Development governance for software management: Using IBM Rational tools to assess process variability and standards compliance

Roger Dunn

October 16, 2007

from The Rational Edge: Read how key performance indicators (KPIs) can be derived from a Rational infrastructure to gain insight into and exert management control with respect to development processes. Look for the Webinar link at the end of this article.

Industry advances, such as Agile and the current move towards dynamic languages\(^1\), are creating new challenges for maintaining management control over key development processes. This compounds the already sophisticated set of functions required to maintain development governance\(^2\). Addressing these challenges in their recent three-part article series on "Best practices for lean development governance"\(^3\), Scott W. Ambler and Per Kroll covered the span of concepts required for organizations to achieve governance with respect to their agile development initiatives: principles and organization; processes and measures; and roles and policies.

This article extends their discussion by examining best practices for specific metrics that management can use to enable teams in these changing times, and how an IBM® Rational® infrastructure can be leveraged to catalyze the benefits conferred through applying these practices. I concentrate on a subset of these overall best practices, elaborating on approaches and guidelines that management can use to initiate and support development governance. The focus is on key performance indicators (KPIs) that can be derived from a Rational infrastructure to gain insight into and exert management control with respect to development processes.

While this discussion extends and elaborates on Ambler and Kroll's recent series, it does not attempt to debate the relative merits of either Agile or more "high ceremony" approaches. If
anything, these approaches can be complementary from a software management perspective and, more to the point, the KPIs described for governance herein can be applied to software development processes generally.

**Governance is not simply management**

Albert Einstein once noted that "Concepts have meaning only if we can point to objects" to which they refer and to the rules by which they are assigned to these objects." In that spirit, I would like to define a common set of terms regarding management's roles and responsibilities with respect to governance. By doing so, I hope to avoid the confusion that comes from inadvertently conflating "governance" with "management," and "management" with "manager."

Development governance is a management responsibility, but management's responsibilities extend beyond governance alone. The role of management is to enable teams through timely and accurate information on team initiatives, project status and trajectory, and issue identification and resolution. From a governance perspective, this translates to ensuring conformance with the organization's governance program, ideally in a fashion that promotes the program to teams in a desirable and compelling manner.

Management normally serves two constituencies: the team and the customer. Management's focus on team enablement can be thought of as inward-facing, overseeing the personnel and operations for which management is responsible. To this is added a set of outward-facing responsibilities: communication with superiors and business unit counterparts, resource allocation and planning, budgeting and forecasting.

Achieving success with one or the other of these constituencies is hard work; balancing the needs of one's team and customer while providing adequate support to both can be downright daunting indeed. To this contemporary management must also now add the challenge of weaving a governance program into delivery and execution.

It can be a bit trickier to distinguish between "management" and "manager." The software industry has long applied a hierarchy that puts "manager" above "programmer" and "software quality assurance analyst" on the organizational chart.

Recent trends blur these distinctions. What is a manager? One definition is someone who is responsible not only for some aspect of the execution of a software project, but also for the successful outcome of the project, with the authority compatible with achieving this result. Formerly this has meant project manager, development manager, QA manager, release engineering manager, and the like. But in today's environments, and as catalyzed by movements such as Agile, the connection between execution and outcome is increasingly extended directly to developers, quality assurance analysts, and other knowledge workers.

It's a healthy sign that the software development community is embracing the lessons gained in other disciplines that favor enabling teams with the timely information and process insights required to excel at their tasks. For the individual developer or quality assurance analyst the lesson
is pretty clear: it may not say "manager" on your business card, but you may well be a key part of development management.

**The governance triad**

As the software industry matures, the terms "governance," "compliance," and "standards" are taking increasingly prominent roles. These terms have specific power and meaning, but are only useful when that power and meaning is well understood by all the affected parties in the software development process. To establish a basic framework for conceptualizing these terms, consider the stack shown in Figure 1.

