost can be redefined.

With Scrum we get a large amount of post-planning flexibility (for both customer and developer).

It may become clear in the daily Scrum Meetings throughout the Sprint that some team members are losing time at non- or less productive tasks. Alternatively, it may also become clear that people need more time for their tasks than originally allocated by management. Developers may turn out to be less competent or experienced at the allocated task than assumed, or they may be involved in political or power struggles. The high visibility of scrum, however, allows us to deal with these problems. This is the strength of the Scrum method manifested through the Scrum Meetings and the Sprints.

Difficulties in grouping Backlog for a Sprint may indicate that priorities are not clear to management or to the customer.

Backlog

Context (From: Sprint)

You are connected to a software project or any other project that is chaotic in nature that needs information on what to do next.

Problem

What is the best way to organize the work to be done next and at any stage of the project?

Forces

Traditional planning methods like Pert and Gantt assume that you know in advance all the tasks, all their dependencies, all task durations, and all available resources. These assumptions are wrong if the project involves any learning, discovery, creativity, or adaptation.

Solution

Use a Backlog to organize the work of a Scrum team.

The Backlog is a prioritized list. The highest priority Backlog item will be worked on first, the lowest priority Backlog item will be worked on last. No feature, addition, or enhancement to a product is worth fighting over; it is simply either more important or less important at any time to the success and relevance of the product.

Backlog is the work to be performed on a product. Completion of the work will transform the product from its current form into its vision. But in Scrum, the Backlog evolves as the product and the environment in which it will be used evolves. The Backlog is dynamic, constantly changed by management to ensure that the product defined by completing the Backlog is the most appropriate, competitive, useful product possible.

There are many sources for the Backlog list. Product marketing adds work that will fulfill their vision of the product. Sales adds work that will generate new sales or extend the usefulness to the installed base. Technology adds work that will ensure the product uses the most innovative and productive technology. Development adds work to enhance product functions. Customer support adds work to correct underlying product defects.

Only one person prioritizes work. This person is responsible for meeting the product vision. The title usually is product manager or product marketing manager. If anyone wants the priority for work changed, they have to convince this person to change that priority. The highest priority Backlog has the most definition. It is also prioritized with an eye toward dependencies.

Depending on how quickly products are needed in the marketplace and the finances of the organization, one or more Scrum Teams work on a product’s Backlog. As a Scrum Team is available (newly formed or just finished a Sprint) to work on the Backlog, the team meets with the product manager. Focusing on the highest priority Backlog, the team selects a subset of the Backlog the team believes it can complete within a Sprint iteration (30 days or less). In doing so, the Scrum Team may alter the Backlog priority by selecting a Backlog that is mutually supportive, that is, one that can be worked on at once more easily than by waiting. Examples are multiple work items that require developing a common module or interface and that make sense to include in one Sprint.

The team selects a cohesive group of top priority Backlog items that, once completed, will have reached an objective, or milestone. This is stated as the Sprint's objective. During the Sprint, the team is free to not do work as long as this objective is reached.

The team now decomposes the selected Backlog into tasks. These tasks are discrete pieces of work that various team members sign up to do. Tasks are performed to complete Backlog to reach the Sprint objective.

Resulting Context

Projec work is identified dynamically and prioritized according to:

1. The customer needs
2. What the team can do

Scrum Meetings

Context (From: Backlog)

You are a software developer or a coach managing a software development team where there is a high percentage of discovery, creativity, or testing involved. An example is a first time delivery where the problem has to be specified, an object model has to be created, or new or changing technologies are being used.

Activities such as scientific research, innovation, invention, architecture, engineering and a myriad of other business situations may also exhibit this behavior.

You may also be a "knowledge worker," an engineer, a writer, a research scientist, or an artist, or a coach or manager who is overseeing the activities of a team in these environments.

Problem

What is the best way to control an empirical and unpredictable process such as software development, scientific research, artistic projects, or innovative designs where it is hard to define the artifacts to be produced and the processes to achieve them?

Forces

Estimation

- Accurate estimation for activities involving discovery, creativity, or testing is difficult because it typically involves large variances, and because small differences in circumstances may cause significant differences in results. These uncertainties come in at least five flavors:
  1. Requirements are not well understood.
  2. Architectural dependencies are not easy to understand and are constantly changing.
  3. There may be unforeseen challenges with the technology. Even if the challenges are know in advance, their solutions and related effort are not known.
  4. There may be bugs that are hard to resolve in the software; therefore, it is typical to see project estimates that are several orders of magnitude off. You can't "plan bugs," you can only plan bug handling and provide appropriate prevention schemes based on the possibility of unexpected bugs.

     Example: You Got the Wrong Number. In projects with new or changing requirements, a new architecture, new or changing technologies, and difficult bugs to weed out, it is typical to see project estimates that are off by several orders of magnitude.
  5. On the other hand, estimation *is* important. One must be able to determine what are the future tasks within some time horizon and prepare resources in advance.

