Controlled Chaos : Living on the Edge

The Origins of Scrum

The Scrum software development process described in this article arose from shared concerns between Advanced Development Methods (ADM) and VMARK Software (VMARK). ADM produces process automation software. VMARK produces object-oriented software development environments. Both companies were concerned over the lack of breakthrough productivity being reported in object-oriented development projects. Both ADM's and VMARK's products are built using OO, and breakthrough productivity had been experienced in both companies.

We were particularly concerned that OO and component-based development were being hindered by currently available development processes. We wanted to ensure that the processes used by our organizations, and other ISV's, were available to our customers and the software development community.

After further analyzing how successful ISV's build software, we found that our development processes are very similar. We all use empirical processes. We all control the empirical processes through quantitative measures. We refer to the results of these processes as "the best possible software" or "good enough software." We all were able to relate stories of peers who used empirical processes without controls, and who were now in the sin bin of failed ISV's.

The ISV's understood that building systems is an art guided by rules of thumb and tips and techniques. A methodologist at one ISV said *"...you can't expect a process to tell you everything to do. Writing software is a creative process, like painting or writing or architecture... ... [a process] supplies a framework that tells how to go about it and identifies the places where creativity is needed. But you still have to supply the creativity...."*

Our first step in formalizing the ISV empirical development process into Scrum was researching why the ISV empirical software development approach delivers breakthrough productivity, and why the defined process advocated by the Software Engineering Institute's Capability Maturity Model (SEI's CMM) doesn't. After all, CMM promises those most elusive of things-predictability, quality, repeatability, and improvability.

Scientists Give Their Opinion

Why do the defined processes advocated by SEI CMM not measurably deliver? We posed this question to scientists at DuPont Chemical's Advanced Research Facility, where research into biochemical processes is applied to process automation.

The scientists inspected the systems development process. They concluded that many of the processes, rather than being repeatable, defined, and predictable, were unpredictable and unrepeatable. With that, the scientists explained the difference between predictable (defined) and unpredictable (empirical).

If a process can be fully defined, with all things known about it so that it can be designed and run repeatably with predictable results , it is known as a defined process, and it can be subjected to automation. If all things about a process aren't fully known-only what generally happens when you mix these inputs and what to measure and control to get the desired output-these are called empirical processes.

A defined process is predictable; it performs the same every time. An empirical process requires close watching and control, with frequent intervention. It is chaotic and unrepeatable, requiring constant measurement and control through intelligent monitoring.

Models of empirical processes are derived by categorizing observed inputs and outputs and defining the controls that cause them to occur within prescribed bounds. Empirical process modeling involves constructing a process model strictly from experimentally obtained input/output data, with no recourse to any laws concerning the fundamental nature and properties of the system. No *a priori* knowledge about the process is necessary; a process is treated like a black box.

The scientists further stated, "We are most amazed that your industry treats treat these ill-formed processes as defined, and performs them without controls despite their irregular nature. If chemical processes that we don't understand completely were handled in the same way, we would get very unpredictable results."

We confirmed that we also get unpredictable results, such as undelivered systems, delivered systems that are unusable by the customer, and the systems development process going on interminably without adequate output generated.

Regarding the systems development process, the scientists concluded that they are mostly empirical, because :

- Applicable first principles are not present
- The process is only beginning to be understood
- The process is complex
- The process is changing and unpredictable

Scrum Empirical Development Process

The Scrum software development process uses an iterative, incremental approach. Interaction with the environment (technical, competitive, and user) is allowed, which will change the project scope, technology, functionality, cost, and schedule whenever required. Controls are used to measure and manage the impact.

Scrum accepts that the development process is unpredictable. The product is the best possible software, factoring in cost, functionality, timing, and quality. This concept has been discussed by James Bach of Software Testing Laboratories in various articles, including "The Challenge of Good Enough Software" .

Scrum formalizes the empirical "do what it takes" software development process used today by many successful ISV's. The empirical approach has been used by these ISV's to cope with the otherwise overwhelming degree of complexity and uncertainty-chaos-in which they develop products. The chaos exists not only in the marketplace where they hope to sell the products, but in the technology that they employ to design and construct these products.

