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
>
> Centralizated, 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 :

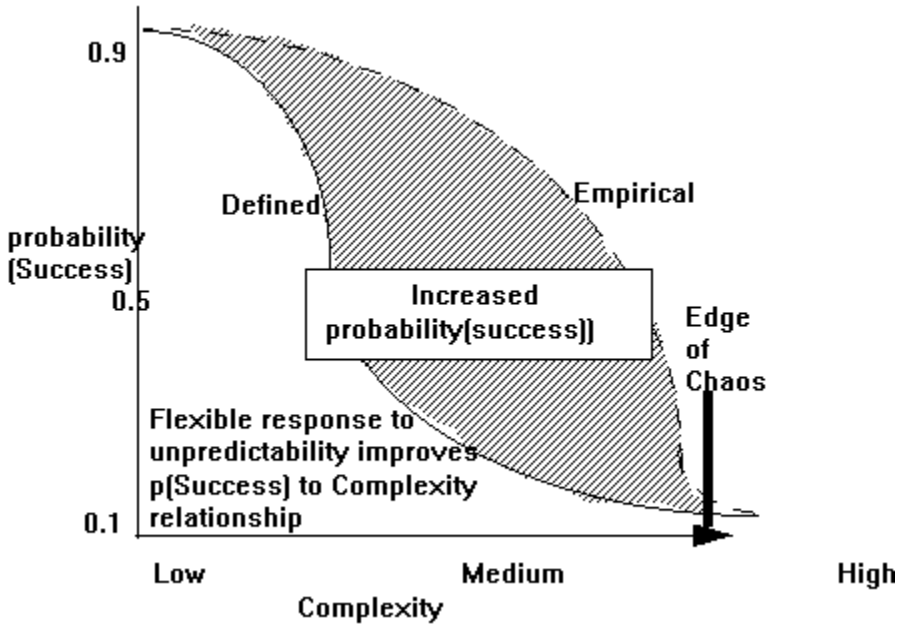
> Chaos-tolerant, expecting the unpredictable and allowing flexible responses.
>
> Constantly measured, with intelligent, timely control.

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 metholodogy. Enterprise IS is not being a pioneer by following the SCRUM methodology ¾ 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.

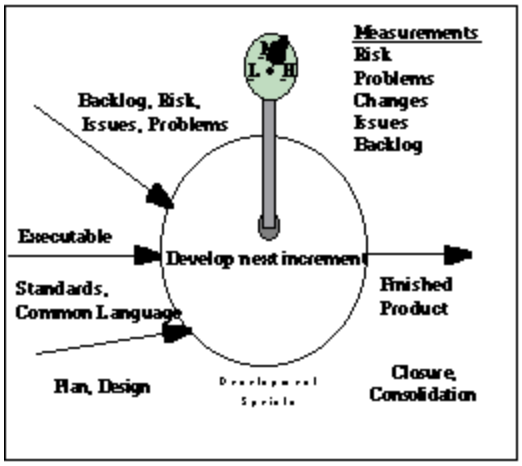
Scrum accepts that the development process is unpredictable. The delivered product is the best possible software, factoring in cost, functionality, timing, and quality. This concept has been discussed by James Bach of Software Testing Laboratories in various articles, including "The Challenge of Good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

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 Ogunnaike, 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