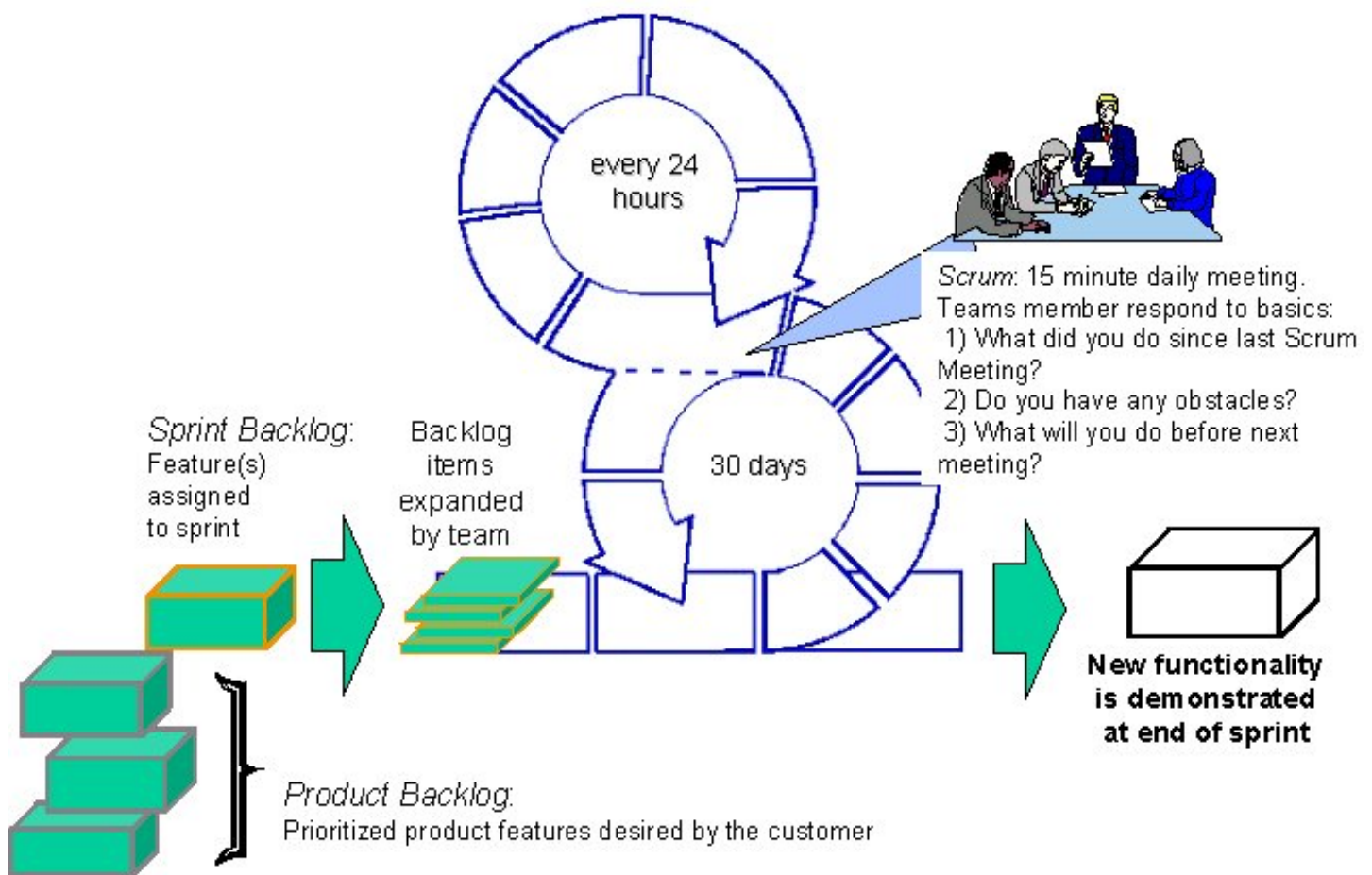


What is Scrum ?