![Figure 1: The governance triad](image)

Considering the terms in Figure 1 from bottom to top:

- Standards are the benchmarks against which work products and processes are measured. Development organizations typically have standards, though their expression may be formal or informal. Regulatory requirements may also call for standards and standard operating procedures.
- Compliance is the assessment of work products and processes against standards.
- Governance is the act of exerting management control to reinforce practices that support compliance, and to rectify practices that are in non-compliance.

By teasing apart these three terms and giving each specific scope and meaning, we are able to take this static representation from Figure 1 and set it in motion, as shown in Figure 2. When viewed as a dynamic process, it becomes clear that each component of the governance triad feeds another component, informing the management and executive audiences of compliance with standards, so that appropriate action can be taken.
The desired results are to catalyze the maturation of the organization, achieve increasing levels of operational efficiencies, and to create standardized, predictable processes that can survive the scrutiny of internal and external auditors alike.

Keeping in mind the emphasis on outcome I noted in the previous section, it is management's job to put the "govern" in governance, and exert appropriate controls to achieve these desired results by supporting compliance with standards and correcting non-compliance.

Without compliance reporting, the cycle is broken. Management is rendered largely ineffective, and project objectives and the team’s success are jeopardized. Standards lose their vitality and context, with teams ignoring them through lack of familiarity or deliberate avoidance. When management attempts to exert controls, the effort is sometimes viewed by the team as naïve or even capricious, and with no mechanism in place to evaluate the impact; good, bad, or indifferent. As the old adage goes, if you don't know where you are going, then any direction will do.

For this dynamic process to work, compliance reporting is imperative, equipping management with ongoing visibility into key development processes. When teams assess compliance with standards, they regain their management compass and are able to chart their course forward with greater confidence in achieving the desired outcome. They gain the ability to evaluate the impact of management efforts, so that those too can be scrutinized for risks and benefits. They move from the sort of chaotic, hero-driven team dynamic experienced by Capability Maturity Model® Integration (CMMI)\(^6\) Capability Level 1 organizations to the more stable, predictable, standards-based results marked by Capability Level 3 and beyond\(^7\).

### Key performance indicators

Some very clever people -- from Lord Kelvin to W. Edwards Deming to Philip B. Crosby -- lived by the notion that one cannot manage that which one cannot measure; process KPIs are today's application of their teachings.

Certain development standards, such as verifiable signoffs and checkpoints, can be requirements for regulatory and audit compliance. Compliance with these standards speaks to the integrity of the team, and its ability to manage change and execute responsibly. But it may not contribute tangibly to increasing team productivity or the measurable quality of software work products.

So without diminishing the importance or necessity of the more procedurally-oriented audit standards, let's consider engineering standards that can be used by teams to materially enhance development processes. As a whole, these standards serve as KPIs for achieving team results.

KPIs generally fall into two categories: process and work product. Process KPIs allow management to get a handle on the variability within their development processes, so that the root cause(s) and attendant risks for this variance can be understood, managed, and ultimately contained. This assessment process is an ongoing activity for management, and is instrumental in achieving and maintaining predictable, repeatable processes.
Considered relative to CMMI, expanding the scope of standards relative to development processes, and raising the relevance and visibility of compliance to these standards to management's attention, are key ingredients for moving from the "managed processes" that characterize Capability Level 2 to the "defined processes" of Capability Level 3. Process KPIs gain heightened relevance and utility as one moves to the "quantitatively managed" and "optimizing" states associated with Capability Levels 4 and 5, respectively.

The specifics of process KPIs are best considered relative to the process itself, and for that one must be somewhat discerning about development methodologies, practices, and procedures. However, processes have certain general characteristics that apply across methodological boundaries. Perhaps you have a software factory with a strong component orientation, or perhaps you have vertically stratified silos of development serving business unit segments. Perhaps you are using waterfall development methods, or perhaps you've adopted Agile. Whatever course you have set for development, that course possesses discrete processes. Hopefully these processes are well defined and generally understood by the team, but even CMMI Capability Level 1 organizations pursue processes, after a fashion. Further, processes possess certain defining characteristics. They have inputs, outputs, and checkpoints, or to put it in the more colorful language of one CTO, they have "gazintas, gazoutas, and ga-whaaaaat?"