These ISV's succeed and thrive amidst chaos. How? Is their development approach applicable to IS organizations. Is it predictable and controlled? Can it be used in large and small projects? Is it scaleable? Does it work for all types of development?

Scrum - Characteristics and Rules

Scrum encapsulates what works at the best ISV's., Several of the characteristics that guide Microsoft's controlled-chaos approach to the development process are inherent in Scrum :

- "It breaks down large products into manageable chunks - a few product features that small teams can create in a few months.
- It enables project to proceed systematically even when team members cannot determine a complete and stable product design at the project's beginning.
- It allows large teams to work like small teams by dividing work into pieces, proceeding in parallel but synchronizing continuously, stabilizing in increments, and continuously finding and fixing problems.

It facilitates competition based on customer feedback, product features, and short development times by providing a mechanism to incorporate customer inputs, set

priorities, complete the most important parts first, and change or cut less important features."

Scrum follows common ISV rules :

- Always have a product you can theoretically ship
- Speak a common language on a single development site
- Continuously test the product as you build it

The key principles the Scrum development process are:

- Small working teams that maximize communication, minimize overhead, and maximize sharing of tacit, informal knowledge
- Adaptability to technical or marketplace (user/customer) changes to ensure the best possible product is produced
- Frequent "builds", or construction of executables, that can be inspected, adjusted, tested, documented, and built on
- Partitioning of work and team assignments into clean, low coupling partitions, or packets
- Constant testing and documentation of a product-as it is built
- Ability to declare a product "done" whenever required (because the competition just shipped, because the company needs the cash, because the user/customer needs the functions, because that was when it was promised...).

Scrum and empirical development are recommended as enabling processes for developing new software or software that already is object-oriented or has "clean interfaces". Work objects or subsystems with high cohesion and low coupling are grouped into packets. Teams are assigned one or more packets, maximizing a team's ability to work independently. Scrum is applicable for any size project.

Applicable to Small Simple and Large Complex Software Systems

Prior to Scrum, large complex systems had to be built using either a waterfall or modified waterfall processes. A defined, stepwise approach provided task-level controls for managing the process. However, many of these projects failed because they were unresponsive to changing technology and user requirements, and the adequacy of the deliverables from the tasks couldn't be determined.

A Scrum software project is controlled by establishing, maintaining, and monitoring key measurements as controls. These controls replace task and time-reporting controls used in the waterfall and other defined development processes. These controls are critical when a

software development project is viewed as an empirical process that encompasses an unknown quantity of instability and unpredictability. Use of these controls is the backbone of the Scrum development process.

Through the formalized controls of Scrum, the empirical development process becomes formally scaleable. Using an initial systems architecture and design, work can be partitioned and assigned to as many teams as required and managed at the team and rollup levels. Overall and team risk, workload, problems and other measurements are always known, assessed, and managed.

During the design and system architecture phase of Scrum, the designers and architects divide the project into packets. Packets are assigned to teams based on priority and scheduling constraints. Based on the degree of coupling between packets, groups of up to six teams are organized into a management control cluster. This continues upwards until a top level cluster has been created.

Empirical software development has been used to build systems as large as Windows NT (approximately 4 million lines of C and C++ code)and the Norfolk Southern Railway freight tracking system. Empirical software development has also been used to rapidly get sophisticated, functionally-rich products to market, such as Borland's Quattro Pro for Windows and Advanced Development Method's process management software, MATE (approximately 4,100 function points). Scrum has also been used for short, simple development projects.

Scrum Productivity

Compared to other development processes, Scrum delivers. Scrum was compared to other popular development processes by Capers Jones from Software Productivity Group: "Scrum methodology - similar to the iterative methodology, but assumes that all requirements are not known in advance, and that the fastest path to surfacing and implementing all requirements will be discovered empirically during the development process. Careful control mechanisms are used to assure on-time delivery of a high quality product, while allowing maximum flexibility of small, tightly coupled, development teams. Requires a well motivated team and good leadership to implement effectively. Productivity gains of 600% have been seen repeatedly in well executed projects. "

The Inner Workings of Scrum

Scrum consists of development processes and measurements that are used to control the development processes.