Scrum is an iterative, incremental process for developing any product or managing any work. It produces a potentially shippable set of functionality at the end of every iteration. It's attributes are:

- Scrum is an agile process to manage and control development work.
- Scrum is a wrapper for existing engineering practices.
- Scrum is a team-based approach to iteratively, incrementally develop systems and products when requirements are rapidly changing
- Scrum is a process that controls the chaos of conflicting interests and needs.
- Scrum is a way to improve communications and maximize co-operation.
- Scrum is a way to detect and cause the removal of anything that gets in the way of developing and delivering products.
- Scrum is a way to maximize productivity.
- Scrum is scalable from single projects to entire organizations. Scrum has controlled and organized development and implementation for multiple interrelated products and projects with over a thousand developers and implementers.
- Scrum is a way for everyone to feel good about their job, their contributions, and that they have done the very best they possibly could.



Scrum naturally focuses an entire organization on building successful products. Without major changes - often within thirty days - teams are building useful, demonstrable product functionality. Scrum can be implemented at the beginning of a project or in the middle of a project or product development effort that is in

trouble.

Scrum is a set of interrelated practices and rules that optimize the development environment, reduce organizational overhead, and closely synchronize market requirements with iterative prototypes. Based in modern process control theory, Scrum causes the best possible software to be constructed given the available resources, acceptable quality, and required release dates. Useful product functionality is delivered every thirty days as requirements, architecture, and design emerge, even when using unstable technologies.

Over fifty organizations have successfully used Scrum in thousands of projects to manage and control work, always with significant productivity improvements. Scrum wraps an organization's existing engineering practices; they are improved as necessary while product increments are delivered to the user or marketplace. As heard about Scrum, "oh, that's just my idea X by another name". Except, Scrum is spelled out as values, practices, and rules in a development framework that can be quickly implemented and repeated.

Scrum has been used to produce financial products, Internet products, and medical products by ADM. In every instance, the organization was logjammed, unable to produce shippable products for such a long period of time that engineers, management, and investors were deeply concerned. Scrum broke the logjam and began incremental product delivery, often with the first shippable product occurring within the same quarter.

Scrum as applied to product development was first referred to in "The New New Product Development Game" (Harvard Business Review 86116:137-146, 1986) and later elaborated in "The Knowledge Creating Company" both by Ikujiro Nonaka and Hirotaka Takeuchi (Oxford University Press, 1995). [Based on their ideas and research in process theory](#) performed in collaboration with DuPont Advanced Research Facility, [Scrum was first formulated and presented to the Object Management Group in 1995](#). The Scrum process is fully described in a recent book from Ken Schwaber and Mike Beedle, *Agile Software Development with Scrum* (Prentice Hall, 2001). [An excerpt of that book can be read here](#).



valuspecs
click here!

- [Home](#)
- [News](#)
- [Match Reports](#)
- [Columns](#)
- [Features](#)
- [Rugby Photos](#)
- [Fun & Games](#)
- [Scrum Fitness](#)
- [Women's Rugby](#)
- [Rugby Guide](#)
- [TV Schedule](#)
- [Links](#)
- [Rugby Ringtones](#)

[Rugby Auction](#)



Rugby Primer

Scrums

Very often a player will lose the ball forward during a tackle or just while running and receiving a pass, thus knocking-on. If the ball is quickly picked up by the other team, the referee will let play continue to allow the recovering team to take advantage of the mistake. If no advantage occurs, then the referee will whistle for a scrum to be set at a spot he indicates on the pitch also called a mark. The team that did not lose the ball is awarded the ball to put into the scrum. A scrum is also awarded whenever a pass is made in which the ball goes forward.

The typical procedure of scrummaging involves each set of front row players binding and the hookers calling for the locks to join the formation. The flankers join on each side of the locks setting their shoulders below a prop's outside buttock. The No. 8 joins at the back between the hips of the two locks. While this is occurring the captain of the forwards can be calling a move while the backs are shouting out code words signalling what move they will be running. The forward pack with the put in is then allowed the courtesy of initiating the coming together of the scrum. Upon a prearranged signal between the hooker and scrumhalf, the scrumhalf will roll the ball into the tunnel underneath the two locked together front rows. Each of the hookers will then attempt to push the ball behind him with a sweep of his foot. All of this is occurring while each pack is attempting to push the other backwards driving themselves over the ball.