Planning

- Planning and reprioritizing tasks takes time. Involving workers in time planning meetings decreases productivity. Moreover, if the system is chaotic, no amount of planning can produce uncertainties.

  Example: Paralysis by Planning. Some projects that waste everyone's time in planning everything to an extreme detail but are never able to meet the plans.
- A plan that is too detailed become large and is hard to follow; the larger the plan is, the more errors it will contain (or at the very least the cost of verifying its correctness grows).

  Example: The Master Plan Is a Great Big Lie. Many projects that try to follow a master plan fall into the trap of actually believing their inaccuracies and often face disappointment when their expectations are not met.

- No planning at all increases uncertainty among team members and eventually damages morale.
  Example: Lost Vision. Projects that never schedule anything tend to lose control over their expectations. Without some schedule pressure no one will do anything, and worse, it will become difficult to integrate the different parts being worked on independently.

Tracking

- Too much monitoring wastes time and suffocates developers.
  Example: Measured to Death. Projects that waste everybody's time in tracking everything to an extreme detail but are never able to meet the plans. (You measured the tire pressure until all the air was out!)
- Tracking does not increase the certainty of indicators because of the chaotic nature of the system. In fact, trying to control normal variations of a system will cause wide oscillations of the system, rendering it more chaotic.
- Too much data is meaningless—the Needle in the Haystack Syndrome.
- Not enough monitoring leads to blocks and possible idle time between assignments.
  Example: What Happened Here? Projects that never track anything tend to lose control over what is being done. Eventually no one really knows what has been done.

Solution

To provide for accurate estimates, plans, and appropriate tracking, meet with the team members for a short time (~15 minutes) in a daily Scrum Meeting, where the only activity is asking each participant the following three questions:

1. What have you worked on since the last Scrum Meeting? The ScrumMaster logs the tasks that have been completed and those that remain undone.
2. What blocks, if any, have you found in performing your tasks within the last 24 hours? The ScrumMaster logs all blocks and later finds a way to resolve the blocks.
3. What will you be working on in the next 24 hours? The ScrumMaster helps the team members choose the appropriate tasks to work on with the help of the Architect. Because the tasks are scheduled on a 24-hour basis, the tasks are typically small (Small Assignments).

This will provide you with more accurate estimates, short-term plans, appropriate tracking, an correcting mechanisms to react to changes and adapt every 24 hours.

Scrum Meetings typically take place at the same time and place every day, so they also serve to build a strong culture. As such, Scrum meetings are rituals that enhance the socialization of status, issues, and plans for the team. The ScrumMaster leads the meetings and logs all the tasks from every member of the team into a global project Backlog. He also logs every block and resolves each block while the developers work on other assignments.

Editors note: The Scrum Board has emerged as a best practice for a team to manage their own tasks. Teams meet in front of the Board which has multiple columns. The first column has User Stories from the Product Backlog (features to be delivered) on large cards prioritized in order of business value. At the start of the Sprint, the tasks to be accomplished for a User Story are in the left column as small cards. Each day developers move tasks to an "In Progress" column, then to a "Validation" column, then to a "Done" column. Estimates are updated on tasks daily and the Burndown Chart can easily be calculated and posted to the board [108].

The blocks logged by the ScrumMaster are now known as the "Impediment List" which needs to be prioritized. The block which is the most critical constraint to system throughput should be at the top of the list and the ScrumMaster should work on that one first. Tuning a development project is similar to tuning a computer system. It may not be obvious where the critical constraint lies and careful analysis may be required. The main choke point must be found and fixed first. The development system as a whole should then be allowed to stabilize and measured. The next critical block after restabilization may be in an unexpected place. That should be fixed next. Fixing too many things at once generates waste by fixing constraints that have minimal impact on throughput. This uses critical resources to change things that do not dramatically improve velocity. It makes it difficult to clarify system dynamics and tires out and demotivates the team, management, and the company.

Scrum meetings not only schedule tasks for developers, but can and should schedule activities for everyone involved in the project, such as integration personnel dedicated to configuration management, architects, ScrumMasters, or a QA team.

Scrum Meetings allow knowledge workers to accomplish mid-term goals typically allocated in Sprints that last a month or less.

Scrum Meetings can also be held by self-directed teams. In that case, someone is designated as the scribe and logs the completed activities of the Backlog and the existing blocks. All activities from the Backlog and the blocks and then distributed among the team for resolution.

The format of the Backlog and the blocks can also vary, ranging from a list of items on a piece of paper, to software representations of it over the Internet/Intranet [17]. The Scrum Meeting's frequency can be adjusted and typically ranges between 2 and 48 hours.

These meetings are often held standing up. This ensures that the meetings are kept short and to the point.