The key to the success of Scrum is using measurements to maximize flexibility and risk while maintaining control. Most projects try to avoid risk. Yet, risk is an inherent part of software development. Scrum embraces risk by identifying and managing risk-so that the best possible product can be built.

A Scrum software project is controlled by establishing, maintaining, and monitoring key control parameters. These controls are critical when a software development encompasses an unknown quantity of uncertainty, unpredictable behavior, and chaos. Use of these controls is the backbone of the Scrum development process.

The variables in the systems development project are risk, functionality, cost, time, and quality. These variables can be roughly estimated at the start of a project. Each variable will start changing from the moment the project starts. Variables are traded off against each other as the project progresses (improved functionality for later time and more money, etc.).

The controls used in Scrum are:

- **Backlog** - an identification of all requirements that should be fulfilled in the completed product
- **Objects/Components** - self-contained reusable things
- **Packets** - a group of objects within which a backlog item will be implemented. Coupling between the objects in a packet is high. Coupling between packets is low
- **Problems** - what must be solved by a team member to implement a backlog item within an object(s) (includes bugs)
- **Issues** - Concerns that must be resolved prior to a backlog item being assigned to a packet or a problem being solved by a change to a packet
- **Solutions** - the resolution of an issue or problem
- **Changes** - the activities that are performed to resolve a problem
- **Risks** - the risk associated with a problem, issue, or backlog item

These controls are measured, correlated, and tracked. The main controls are *backlog* and *risk*. *Backlog* should start relatively high, get higher during planning, and then be whittled away as the project proceeds - either by being solved or removed, until the software is completed. *Risk* will rise with the identification of *backlog* , *issues*, and *problems*, and fall to acceptable levels when the software is complete and delivered.

The Scrum methodology consists of three distinct processes.

Planning and System Architecture

Product functionality and estimated delivery requirement are planned based on current *backlog* and assessed *risk.* The result of this process is the most optimal, elegant design that meets product performance and architectural requirements.

The definition of a new release is based on currently known and uncovered *backlog*, along with an estimate of its schedule and cost. If a new system is being developed, conceptualization and analysis are performed. If this project is the next release of an existing system, the analysis is more limited.

A baseline product is established. Changes in the user, customer, competitive, and technological environment will require changes to this baseline definition as the project proceeds. Increased productivity through good tools or uncovered components may open the opportunity for adding more backlog to the product, or for releasing the product earlier.

Backlog and *risk* management will allow the product release to be planned and managed to optimize the product content and its chance of success - given the environment and resources available. In other words, the very best product possible will be built. Development will occur in a controlled environment, as close to the edge of chaos as possible.

Key is pinning down the date at which the application should be released, prioritizing functionality requirements, identifying resources available for the development effort, envisioning the application architecture, and establishing the target operating environment(s). Compared with other methodologies, the planning phase is conducted relatively quickly because it assumes that pragmatic managers and the course of events will require that these initial parameters will be later changed.

What differentiates the Scrum Planning and System Architecture process from other methodologies is:

- Controls are established : backlog (requirements) for this project is established and prioritized, risks are defined, objects for implementing backlog are identified, and problems are stated for implementing the backlog into the related objects.
- Team assignments : backlog is assigned to teams of no more than 6 developers, maximizing communication bandwidth and productivity.
- Prioritization : backlog items are prioritized for teams to work on, starting with infrastructure, then most important functionality to least important functionality.

Sprint (consisting of multiple sprints)

Visualize a large pressure cooker. Scrum development work is done in it. Gauges sticking out of the pressure cooker provide detailed information on the inner workings, including backlog, risks, problems, changes, and issues. The pressure cooker is where Scrum sprints occur, iteratively producing incrementally more functional product.