**Process KPIs**

Process KPIs are based on the general characteristics described above, and break down into two areas: volatility and volume.

The audience for process KPIs is typically people who have responsibility for understanding and optimizing the process, and for whom monitoring trends over time yields insights with respect to the compliance (or non-compliance) with the process. From a development governance standpoint, process KPIs are often more impactful and actionable for people who manage programming activities, and less so for people responsible for quality assurance. Where true, the general cause for this is that quality assurance does not perceive itself to be in a position to affect the development methodology, or to mandate a change to it; this is the classic separation of functional disciplines, the "12-foot brick wall topped with razor wire between programmers and QA" that is still common in the industry. In more progressive development cultures, developers and quality assurance partner more actively on test methodologies and compliance, establishing consensus on the standards to be applied, and achieving more with process KPIs. However, whether classic or progressive, understandings of volatility and volume are fundamental to assessing software projects.

**The volatility KPI**

A process undergoes a certain amount of volatility as it moves from start to finish. Volatility measures the amount and rate of churn accomplish a goal. Volatility is typically subject to two measures: predicted and actual. The predicted volatility expresses the estimated amount of gross churn that will take place to accomplish a task, milestone, or project, whereas the actual volatility measures what actually took place. From a management standpoint, volatility is a familiar concept,
with correlates to project planning exercises that also incorporate notions of "budgeted versus actual."

When management has an understanding of predicted versus actual volatility, it is able to assess variance from plan and take steps to rein in projects that are heading out of control. Fundamentally, this is a governance issue: assessing compliance with standards -- the predicted project volatility curve -- to equip management to appropriately enable the team to achieve a successful project outcome. If you want to achieve standardized, predictable processes, concentrate on assessing and managing the volatility of those processes. This is especially true for software maintenance activities, which often follow established patterns more readily than new development.

An excellent metric for volatility is gross change per unit time, with the overall assessment being the trend over time for this metric. There are a variety of ways to measure gross change: lines of code (LOC), revisions, bugs fixed, enhancement requests satisfied, and so forth. Teams should discuss these options relative to their approach and methodology before settling in one or the other, and should maintain an open mind towards evolving this metric as they advance in capabilities, and as new technologies and approaches become available. As a starting point, a simple measure of gross LOC changed per unit time across a project phase will chart the trajectory of the project, and reveal whether or not the project is achieving a "glide path" compatible with a successful landing at the completion date. Often, even mature development organizations will find, upon their first inspection of the volatility KPI, that their processes are subject to unexpected and unreported "late-breaking" changes that sneak past peer review and quality assurance scrutiny, or that their process contain latencies and lag times that are contrary to their desired Agile methods.

The volume KPI

The volume KPI speaks to the amount of logic or other relevant artifacts that are produced in development. In some organizations, the notion that the volume of logic may have relevance has been controversial, at times garnering a degree of skepticism and even contempt. However, the basic idea of "volume" is fundamental to three software development disciplines: project forecasting; quality assurance; and programming.

A significant contributing factor to the cost of maintenance is the size of the code base in question. It is generally true that it takes more people to maintain institutional memory and stay fresh on a large code base than it does a smaller one. One may argue as to whether or not maintenance efforts correlate linearly to the size of a code base, and in fact the answer to this question may be a function of factors including your software design approach, degree of componentization, resource skill level, centralized versus distributed teams, and use of outsourced or offshore personnel. However, a great way for your management to nail down the correlation between the size and composition of a maintenance team with the size and composition of the project that the team is maintaining is to begin measuring and applying the volume KPI.

This KPI can yield an added benefit for project forecasting in that it allows the software team to communicate, in general terms, the efficiency and productivity of the team in responding to customer requirements, enhancement requests, and software defects. In environments that follow
a customer-funded model for software development, such as many corporate IT departments, the volume KPI can be a valuable visualization tool to put the software team and its business unit counterpart quite literally on the same page.