Rationale

It is very easy to over- or under-estimate, which leads either to idle developer time or to delays in completion of an assignment. Therefore, it is better to frequently *sample* the status of small assignments. Projects with a high degree of unpredictability cannot use traditional project planning techniques such as Gantt or PERT charts *only*, because the rate of change of what is being analyzed, accomplished, or created is too high. Instead, constant reprioritization of tasks offers an adaptive mechanism that provides sampling of systemic knowledge over short periods of time.

Scrum Meetings help also in the creation of an "anticipating culture" [103] because they encourage these productive values:

- They increase the overall sense of urgency.
- They promote the sharing of knowledge.
- They encourage dense communications.
- They facilitate honesty among developers since everyone has to give a daily status.

This same mechanism encourages team members to socialize, externalize, internalize, and combine technical knowledge on an ongoing basis, thus allowing technical expertise to become community property for the community of practice [109]. Scrum Meetings are therefore rituals with deep cultural transcendence. Meeting at the same place at the same time, and with the same people, enhances a feeling of belonging and creates the habit of sharing knowledge.

Seen from the System Dynamics point of view [88], software development has a scheduling problem because the nature of programming assignments is rather probabilistic. Estimates are hard to come by because:

- Inexperienced developers, managers, and architects are involved in making the estimates.
- There are typically interlocking architectural dependencies that are hard to manage.
- There are unknown or poorly documented requirements.
- There are unforeseen technical challenges.

As a consequence, the software development becomes a chaotic *beer game*, where it is hard to estimate and control the *inventory* of available developer's time, unless increased monitoring of small assignments is implemented [88, 110]. In that sense the Scrum Meeting becomes the equivalent of the thermometer that constantly samples the team's temperature.

From the Complexity Theory perspective [104, 105], Scrum allows flocking by forcing a faster agent interaction, therefore accelerating the process of self-organization because it shifts resources opportunistically through the daily Scrum Meetings.

This is understandable, because the relaxation of the self-organized multi-agent system is proportional to the average exchange among agents per unit of time. And in fact, the "interaction rate" is one of the levers one can push to control "emergent" behavior—it is like adding an enzyme or catalyst to a chemical reaction.

In Scrum this means increasings the frequency of the Scrum Meetings, and allowing more *hyperlinks* as described below, but up to an optimal upper-frequency bound on the Scrum Meetings (meetings/time), and up to an optimal upper bound on the hyperlinks or the Scrum Team members. Otherwise the organization spends too much time socializing knowledge, instead of performing tasks.

Known Uses

(Mike Beedle) At Nike Securities in Chicago we have been using Scrum Meetings since February 1997 to run all of our projects including BPR and software development. Everyone involved in these projects receives a week of training in Scrum techniques.

(Yonat Sharon) At Elementrix Technologies we had a project that was running way late after about five months of development. Only a small part (about 20 percent) was completed, and even this part had too many bugs. The project manager started running bi-daily short status meetings (none of us was familiar with the term Scrum back then). In the following month, the entire project was completed and the quality had risen sharply. Two weeks later, a beta version was out. The meetings were discontinued, and the project hardly progressed since. I don't think the success of the project can be attributed to the Scrum Meetings alone, but they did have a big part in this achievement.

One of my software team leaders at Rafael implemented a variation of Scrum Meetings. He would visit each developer once a day, and ask him the three questions; he also managed a Backlog. This does not have the team building effects, but it does provide the frequent sampling.

C3 and Vcaps projects (described on wiki [111]) also do this. (BTW, I adopted this name in Hebrew, since in Hebrew "meeting" is "sitting," and so we say "standup sitting".)

Resulting Context

The application of this pattern leads to:

- Highly visible project status

- Highly visible individual productivity
- Less time wasted because of blocks
- Less time wasted because of waiting for someone else
- Increased team socialization

Conclusion

Scrum is a knowledge creating process with a high level of information sharing during the whole cycle and work progress.

The key to Scrum is pinning down the date at which we want completion for production or release, prioritizing functionality, identifying available resources, and making major decisions about architecture. Compared to more traditional methodologies, the planning phase is kept short since we know that events will require changes to initial plans and methods. Scrum uses an empirical approach to development where interaction with the environment is not only allowed but encouraged. Changing scope, technology, and functionality are expected; and continuous information sharing and feedback keeps performance and trust high.

Its application also generates a strong culture with well-defined roles and relationships, with meaningful and transcending rituals.

Acknowledgements

We would like to thank all of the Scrum users and reviewers from whom we have received feedback over the years. Also, we thank all of the members of the Chicago Patterns Group that attended an early review session of the Scrum Meeting pattern (especially Brad Appleton, Joe Seda, and Bob Haugen). Finally we thank our PloP'98 shepherd, Linda rising, for providing us comments and guidance to make our paper better.

(Personal acknowledgement from Mike Beedle.) I'd like to thank both Jeff Sutherland [112] and Ken Schwaber [113] for adapting the Scrum techniques to software in the early 1990s, and for sharing their findings with me. Scrum has made a significant contribution to the software projects in which I used the technique.