Improving Software Development Economics Part III: Improving Team Proficiency and Improving Automation

by Walker Royce
Vice President and General Manager
Strategic Services
Rational Software

This is the third installment of a four-part series of articles that summarize our experience and discuss the key approaches that have enabled our customers to make substantial improvements in software development economics. In the first article, I introduced a framework for reasoning about economic impacts and introduced four approaches to improvement:

1. *Reduce the size or complexity of what needs to be developed.*

2. *Improve the development process.*

3. *Use more proficient teams.*

4. *Use integrated tools that exploit more automation.*

In last month's article, I discussed the first two approaches. This month, I will discuss some of the discriminating techniques involved with the last

two approaches: using more proficient teams and exploiting more automation.

Improving Team Proficiency

Getting more done with fewer people is the paramount underlying need for improving software economics. The demand for competent software professionals continues to outpace the supply of qualified individuals. In almost every successful software development project and software organization that Rational encounters, there is a strong commitment to configuring the smallest, most capable team. However, most troubled projects are staffed with more people than they require. "Obese" projects usually occur because the project culture is more focused on following a process rather than achieving results. In the previous article covering process improvement, and in the Rational Unified Process, there is a continuous, lifecycle emphasis on achieving results. This is a subtle but paramount differentiator between successful, results-driven, iterative development projects and unsuccessful process-driven projects.

So how can organizations use smaller, more capable teams? Rational has identified three different levels that need to be addressed: enhancing individual performance, improving project teamwork, and advancing organizational capability.

Enhancing Individual Performance

Organizations that analyze how to improve their employees' proficiency generally focus on only one dimension: training. Although training is an important mechanism for enhancing individual skills, team composition and experience are equally important dimensions that should be considered.

Balance and *coverage* are two important characteristics of excellent teams. Balance requires leaders and followers, visionaries and crank-turners, optimists and pessimists, conservatives and risk takers. Whenever a team is out of balance, it is vulnerable. Software development is a team sport. A team loaded with superstars, each striving to set individual records and be the team leader, can be embarrassed by a balanced team of solid players with a few leaders focused on the team result of winning the game. Managers must nurture a culture of teamwork and results rather than individual accomplishment. The other important characteristic, coverage, requires a complement of skill sets that span the breadth of the methods, tools, and technologies.

Two dimensions of experience necessary to achieve sustained process improvements are equally important: *software development process maturity* and *domain knowledge*. Unprecedented systems are much riskier endeavors than systems that have been built before. Experience in building similar systems is one of the paramount qualities needed by a team. This precedent experience is the foundation for differentiating the 20 percent of the stuff that is architecturally significant in a new system. A mature organization that builds real-time command and control systems will not be capable of exhibiting its usual mature performance if it takes on

a new application domain such as e-business Web site development.

Improving Project Teamwork

Although it is difficult to make sweeping generalizations about project organizations, some recurring patterns in successful projects suggest that a core organization should include four distinct subteams: management, architecture, development, and assessment. The project management team is an active participant, responsible for producing as well as managing. Project management is not a spectator sport. The architecture team is responsible for design artifacts and for the integration of components. The development team owns the component construction and maintenance activities. The assessment team is separate from development, to foster an independent quality perspective as well as to focus on testability and product evaluation activities concurrent with on-going development throughout the lifecycle. There is no separate quality team because quality is everyone's job, integrated into all activities and checkpoints. However, each team takes responsibility for a different quality perspective.

Some proven practices for building good software architectures are equally valid for building good software organizations. The organization of any project represents the architecture of the team and needs to evolve in synch with the project plans. Defining an explicit architecture team with ownership of architectural issues and integration concerns can provide simpler and less error-prone communications among project teams.

Figure 1 illustrates how project team staffing and the organizational center of gravity evolves over the lifecycle of a software development project.

- *Inception*: A management team focus on planning, with enough support from other teams to ensure that the plans represent a consensus of all perspectives.

- *Elaboration*: An architecture team focus, where the driving forces of the project reside in the software architecture team and are supported by the software development and software assessment teams as necessary to achieve a stable architecture baseline.

- *Construction*: A development team focus, where most of the activity resides in the software development and software assessment teams.

- *Transition*: A customer-focused organization, where usage feedback is driving the organization and activities.