A sprint is a set of development activities conducted over a pre-defined period resulting in a demonstrable executable. The interval is based on product complexity, risk assessment, and degree of oversight desired. Sprint speed and intensity are driven by the selected duration of the sprint. The duration also depends on where it occurs in the

sequence of sprints (initial sprints are usually given longer duration, later sprints less duration), the degree of control desired, and the level of domain expertise of the various teams. Duration range from one to six weeks

This is where Scrum radically differs from traditional enterprise application methodologies because the planned product (*backlog, risk*) can be changed at the end of any sprint, in response to the environment (competition, new technology, development tool flaws, etc.). This ensures that the delivered product is a usable product.

Also different, the Sprint phase duration is unknown. The Sprint duration was initially planned-however, as the sprint iterations occur and the product is incrementally developed, the product may be deemed worth delivery at the end of any sprint ... because of market announcements, competitive pressures, or the product is just ready.

The project manager establishes sprint teams consisting of between 1 and 7 members (a fully staffed team should include a developer, quality assurance person, and documentation member). Each sprint consists of one or more teams working on their assigned *packets* to solve the *problems* posed for implementing *backlog* in the objects. Each team is given its assignment(s) and all teams are told to sprint to achieve their objectives on the same day between 1 and 6 weeks from the start of the sprint. However, this process is not as undisciplined as it may seem each team must deliver executable code to successfully end a sprint.

The project remains open to environmental complexity, including competitive, time, quality, and financial pressures, throughout the sprints. The definition of the planned product can be changed during any review meeting of a Sprint. The project and product remain flexible; the delivered product is the best possible and most relevant software possible.

Closure and Consolidation.

When the management team determines that the product is ready for delivery-based on the competition, requirements, cost, and quality-they end the Sprint phase and declare the release "closed". Closure is performed when a build is considered to have reduced risk adequately and resolved and implemented required backlog.

Closure consists of finishing system and regression testing, developing training materials, and completing final documentation. Developed product are prepared for general release. Integration, system test, user documentation, training material preparation, and marketing material preparation are among closure tasks.

The Scrum process asserts that a product is never complete; after the initial construction, it is constantly under development (otherwise known as maintenance and enhancements). Consolidation prepares for the next development cycle. The purpose of consolidation is to clean up the products and artifacts of this development cycle for a clean start on the

next development cycle. This provides an opportunity to clean up all of the loose ends that were let slip during the pressure of getting the release out the door.

Summary

Scrum produces breakthrough productivity, enabling building the best systems possible in complex, unpredictable environments.

We believe that we have captured the best practices at ISV's in Scrum, making them available to IS organizations. Scrum controls and flexibility puts the teams back in charge.**References**

The following sources contain reference material for the subjects of Scrum, "good enough software", and empirical (theoretical) processes.

WorldWideWeb

ADM's home page : http://www.tiac.net/users/virman/

Jeff Sutherland's page on Scrum : http://www.tiac.net/users/jsuth/scrum/index.html

James Bach's views : http://www.stlabs.com/real_01.htm

Books

Cusamo, M. and Selby, R. Microsoft Secrets, The Free Press, 1995

DeGrace, P. and Hulet-Stahl, L. *Wicked Problems, Righteous Solutions*. Yourdon Press, 1990

Gleick, J. Chaos, Making a New Science Penguin Books, 1987

Ogunnaike, B. *Process Dynamics, Modeling, and Control*. Oxford University Press, 1994

Takeuchi, Hirotaka and Nonaka, Ikujiro, The Knowledge-Creating Company : How Japanese Companies Create the Dynamics of Innovation. Oxford University Press, 1995

Articles

Bach, James. The Challenge of "Good Enough" Software, *American Programmer* October, 1995

Curtis, B. A Mature View of the CMM. In *American Programmer* ,September, 1994

Racoon, L.B.S. The Chaos Model and the Chaos Life Cycle. *In Software Engineering Notes*, vol. 20 no. 1, January 1995

Rumbaugh, J. What Is A Method? In *Journal of Object Oriented Programming* , October, 1995