RADical Software Development

Dr. Sam Bayer - Director Market Development, Sapiens, Inc.
Jim Highsmith - Principal, Knowledge Structures, Inc.

Introduction[i]

The challenges facing software development organizations are well documented in articles and conference speeches. They can be summarized as more, better, and faster. In each of these presentations there is usually a “silver bullet” of sorts — object-oriented, the latest glittery development tool, or empowered teams. While each of these “bullets” has merit — be it technique, method, tool, organization -there remains a need for a transformational framework. This framework should both integrate diverse disciplines and, at the same time, provide incentive and clear returns for major change.

Rapid or accelerated development has garnered much interest in the last few years. RAD (Rapid Application Development), an outgrowth of prototyping methods and conceptual work by Barry Boehm (Spiral Model) and Tom Gilb (Evolutionary Life Cycle), has appeal in an environment where getting products out quickly has changed from a competitive advantage to a competitive necessity. But RAD can be the catalyst for more far-reaching change, a RADical change, transformation, or re-engineering of how organizations develop software. Michael Hammer, in a recent Computerworld interview, emphasizes the point: “Radical surgery is needed in IS processes. One of the first ideas that will have to go is the whole notion of traditional systems development life cycles.”[1]

This article has two objectives: first, to provide a framework for achieving successful, accelerated projects. Second, to show how RADical project success, viewed within the larger organizational context, can be a catalyst for a more general transformation of software development processes. We have described *RADical Software Development* as:

A *customer-driven* application development lifecycle that:

delivers *quality* solutions
is an *evolutionary* process
uses *continuous* application engineering techniques
is performed by a *dedicated* professional team
uses *timeboxed* project management
is *enabled* by powerful development tools
results in *profound* productivity benefits.

RADical software development focuses on three areas: customer, product, and process. Much of the current literature on rapid development focuses on particular techniques (JAD, prototyping, timebox). To raise the chances that these projects are successful, and certainly for these projects to have an impact on changing the larger organization, we need to expand their focus.

Close to the Customer

Stanley Marcus of Neiman Marcus put the issue of customer and product as succinctly as anyone has, “There are only two things of importance. One is the customer, and the other is the product. If you take care of customers, they come back. If you take care of product, it doesn’t come back. It’s just that simple. And it’s just that difficult.”

[I] This article published in the June 1994 issue of *The American Programmer* magazine.

There are a number of useful techniques to build a customer driven organization, but three seem to have particular application to a software development project:

Customer Satisfaction Surveys
Joint Application Development (JAD) Sessions
Customer Focus Groups

Customer satisfaction surveys can take a variety of forms, from a written questionnaire to individual interviews. Each has benefits and shortcomings, but the key issue is to measure. A series of questions using a 1-7 scale is appropriate for qualitative measures. More quantitative measures can be developed as areas are highlighted by customers. The key questions are: "Which product and service characteristics are important to your customers? What is the relative importance of each of their wants?

JAD sessions have been touted for years as an important technique for gathering specifications (or design or other information) for software applications. They engage customers and developers in a common dialog, and speed the development process. The key short-coming of the process, not really JAD's themselves, is untimely feedback. Customers attend JAD sessions, usually get documented results, usually feel pretty good about the sessions, but product results are still ephemeral. Our contention is that without an evolutionary development cycle (discussed later) that produces interim product versions, the benefits of JAD are quickly dissipated. JADs need to be quickly followed by product versions, and then the next customer-oriented technique -- the focus group meeting.

Like JAD sessions, customer focus groups are special meetings with several imperatives:

Each session is a review of the product itself, not a document.
The objective of each session is to find and record customer change requests.
The meeting has roles, such as a facilitator, like a JAD session
Developers must act as if they are behind one-way glass -- be seen, not heard
Everything is a success! Whether comments are positive or negative, something is learned about the product.

In a variety of projects, focus groups have proven to be remarkably successful. Customers like the fast turnaround: they evaluate a product rather than some indecipherable diagram, their comments are taken seriously -- and they invariably become what developers have wanted -- they become involved!