If the ball is won cleanly, most often the scrumhalf will run to the back of the scrum to retrieve the ball from in front of the No. 8's feet and pass it to the backs, to a breaking loose forward, or make a run or kick of his own. The opposing scrumhalf will follow looking for a chance to snap up any loose ball. The No. 8 may also decide to pick up the ball himself, and start a back row move from the back or base of the scrum.

London: 12.21 hrs Friday 11th Jun

Rugby Guide Menu...

Rugby Dictionary:



Rugby Primer:

- [The General Principles](#)
- [The Playing Field](#)
- [Fifteens Players](#)
- [Game Start](#)
- [Rucks & Mauls \(Loose Plays\)](#)
- [Penalties](#)
- [More on Running & Tries](#)
- [Tactical Kicking](#)
- [Scrums](#)

Following is a simple representation of how the players will line up at the start of a scrum awarded on the left side of the pitch:



One exciting aspect of scrummaging is the pushover try. A pushover try is scored when a scrum is set close to the attacking tryline. The attacking scrum will keep the ball at the No. 8's feet driving the defending pack backwards across the tryline. Once the ball has been dragged across the tryline, the No. 8 or scrumhalf will touch the ball down for the try.

The [scrum.com rugby dictionary](#) is available as a glossary of rugby terms. For a shortcut to any of the more detailed descriptions of various aspects of play, please use the buttons on the right hand side of the page to navigate the primer.

- [Lineouts](#)
- [Completion of Play](#)
- [Sevens & Tens](#)



📱 Rugby Betting...

Click below for the latest from our odds comparison service:

[<< More Betting & Games](#)

Get your rugby stuff here...



📱 News Search...

By Date:

Or - By Keyword(s):

[World Cup](#)

[Six Nations](#)

[Super 12](#)

[European Cup](#)

[Zurich](#)

Controlled-Chaos Software Development

This article presents a macro-process for developing object-oriented or clean-interface systems. The macro-process, Scrum, is a formalization of development processes used by many Independent Software Vendor's (ISV's).

Overview

Scrum arose from shared concerns between two ISV's, Advanced Development Methods (ADM) and VMARK Software (VMARK)ii. 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 companiesiii.

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.

Development in the Real World

With the availability of open systems tools, three-tier architectures, components, relational/object data bases, Internet, intranet, Notes, and event-driven systems, the complexity of systems development has increased exponentially. For instance, compare the testing of a user-responsive GUI to a controlled batch system.

Scrum's Characteristics

Projects are controlled through ongoing measurement and control of backlog, issues, risk, problems and changes -- task level management is not used

A deliverable product is always ready through the use of constant builds and testing in parallel with development

Small teams develop and enhance OO, component-based systems with clean interfaces

Chaos is found in environments with a high degree of unpredictability and complexity. Chaos is present in most systems development projects. Evolution favors those that are able to operate with maximum exposure to environmental chaos. Evolution deselects those who have insulated themselves from environmental change and have minimized chaos and complexity in their environment. The amount of chaos that a project can embrace and still succeed is maximized within the Scrum process.

Scrum formalizes "empirical", chaos-tolerant software development practices. Originating from industrial process control and biochemical process research, Scrum offers a productive alternative to the micro-management of traditional development processes, and the progressively imposing structure of the Software Engineering Institute's Capability Maturity Model.

Scrum allows a project team to determine when the system is "good enough" for its application. Unnecessary effort to create a system more robust than its environment demands doesn't have to be expended.

The payback from the Scrum development approach is breakthrough productivity, on the order of 600%iv, full utilization of the most current development tools, and the best possible software for the user's requirements.

Background

ADM is a research and development organization started in 1985. We have directed our efforts toward 1.) how to manage and control software development projects, and, 2) how to make a developer's job easier.