Volume is also fundamental to quality assurance, as the size of the software product is the denominator for the defect density ratio. Observing the trend of volume over time brings out some of the subtleties around defect density, and help to reinforce the insights from this KPI. For example, the volume of a product may expand rapidly in a short span of time before quality assurance has had an opportunity to find and enter the defects associated with this new code; this will have the effect of seemingly decreasing the defect density, thereby masking the fact the actual situation may call for augmented test scripts or test automation to audit the new code. An analogous situation can happen when code is refactored to collapse the code base down to a more carefully rationalized set of routines or components. When the overall code size drops, defect density increases, making it seem as though the overall quality level is worse even though the code base has moved to a place of improved maintainability and superior code coverage. By monitoring the volume KPI in concurrence with defect density, quality assurance can observe these important trends, and plan accordingly.

Programmers tend to have a fairly instinctive reaction to volume, but one that is typically not informed by visibility into the overall trend in the change in the size of code base, as that trend can be time-consuming and difficult to derive and visualize. But the programmer's gut check is deeply meaningful, and goes to the heart of the issue. Programmers know when they're dealing with bloated code, and are aware -- often keenly so -- of its negative impact on the project. In this case, just as defect density is a retrospective measure of the efficacy of programming, programmers may intuitively apply volume as a retrospective measure of the efficacy of a software design. When visualized by programmers, rapid, unexpected increases in volume are highly suggestive of poor design. They reveal technology stacks that are a mile wide and an inch deep, that do not lend themselves to testability or maintainability, and that generally offend one's sense of design elegance.

Typical measure for volumes include LOCs, source LOCs (SLOCs), effective LOCs (ELOCs), statements, semicolons, method count, class count, and file count. The industry has generally favored LOCs of one sort or another as an indication of size, as shown by defect density almost invariably calculated as defects per LOC.

Because volume can be perceived by some as a controversial metric, occasionally organizations get bogged down debating what volume measure is "best." To a certain extent, if you are not yet measuring volume at all, it's somewhat meaningless to dwell on this debate. Few software metrics are great, but many (like LOCs) are good; what's important is to enhance management's governance capability and the team's visibility into this important dimension of their work, knowing that the KPI can and will be improved upon over time.

In general, it is a recommended practice to assess volatility and volume with the same basis. If you are measuring volatility as the gross amount of change over time in LOCs, then you will want to measure volume as the net change over time in LOCs as well.
Work-product KPIs

There are many tools available for assessing the quality and integrity of a software work-product, whether it is a component, subsystem, service, or application. The selection and use of these tools forms an integral part of an organization’s governance program.

In general, these tools apply a set of rules to a static (source code) or a dynamic (runtime) environment. When brought into a governance program, rule violations typically identify non-compliance with engineering standards. At first blush, this may sound a bit harsh, over-emphasizing the negative without some offsetting silver lining. But there is a practical reason: It is relatively easy to test for the presence of non-compliance, but much harder to test for compliance other than it being the lack of non-compliance. This governance topic sometimes sparks debates that sound vaguely philosophical, like "Is good simply the absence of evil?" While the answer to that one is a bit intractable, it’s certainly the case that tools can find security holes, complex code, and performance bottlenecks that are evil indeed.

The output from these tools becomes work-product KPIs when it is brought to management’s attention. The problem with many of these tools is that they are designed to be used by individual developers or from within runtime environments where the assessments are performed with irregular frequency and the results not persisted. In order to participate in a governance program, this usually means integrating the tools with a consistent framework for analysis and reporting.

Accepting for the moment that vendors such as SourceIQ have bridged the gap between tools and management reporting, let's examine the work-product KPIs that can have a rapid and positive impact on development governance: coding guidelines, language guidelines, and complexity.

Coding guidelines

Organizations adopt coding guidelines for the best of reasons. Code developed under common guidelines is more accessible and maintainable between team members, is easier to share across a large organization, and is more cost effective in maintenance, especially when transferred to a team that was not part of the original work.

Unfortunately, it is often the case that these guidelines are subject to selective application and erosion over time by individuals on the team. Without work-product KPIs, this degradation in the code base occurs with management being unaware. Consequently, when new team members have a hard time coming up to speed, peer review becomes awkward to the point of unfeasibility, or maintenance costs skyrocket, the root cause is invisible to the management that is trying to rectify the problem.