Results-Oriented Quality

Is *quality* important? According to Capers Jones, "A cumulative defect removal rate of 95% on a project appears to be a nodal point where several benefits accrue ... these projects have the shortest schedules, the lowest quantity of effort, and the highest level of user satisfaction."[2]

This leads to the question of what defines quality. Is it an intrinsic part of the product, or is it a some person's viewpoint? We use Jerry Weinberg's definition, "Quality is value to some person."[3] Software "value" appears to have three broad characteristics -- Scope, Schedule, and Resources. Value is also tied to use. For example, what value would you consider Windows 3.1 to have if it was controlling *your* heart-lung machine? Given that usage, Windows would have very little value. But if you want to run a suite of integrated desktop applications, Windows has significant value.

So to succeed in the software development business, we must build high quality software, and that quality must be defined (primarily) by the customer. And whether we like it or not, whether we think it is fair or not, the one item that dissatisfies most customers of software products is *speed of delivery*. Our customer's perception of quality is highly "speed" oriented. The development process must be reengineered to meet this expectation.

Software Process Reengineering

If we are to attain processes that produce software at higher speeds, we first need to closely examine the beliefs or attitudes about "control" . The goal of a development organization should be getting product out the door, i.e., delivery or throughput. Throughput is the rate at which the system generates money through sales.[4]

Our belief in control leads to barriers to throughput. Belief in control leads to hierarchical layers of management. In large, multiple project software development organizations, how much time loss is due to control points in development? Management reviews, data administration approvals, operations standards, supervisor reviews, architectural guidelines, design standards -- all developed with the best intentions, but as a whole, place barriers in the way of finishing projects. Most development organizations end up with the equivalent of high "work in process inventory" and very little throughput. Per Goldratt's definition above, organizations are not paid for inventory, but for sales, delivery to the customer. The "waterfall" type development methodologies are an outgrowth of the belief in control.

As we talk about RADical development -- powerful tools, teams, JADs, etc. -- it could be one reason it works is that it emphasizes throughput, getting something out the door, and de-emphasizes control, blocking the door. Giving a team a clear objective, concentrating the resources, and getting obstacles out of the way is often half the secret in making rapid development successful.

There are four key aspects to re-engineering the software development process.

1. The overall process, the software development life cycle, must undergo a transformation From a static, documentation orientation to a dynamic, evolutionary, product orientation.

2. Doing activities faster requires a skill at "timeboxed" project management techniques. An evolutionary life cycle will not yield the desired benefits without promoting highperformance, dedicated teams.

3. Transformation to a RADical approach does not mean abandoning the critical software andinformation engineering skills gained in the last 15 years. Instead, these skills need to be redirected into -- continuous application engineering.

4. Transformation to a RADical approach does not mean abandoning the critical software and information engineering skills gained in the last 15 years. Instead, these skills need to be redirected into – continuous application engineering.

Evolutionary Product Development

Traditionally, the model for software development has been the "waterfall model." Most of the commercially available methodology products follow this approach. While waterfall methods brought some stability and order to the chaos of earlier development efforts, their shortcomings are becoming increasingly apparent and many software development organizations are lost in the "swamp" of waterfall development.

Maintainability and life cycle costs have been a major reason for installing software engineering techniques and methodologies. One issue often raised with rapid development is that of maintainability due to poor design or architectural planning. If rapid development is implemented as a return to coding without thinking, then the issue is valid. Done properly, maintainability in an evolutionary environment is even **better**! Why? Because the essence of evolutionary development is change, incorporating a large volume of customer changes as the project progresses is require of the process. In order to do these effectively, the development approach focuses on making change easy, including good design, architecture, coding guidelines, etc. This tests the change process over and over again as the project progresses. Done properly, evolutionary development produces more maintainable products.

The evolutionary model produces appropriate documentation, but focuses on delivering versions of the product. The documentation produced is more appropriate to the phase at hand and it “evolves” just like the product. For example, the first release might include a “sketch” of a data model, and the final release might include a complete data model.

Evolutionary development speeds the process and improves feedback. It can:

Establish applications that can evolve over time
Accommodate changing business needs
Adapt development processes to the application
Deliver earlier benefits to customers
Reduce the risks of major failures
Help in gaining customer confidence early in the process.

The following figures show different perspectives on an evolutionary development process. Figure 3, illustrates that each application is both initiated as a project and planned in an evolutionary mode. Evolutionary planning would include developing high-level requirements, segmenting the application (if required), initiating an overall architectural framework (especially if several segments are identified), and planning the iterative version cycles.