In support of these efforts, ADM has 1.) developed process repositories that allow organizations to acquire, correlate, and disseminate knowledge to project managers and developers, and, 2) provided project managers and developers with workbenches for easily using the knowledge to plan, manage, and perform work.

ADM's efforts have led to a number of firsts :

1987 -- CASE-oriented development methodology

1988 -- dictionary-based process management tool for automating methodologies

1991 -- hypertext process management tool for automating methodologies

1993 -- workgroup enabled OO process management tools

We have partnered with banks, pharmaceutical houses, computer companies, consulting organizations, and manufacturers in our efforts to automate and improve their use of system development processes. The resulting process automation tools and marketplace are a significant step up from the old paper-based methodologies.

In 1994 we reviewed of our approach. Our customers had been bedeviled by :

Currently available development processes often constrain and stifle development

The environment in which systems are built is undergoing radical transformations

Current process automation adds administrative work for managers and developers

Development processes are marginally used and soon become shelfware or diskware

Centralized, automated knowledge is best provided in books

Considering our observations, we established the following areas for research and development during 1995:

What is the best way to author and access knowledge as it moves from a tacit, unexpressed form to an explicit, published formv.

What sort of development process is appropriate for building systems in complex, unpredictable, changing environments (technical and business).

This article provides our answer to the second question. We reviewed recent advances in development processes, we studied the development processes in use at the most successful ISV's, and we turned to science to arrive at this answer.

Defined vs. 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 to ensure the desired result.

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.

Scientific Research

A formal body of knowledge regarding industrial and biochemical processes and their automation already exists.vi We asked scientists familiar with this knowledge why the systems development process was so problematicvii.

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 defined processes and empirical processes.

The scientists stated, "We are most amazed that your industry treats 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.

The scientists recommended -- since our business is an empirical process -- that we use measurements and controls, as done elsewhere in the physical world. The scientists provided the concepts of defined and empirical processes .. and they told us that we needed controls to manage the empirical systems development process.

Industry Research

We studied the development process at the most chaotic, pressure-ridden development environments known -- those at successful ISV's. Michael Cusamo summarizes this process in a recent book about Microsoftviii:

"It breaks down large products into manageable chunks -- a few product features that small teams can create in a few months.

It enables projects 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.ix"

From our experience, we added :

Our customers and users conduct business in an ever changing, competitive environment. They never can give us a "final spec" because their needs are constantly evolving. The best we can do is evolve a product as their needs evolve.

There is no end to productivity saving techniques and tools offered to us by software vendors and methodologists. It would be a shame not to use those that seem worthwhile.

Components are becoming mainstream. When available, they let us offer our customers and users more than planned.

Process Requirements

From the research, we devised the following requirements for a systems development process. It must be :

Chaos-tolerant, expecting the unpredictable and allowing flexible responses.