The key shift in thinking occurs when team members embrace their responsibilities to their colleagues, and incorporate the coding guidelines designed to make everyone's job a bit easier. There are plenty of tools and scripts that can be customized for an organization's particular coding guidelines. When these tools are brought into a governance context, management acquires the ability to nudge the development organization in a direction of increased compliance. There may be some ruffled feathers on the way, which is a healthy thing as it may further improve the
guidelines in place. The cumulative effect tends to be very positive, with team members being able to share code, debug in teams, and generally understand each other's work.

**Language guidelines**

Language guidelines are similar to coding guidelines, but typically come from outside the development organization, and not from within. For example, Sun Microsystems publishes a set of coding conventions for the Java language. Commercial and open source tools exist that automate the assessment of code against the language vendor's guidelines, typically with a rule engine or grammar that makes it easy to adapt the rules to meet the needs of your development governance program. In some cases, the language vendor's rules are fairly benign, and may impact only some small aspect of the expressiveness or maintainability of the code. However, in other cases the presence of a rule violation in one's code can be a red flag that something can go seriously awry at runtime, even if the compiler lets it slip by.

As tools such as these are incorporated into a governance program, it is worth noting that software tools are undergoing rapid change and evolution in the industry today. IBM®'s recent acquisition of Watchfire underscores the maturation among tools that can pinpoint security vulnerabilities, and it may make sense for you to consider the relation of this sort of work-product KPIs relative to your development objectives and audit requirements. The important thing is to keep on top of industry trends, and stay informed.

**Complexity**

Where the aforementioned work-product assessments are typically qualitative, complexity analysis forms the basis for a powerful work-product KPI. Even more than coding guidelines, complexity analysis is the foundation for identifying the riskiest, more error-prone, most difficult and costly-to-maintain code in your system. Many organizations have failed to take advantage of this key metric, because it has been perceived as too time-consuming or too difficult for already burdened development teams to produce. But with the advent of automated systems for deriving management metrics, such as SourceIQ, management can now incorporate complexity as a vital dimension to its development governance program.

**Deriving KPIs from IBM Rational® tools**

Organizations that pair assessments against standards, expressed as KPIs, with their Rational infrastructure achieve true compliance reporting with respect to those standards.

This may seem like a bit of a leap, but let's jettison some historical baggage. The Rational suite is seen by some as a collection of tools: developers use IBM Rational ClearCase®, quality assurance teams use IBM Rational ClearQuest®, etc. But a Rational infrastructure is more than a set of tools. It is a set of integrated tools that contain a treasure trove of facts and transactions about key development processes, and that can be mined and analyzed for compliance reporting against standards.
A Rational infrastructure is the basis for deriving management KPIs required for effective governance, and ensuring that the dynamic governance triad mentioned earlier remains intact, has rapid response, and is effective across teams whether domestic, offshore, or globally distributed. With Unified Change Management (UCM), that infrastructure becomes even more powerful, and its insights even more focused.

Three members of the Rational suite provide all the foundation you need: ClearQuest, ClearCase, and IBM Rational Build Forge®.

ClearQuest provides the chronological repository of facts and transactions with respect to enhancement requests and defects required to assess the process KPIs that enable governance of the endpoints of the development process: incoming requests and outgoing audits. There are obviously alternatives to ClearQuest, but it has a singular advantage with respect to UCM, noted below.

ClearCase is the chronological repository from which one case derives the state of the process and work-products across time. With UCM, ClearCase gains in governance value, acquiring the notion of an immutable repository for change that is required for many external compliance audits. The singular value of this repository is explored in the next section; the reader will no doubt be acquainted with configuration options around ClearCase such as UCM and multi-site ²⁰.

ClearQuest-enabled UCM is the ideal configuration for empowering a development governance program with traction. Whether your development approach is Agile or waterfall, this technology eases the difficulties of standardizing on development processes, and provides vital context to the transactions that represent change through the course of change orders, coding, and test.