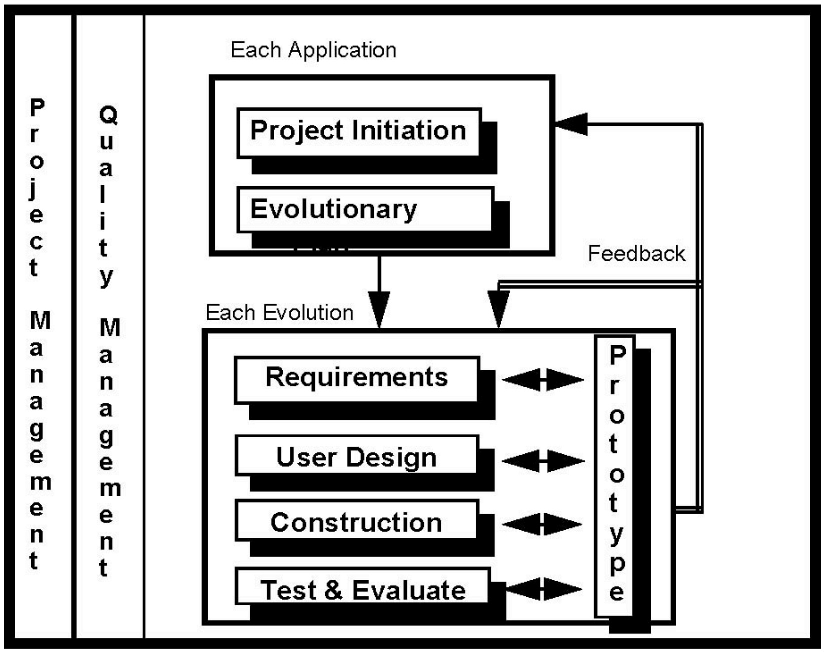

Figure 3 -- The Evolutionary Life Cycle

Figure 4 shows that evolutionary planning occurs prior to any development work. For small applications, this front end work may be very brief, essentially setting up the project. At the end of the project there is usually an installation phase (internal products) or a manufacturing/ distribution phase (external products). Between planning and release there may be several major versions (milestones) and a series of interim versions. With more than one application segment, there is usually a combined integration/ quality assurance/ beta test phase as part of manufacturing/installation.

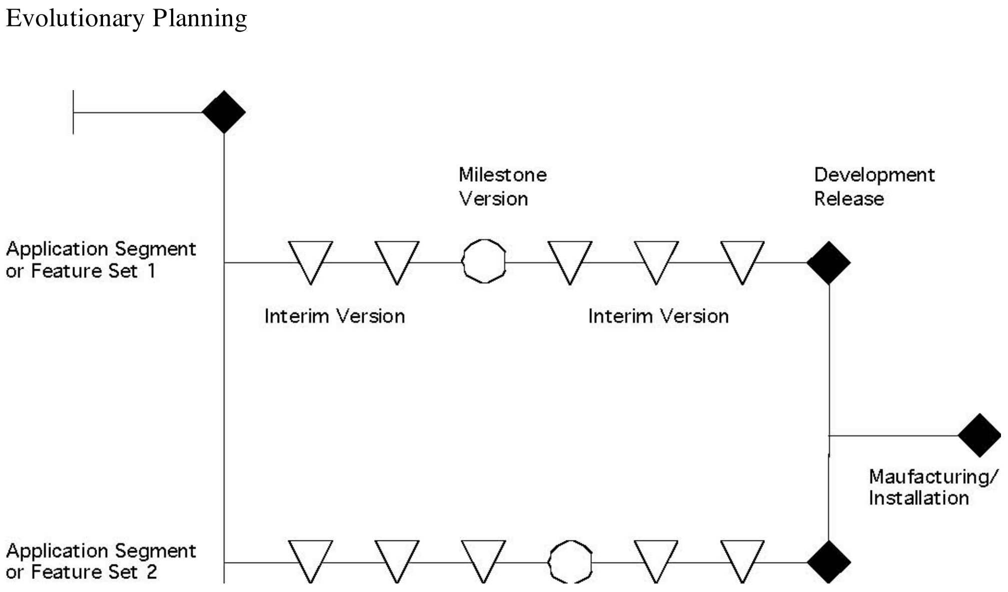
Evolutionary Planning
Milestone Version
Development Release
Application Segment or Feature Set 1
Interim Version
Interim Version
Maufacturing/ Installation
Application Segment or Feature Set 2

Figure 4 -- Evolutionary Versions

Figure 5 shows the converging nature of an evolutionary project. By constant attention to and involvement of the customer through JADs, customer focus groups, and ongoing project participation, the projects "converge" on success. Within a version, a tighter 1-day (continuous) to 3-week cycle would operate as shown by the interim version arrow in the figure. A major customer focus group session defines the end-point of a version.