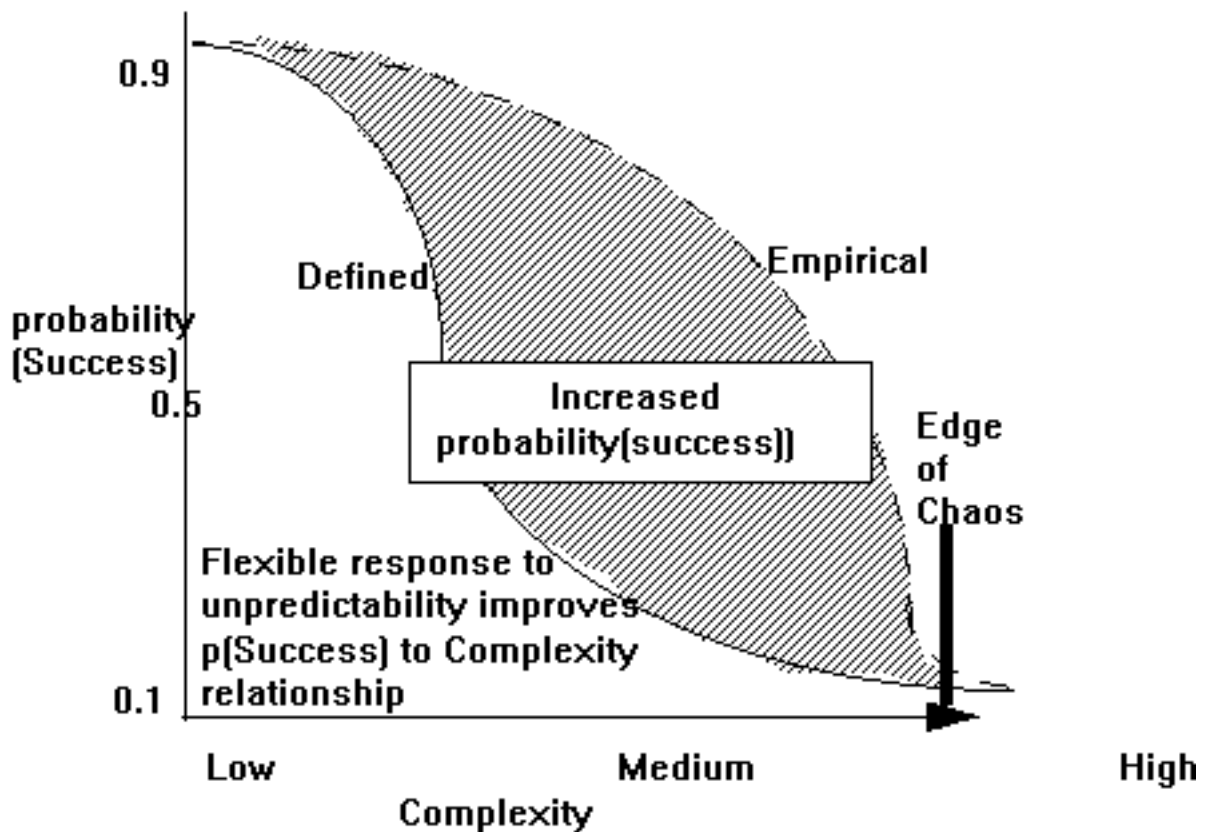
Constantly measured, with intelligent, timely control.

Assume that the product is never complete; working solutions must be provided as needed, but the product will continue to evolve with the customer and user

Maximize communications bandwidth and information sharing.

Provide the best possible software given the required functionality, quality, timetable, and resources.

We graphed our expectations for this new development process. Primarily, it should meet the above requirements through chaos-tolerance. The y axis indicates the probability of the best possible software being delivered when the customer needs it.



The x axis is the degree of complexity and unpredictability in which the system is being developed. The lower curve defines the performance of defined processes used for developing systems in increasingly chaotic environments. The upper curve defines the performance of empirical processes with controls used for developing systems in increasingly chaotic environments. The cross-hatched area between these two curves is the advantage provided by the use of empirical, controlled processes over defined processes.

The primary difference between the defined lower curve (waterfall, spiral and iterative) and empirical upper curve is that the empirical approach assumes that the development process is unpredictable and chaotic. Controls are used to manage the unpredictability and control the risk. Flexibility, responsiveness, and reliability are the results.

Running a project with good controls is like driving a sport car through a sharp curve. The constant, immediate feedback lets you control the car and maximize the speed while controlling the risk. The small teams used in a Scrum project are like top quality components in a sports car. When you respond to a measurement, the effect is immediate and appropriate.

Scrum Empirical Development Process

Scrum is a macro-process that defines and implements controls. Scrum consists of tasks that establish, monitor and manage backlog, work, risk, issues, problems, and changes. Micro-processes, such as object-domain analysis, are used for actual product construction.

Aberdeen Conclusions

New database, hardware, application, and networking technologies are not proprietary -- your competitors can use them as effectively as you. ISVs understand this, and bet on high-speed, quick-U-turn development processes to succeed in cutthroat markets. IS organizations can take the same tack: learn from ISVs and emphasize speed-to-deliver and flexibility rather than costs and backward compatibility. Methods like ADM's SCRUM give IS a way to drive ISV best practices deep into the development organization.