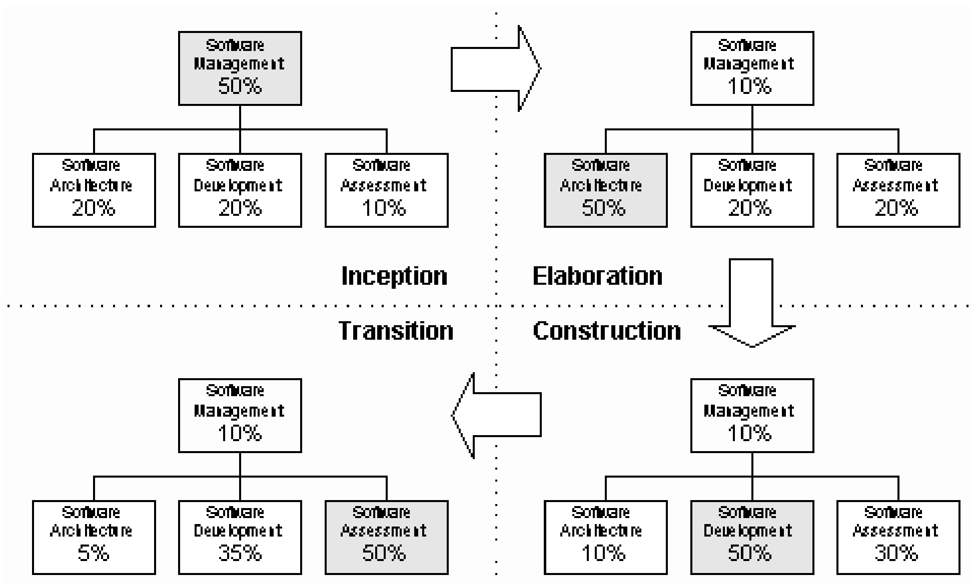

Figure 1: Team Evolution Over the Software Lifecycle

Teamwork is much more important than the sum of individual skills and efforts. Project managers need to configure balanced teams with a foundation of solid talent and put highly skilled people in the high-leverage positions. These are some project team management maxims:

- A well-managed project can succeed with nominal engineering talent.
- An expert team of engineers will almost never succeed if a project is mismanaged.
- A well-architected system can be built by a nominally talented team of software builders.
- A poorly-architected system will flounder even with an expert team of builders.

Advancing Organizational Capability

Organizational capability is best measured by trends in project performance rather than by key process area checklists, process audits, and so forth. Figure 2 provides some simple graphs of project performance over time to illustrate the expectation for four different levels of organizational capability.

1. *Random*: Immature organizations use *ad hoc* processes, methods,

and tools on each new project. This results in random performance that is frequently unacceptable. Probably 60 percent of the industry's software organizations still operate with random, unpredictable performance.

2. *Repeatable*: Organizations that are more mature use foundation capabilities roughly traceable to industry best practices. They can achieve repeatable performance with some relatively constant return on investments in processes, methods, training, and tools. In our experience, about 30 percent of the industry's software development organizations have achieved repeatable project performance.

3. *Improving*: The industry's better software organizations achieve common process frameworks, methods, training, and tools across an organization within a common line of business. Consistent, objective metrics can be used across projects, which can result in an improving return on investment from project to project. This is the underlying goal of ISO 9000 or SEI CMM process improvement initiatives, although most such initiatives tend to take process- and activity-focused perspectives rather than project-result-focused perspectives. At most, 10 percent of the industry's software development organizations operate today at this level of capability.

4. *Market leading*: Organizations achieve excellent capability, which should align with market leadership, when they have executed multiple projects under a common framework with successively better performance; have achieved an objective experience base from which they can optimize business performance across multiple performance dimensions (trading off quality, time to market, and costs); and practice quantitative process management.

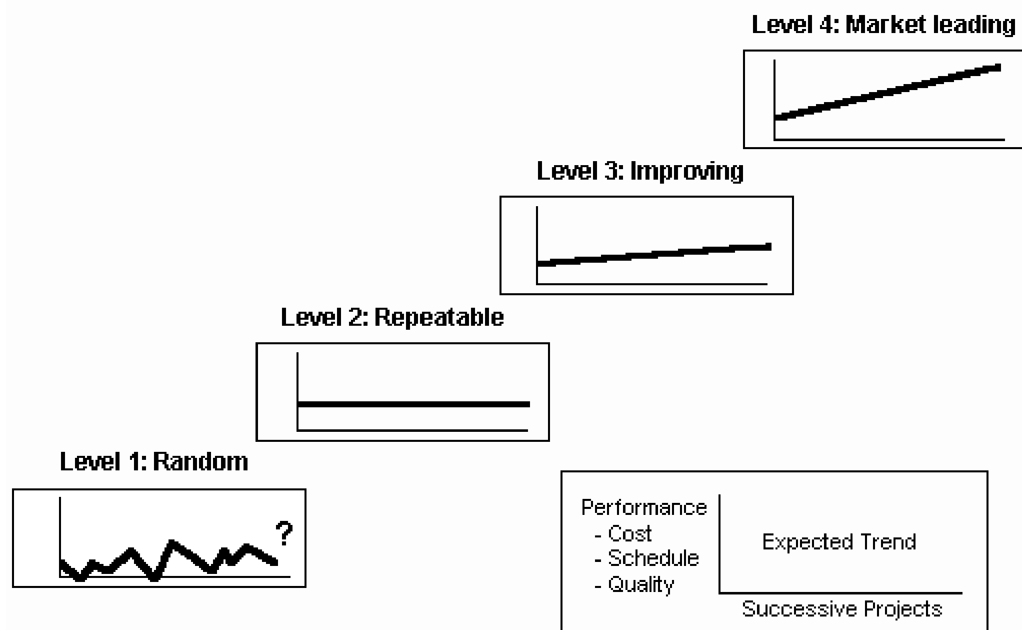

Figure 2: Organizational Capability Improvement Measured Through Successive Project Performance

In any engineering venture, where intellectual property is the real product, the dominant productivity factors will be personnel skills, teamwork, and motivations. To the extent possible, a modern process encapsulates the requirements for high-leverage people in the early phases, when the team is relatively small. The later production phases, when teams are typically much larger, should then operate with far less dependency on scarce expertise.