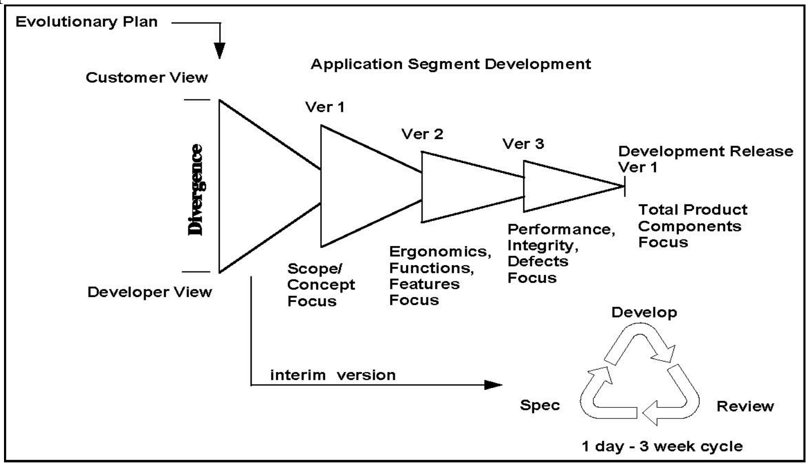
Evolutionary Plan
Customer View
Application Segment Development
Ver 1
Ver 2
Ver 3
Development Release Ver 1
Divergence
Total Product Components Focus
Performance, Integrity, Defects Focus
Ergonomics, Functions, Features Focus
Scope/ Concept Focus
Developer View
Develop
interim version
Spec
Review
1 day - 3 week cycle

Figure 5 -- Evolutionary Versions

Timeboxed Project Management

Good project management is even more important as the pace of projects accelerates. There is little time for the delays experienced on long projects. Quick decisions are needed. Barriers are not merely eliminated, they are anticipated. Meetings cannot be delayed until next week. The project focus -- goals, objectives, constraints -- must be clearly articulated and re-reinforced. Selecting the right projects for an evolutionary approach is critical to success. The executive sponsor, who is the prime customer decision-maker commissioning the project, needs to be a very active participant.

It is not the intent of this article to repeat the tenets of good project management. However, the following are of particular importance to accelerated projects:

> RADical development is not appropriate for every project. Projects should be carefully selected.
> Having an executive sponsor who is knowledgeable and committed to the process.
> Time is the project management driver. Scope or resource adjustments are traded-off rather than altering time goals.
> Management controls and guideposts will be different.
> The change control and configuration management processs are even more important.
> The team should be located together.
> Risks must be addressed! Contingency plans should be considered early.
> Accelerated schedules do not mean irrational schedules.

The final important point about RADical projects is to make noise. Keep the project's progress visible to the team, customers, and management. Since RADical development will be new for many organizations, make sure to market it well. Part of this "visibility" is to ensure improvements are measurable...things like elapsed time for projects, total resources expended, function points delivered, perceived quality by the customer, and change requests processed need to be monitored and reported on a regular basis.

High-Performance Teams

There is a tremendous emphasis in many organizations today on "teams." This is often stated as a goal of moving from a hierarchical, top-down organization to an empowered, team-oriented organization. This team orientation is one of the foundation blocks for many process improvement efforts. Software development organizations seem to be good candidates for this "team" orientation, but building teams, much less high performance teams, takes enormous skill and discipline, and, oh yes, a dusting of luck!

There is considerable difference between a real team and a group of people who work together. Given the proper environment, results can be startling:

> *A jelled team is a group of people so strongly knit that the whole is greater than the sum of its parts. ... Once a team begins to jell, the probability of success goes up dramatically. The team can become almost unstoppable, a juggernaut for success.*[5]

Teams can just as easily degenerate into being less than the sum of its parts. So what ingredients go into making successful teams? One of the best answers to this question comes from Katzenbach and Smith The Wisdom of Teams.[6] We use their definition of a "real" team as a framework for building good software development teams:

A team is a small number of people with complementary skills who are committed to a common purpose, performance goals, and approach for which they hold themselves mutually accountable.