At the same time, IS should not have to be on the bleeding edge of yet another new software development methodology. Enterprise IS is not being a pioneer by following the SCRUM methodology $\frac{3}{4}$ ISVs are already using it. ADM is simply making it available in the form of the Product Management Facility.

Most importantly, IS should consider where this is leading in the long term. The new mission-critical applications can give the enterprise a competitive advantage -- but can the IS organization keep delivering after the first home run? ISV's that stay around for more than 10 years don't rest on their laurels: they use rapid development and flexible processes to gain more and more competitive advantage. Methodologies like SCRUM aren't just a passing fad or one-shot fix. If IS wants to succeed in the long term like the best ISVs, it should take a good hard look at ideas like SCRUMx.

Scrum formalizes the empirical "do what it takes" software development process used today by many successful ISV's. An 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.

Scrum combines our research and these ISV development processes into a formal process. See the sidebar for an industry analyst's conclusions regarding Scrum.

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

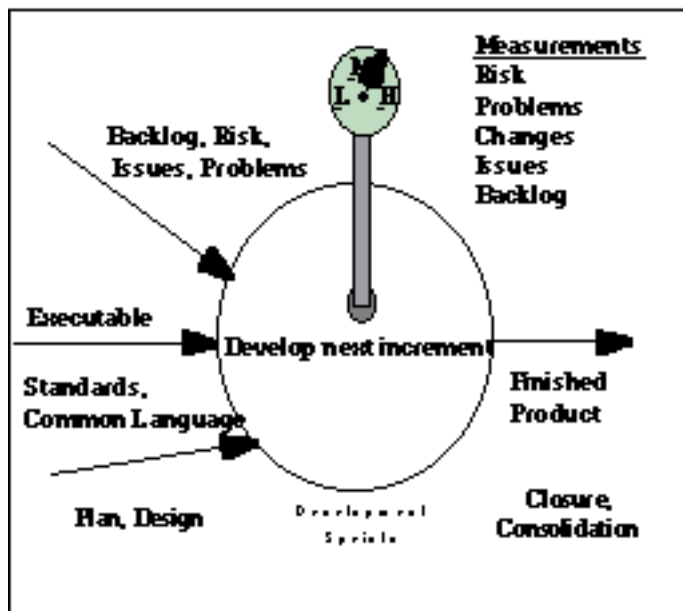
Scrum accepts that the development process is unpredictable. The delivered product is the best possible software, factoring in cost, functionality, timing, and quality. This concept has been discussed by James Bach of Software Testing Laboratories in various articles, including "The Challenge of Good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

Scrum Overview

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.

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 as the software is changed.

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 Sprint cycles occur, iteratively producing incrementally more functional product.



The Planning and System Architecture phases prepare the input that is placed in the pressure cooker. The input consists of ingredients (backlog, objects, packets, problems,

issues), recipe (system architecture, design, and prototypes), and cooking sequence (infrastructure, top priority functions, next priority...).

The Closure phase removes the final product (executable and documentation) and prepares it for shipment. The Consolidation Phase cleans up the pressure cooker and ingredients for the next batch.

The Inner Workings of Scrum

Scrum consists of development processes and measurements that are used to control the development processes. "The Scrum methodology consists of three distinct processes.

The initial process is Planning and System Architecture. Key in this phase is pinning down the date at which the application should be placed in production or released to the market, prioritizing functionality requirements (ranging from good-enough to best-possible), identifying the resources available for the development effort, envisioning the application architecture, and establishing the target operating environment(s).

However, compared to other methodologies, this planning phase is conducted relatively quickly because it assumes that pragmatic managers and the course of events will require that any or all of these initial parameters will be changed during the Sprint phase.

The next process, consisting of multiple Sprints, is where Scrum radically differs from traditional enterprise application methodologies. 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 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. Each team can select its own object-oriented tools and its own means to achieve its objectives -- these are not selected from on high and then forced on the developers and other team members. However, this process is not as undisciplined as it may seem -- each team must deliver executable code to successfully accomplish the Sprint.