Improving Automation Through Integrated Tools

In last month's article, I described process improvements associated with transitioning to iterative development. These improvements are focused on eliminating steps and minimizing the scrap and rework inherent in the conventional process. Another form of process improvement is to improve the efficiency of certain steps by improving automation through integrated tools. Today's software development environments, combined with rigorous engineering languages like UML, enable many tasks that were previously manual to be automated. Activities such as design analysis, data translations, quality checks, and other tasks involving a deterministic production of artifacts can now be done with minimal human intervention. Environments should include tools for requirements management, visual modeling, document automation, host/target programming tools, automated regression testing, integrated change management, and feature/defect tracking.

Today, most software organizations are facing the need to integrate their own environment and infrastructure for software development. This typically results in the selection of more or less incompatible tools with different information repositories, from different vendors, on different platforms, using different jargon, and based on different process assumptions. Integrating and maintaining such an infrastructure has proved to be much more problematic than expected. An important

emphasis of a modern approach is to define an integrated development and maintenance environment as a first-class artifact of the process. Commercial processes, methods, and tools have synthesized and packaged industry best practices into mature approaches applicable across the spectrum of software development domains. The return on investment in these commercial environments scales up significantly with the size of the software development organization, promotes useful levels of standardization, and minimizes the additional organizational burden of maintaining proprietary alternatives.

Improving Human Productivity

Planning tools, requirements management tools, visual modeling tools, compilers, editors, debuggers, quality assurance analysis tools, test tools, and user interfaces provide crucial automation support for evolving the intermediate products of a software engineering effort. Moreover, configuration management environments provide the foundation for executing and instrumenting the process. Viewed in isolation, tools and automation generally yield 20 to 40 percent improvements in effort. These same tools and environments, however, are also primary vehicles for reducing complexity and improving process automation, so their impact can be much greater.

Tool automation can help reduce the overall complexity in automated code generation from UML design models, for example. Designers working at a relatively high level of abstraction in UML may compose a model that includes graphical icons, relationships, and attributes in a few diagrams. Visual modeling tools can capture the diagrams in a persistent representation and automate the creation of a large number of source code statements in a desired programming language. Hundreds of lines of source code are typically generated from tens of human-generated visual modeling elements. This 10-to-1 reduction in the amount of human-generated stuff is one dimension of complexity reduction enabled by visual modeling notations and tools.

Eliminating Error Sources

Each phase of development produces a certain amount of precision in the product/system description called software artifacts. Lifecycle software artifacts are organized into five sets that are roughly partitioned by the underlying language of:

1. Requirements (organized text and UML models of the problem space)

2. Design (UML models of the solution space)

3. Implementation (human-readable programming language and associated source files)

4. Deployment (machine-processable languages and associated files)

5. Management (*ad hoc* textual formats such as plans, schedules, metrics, and spreadsheets)

At any point in the lifecycle, the different artifact sets should be in balance, at compatible detail levels, and traceable to each other. As development proceeds, each part evolves in more detail. When the system is complete, all five sets are fully elaborated and consistent with each other. As the industry has moved into maintaining different information repositories for the engineering artifacts, we now need automation support to ensure efficient and error-free transition of data from one artifact to another. Round-trip engineering describes the environment support needed to change an artifact freely and have other artifacts automatically changed so that consistency is maintained among the entire set of requirements, design, implementation, and deployment artifacts.

Enabling Process Improvements

Real-world project experience has shown that a highly integrated environment is necessary both to facilitate and to enforce management control of the process. An environment that captures artifacts in rigorous engineering languages such as UML and programming languages can provide semantic integration (where the environment understands the detailed meaning of the development artifacts) and significant process automation to improve productivity and software quality. An environment that supports incremental compilation, automated system builds, and integrated regression testing can provide rapid turnaround for iterative development, allow development teams to iterate more freely, and accelerate the adoption of modern techniques.

Objective measures are required for assessing the quality of a software product and the progress of the work, which provide different perspectives of a software effort. Architects are more concerned with quality indicators; managers are usually more concerned with progress indicators. The success of any software process whose metrics are collected manually will be limited. The most important software metrics are simple, objective measures of how various perspectives of the product/project are changing. Absolute measures are usually much less important than relative changes with respect to time. The incredibly dynamic nature of software projects requires that these measures be available at any time, tailorable to various subsets of the evolving product (subsystem, release, version, component, team), and maintained such that trends can be assessed (first and second derivatives). Such continuous availability has only been achieved in development/integration environments that maintain the metrics online, as an automated by-product.

In next month's issue of The Rational Edge*, I will conclude this series with a summary of how to balance priorities and a few lessons we've learned*

about eliminating sources of friction associated with organizational changes targeted at improving software economics.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!

Copyright Rational Software 2001 | Privacy/Legal Information