"...committed to a common purpose, performance goals." Again and again, Kazenback and Smith reiterate that all other team-building efforts combined will not produce the desired results without the right purpose and goals to which the team is committed.

"...small number of people with complementary skills." Real teams are small; fewer than ten should be on the core team. The intense interaction necessary for a team to "jell" is nearly impossible with larger groups. In addition to size, every team needs a blend of skills --technical skills, customer area business skills, problem-solving and decision making skills, and interpersonal skills.

"...common approach." Teams can squander their advantage of a common purpose and goals if members constantly argue over approach. One faction believes in technical reviews; one does not. One faction believes strongly in an evolutionary approach; one holds firm to a traditional approach. In a jelled team, approach issues are discussed, decisions are made and members support the decisions.

"...for which they hold themselves mutually accountable." In a society that stresses individuality, and particularly organizational cultures that emphasize individual accountability, it is often very difficult for team members to subordinate their "performance" to that of the team.

Truly jelled teams develop their own sense of reward because they believe in the project's vision. They are willing to subordinate their personal agendas and rewards to those of the team, and thereby provide the foundation that allows the team to far outperform a group of individuals. It is not easy. Building the trust to achieve mutual accountability involves hard work, and the personal risk associated with resolving conflicts. Many groups avoid that risk and thereby deny themselves the chance to be a real team. Building high-performance teams begins with individuals willing to take this personal responsibility.[7]

Continuous Application Engineering

Some proponents of rapid development methods see them as a replacement for software and information engineering techniques. In addition, many writers of iterative and prototyping methods eschew formal techniques. But abandoning what we have learned about building high-quality systems is not the answer. Many of the frustrations with software engineering have come about from two sources.

First, the confusion between the techniques themselves (data flow modeling, entity relationship modeling, object modeling) and the "Methodologies" that have been developed to incorporate many of these techniques. No matter what kind of project is undertaken, some analysis work is required. We need to use whatever appropriate techniques we have in our tool bag to accomplish that analysis.

Whether you are a "software engineer" or an "information engineer" or an "object engineer," don't abandon the useful techniques developed over the years. As mentioned during the discussion of the evolutionary life cycle, the analysis activity lasts throughout the project, as do other major technical activities.

The more skilled the analyst, designer, data modeler, or programmer is at these techniques, the faster, and better the project will be. In fact, without these skills, RADical projects will not be any more successful than traditional ones.

Conclusions

At the beginning of this article, we stated its objectives as two fold: "First, to provide a framework for achieving successful accelerated projects. Second, to show how RADical project success, viewed within the larger organizational context can be the catalyst for transforming software development processes."

Hopefully, we have kindled interest in the transformation process by showing a glimpse of potential results, and a framework for achieving them. A transformation process begins with a single project having the goal of delivering visible and tangible benefits to the customer. That first project should focus on project related skills and delivering a useful software product to the customer. A second series of projects might then focus on delivering useful products and refining the evolutionary process. A third series of projects might then be undertaken with a further goal of implementing the organizational and management changes needed. This three-pronged attack should be viewed as a 12- to 18- month undertaking. Each project is used both as a vehicle for longer range transformation and to focus on various aspects of the RADical Process.

Reengineering processes involves major changes and major change is extremely difficult. But the driving global competitive forces to change are awesome --heightening customer dissatisfaction with information technologies' inability to keep pace. If information technology is one key to re-engineering corporations, it is time to think seriously about re-engineering the engineers.

1 Hammer, Michael, "One on One Interview,"*Computerworld*, January 24, 1994

2 Jones, Capers, Applied Software Measurement, 1992.

3 Weinberg, Jerry, *Quality Software Management*, Vol. 1(1992) & 2(1993).

4 Goldratt, Eliyau and Robert Fox, *The Race*, 1986.

5 DeMarco, Tom and Tim Lister, *Peopleware*, 1987.

6 Katzenbach, Jon and Douglas Smith, *The Wisdom of Teams*, 1993.

7 Jim Highsmith, "Software Ascents", *American Programmer*, June 1992, pp. 20-26.