At the end of the Sprint period -- three weeks, for example -- all the teams meet to review their progress (including executable code delivered to date) with each other, the project manager, customers/prospects, and the enterprise's senior executives. At the conclusion of the review session, the project manager and his or her superiors have the opportunity to change anything and everything. This built in flexibility allows the organization to add to

(or, rarely, subtract from) the requirements for the application during the Sprint phase.

When the objectives of the application development effort are completed in terms of functionality and quality, or the time/budget constraints are reached, the Sprint phase is completed. Approved modifications to the original planning and systems architecture are lumped into a category called backlog and assigned to the teams (whose resources may also be changed to reflect the new objectives) at the beginning of the next Sprint period.

Finally, Scrum ends with the Closure phase. Closure consists of finishing system and regression testing, developing training materials, and completing final documentation. Approved modifications to the original planning and systems architecture are lumped into a category called backlog and assigned to the teams (whose resources may also be changed to reflect the new objectives) at the beginning of the next Sprint period.

As opposed to traditional methods, which try to lock in requirements during the planning phase, Scrum (and Sprints in particular) provides the development team with a tremendous amount of post-planning flexibility. For example, many enterprises that had selected OS/2 as their chosen client operating environment in early 1995 decided by late 1995 to move to Windows NT Client. This change -- replace OS/2 with Windows NT -- would have been assigned to the appropriate teams to implement at the beginning of the next Sprint."xii

Summary

We have formalized empirical, chaos-tolerant development processes into a macro process named Scrum. The detailed micro processes of OO, BPR, and other techniques are implemented within this macro process as needed and as standardized within an organization.

Systems development is not now, and may never be, a cookbook process. Component-based development already eases our job, but the intelligent, adaptive, ongoing interaction of a project team and the environment is mandatory to successful product delivery. Controls ensure that that product is the best possible that could be produced by that team, given that technology, for that environment.

-
- i The word Scrum was first applied to the development process by Takeuchi and Nonaka in their seminal article, The New Product Development Game (Takeuchi, Hirotaka and Nonaka, Ikujiro). The New Product Development Game. Harvard Business Review.

Jan/Feb 1986). Scrum describes a product development process used by the most innovative American and Japanese companies (ibid, *The Knowledge-Creating Company : How Japanese Companies Create the Dynamics of Innovation*. Oxford University Press, 1995). Scrum was also described as a form of systems development process by Peter DeGrace (DeGrace, P. and Hulet-Stahl, L. *Wicked Problems, Righteous Solutions*. Yourdon Press, 1990).

- ii ADM produces process automation software. VMARK produces object-oriented software development environments
- iii ADM and VMARK are members of the Object Management Group. The original Scrum development process paper was presented at a meeting of OMG's Business Object Modeling Special Interest Group (OMG BOMSIG) at OOPSLA95
- iv From Business Object Architecture presentation given at Object World, 1995, by Jeff Sutherland, based on research done by Capers Jones
- v We found that the answer lies in learning theory, the size and organization of development teams, tactics for capturing tacit knowledge and moving it to explicit, published knowledge, the hierarchy of knowledge, and the world-wide-web; this question is addressed in another paper
- vi Ogunnaiké, B. *Process Dynamics, Modeling, and Control*. Oxford University Press, 1994
- vii Scientists at DuPont's Advanced Research Facility in Wilmington, Delaware collaborated during the spring and summer of 1995
- viii Cusamo, M. and Selby, R. *Microsoft Secrets*, The Free Press, 1995
- ix ibid
- x Aberdeen Group , *Upgrading To ISV Methodology For Enterprise Application Development Product Viewpoint Volume 8/Number 17*, 1995
- xi Bach, James. *The Challenge of "Good Enough" Software*, American Programmer October, 1995
- xii op.cit. Aberdeen Group

Controlled Chaos : Living on the Edge

Copyright 1996 Advanced Development Methods, Inc.

All Rights Reserved

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 scalable. 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