While ClearQuest and ClearCase provide the data and the context, Build Forge is the vehicle for automating the generation of compliance reports. Since many organizations pursue nightly builds or continuous integration, this addresses the issue of the frequency and periodicity of compliance reporting, and enables management to be more responsive to addressing non-compliance situations.

What endures?

The software development industry is constantly re-inventing itself, cannibalizing old ideas and approaches and transforming them into something new, different, and hopefully a bit better. This flux has a direct impact on the standards that can be put in place, the management controls that can be exerted, and the governance that can be achieved.

For example, many organizations are keenly interested in gaining from the benefits of dynamic languages such as Ruby and PHP. But many of these same organizations cannot accept the risks attendant with a development process that they perceive as "author, publish," a process that lacks the more structured checkpoints that come with traditional languages with their compilation and linkage stages, and tooling around those steps.
Sometimes it's helpful to take a step back, assess priorities, and ask the question: What really matters? After examining this question with many development organizations, SourceIQ has arrived at a rather unexpected answer, one that is related to the very churn that we see in the software development industry.

Before I give away the answer, let's look at how things churn. Software tools churn pretty quickly, with vendors issuing major new updates every year or so, sometimes even faster. Programming languages churn a bit less, with major shifts every seven to ten years. Software metrics that form the intellectual basis for KPIs churn even more slowly, many dating back two to three decades.

But empirically we've found that the most enduring asset of software development is the actual source code itself that organizations produce. Once created, it may be subject to seemingly never-ending maintenance. But the sunset rarely seems to fall on code. Companies that were doing mainframe IT development in the 1960s are still running core systems based on that original logic.

Requirements, software designs and even test scripts have relatively short life expectancies compared to code. While important, they seem to become stale quickly, fall into disrepair or disuse, and are discarded. But code is not only long-lasting, it can even acquire an insidious quality of being built into systems over and over again, even long after it has ceased to actually be referenced by anything!

In no way is this discussion meant to take away from the level of rigor and scrutiny that should be applied to requirements definition, design, and test. If anything, the artifacts from these disciplines might well benefit from becoming as integral to the long-term process as the code seems to be.

But from a purely pragmatic standpoint, the enduring nature of code highlights a priority for a development governance program. The programmers will come and go on a project. The code may move to a different continent. The platform it runs on will change. The company name on the office building may even change, but the code will endure in ways that defy belief. Consequently, the more management can know about a project's code base, its provenance and current status, and its assessment against a core set of meaningful KPIs, the more empowered the team will be in adapting that code base to meet coming demands.

**How do I get started?**

If you have an IBM Rational infrastructure, a need for development governance, and you're itching to get started on some management KPIs, the first thing to do is start asking some questions. What standards are already in place? What standards are missing? Is standards compliance a priority for the development organization? If you have outsourced or offshore partners, are your standards part of their service level agreements?

If governance is to be catalyzed by and predicated on a change management infrastructure, you'll want standard usage models. Are these models in place within and across project and teams?

Another set of questions relates to the audiences for the management KPIs. While metrics are nothing new, some folks may need to refresh their understanding of the KPIs, and the relevance of those KPIs to achieving development governance. The best way to build consensus and shared
understanding is to engage with all the audiences that are affected by development governance: architects, developers, quality assurance analysts, release engineers, and perhaps even business unit liaisons. When people start talking and questions of each other, they begin to have a stake in the outcome and take on ownership for the success of executing the governance program.

Finally, governance is a management function, and the buck has to stop somewhere. Who is responsible for governance? Who is accountable? Do the people who are responsible for exerting governance, implementing to standards, and assessing compliance have authority compatible with performing their job functions?

As the standards start to take shape, focus on a small, core set of metrics and KPIs that will resonate across the development team, one that offers superior management insight, and that can be adopted and understood in a short amount of time. This article describes a set that achieve that effect with most teams. Your team may have its own complexities or nuances, and may demand something different, but the guiding principle of "less is more" still applies.

Once you have arrived at an initial set of standards, it's time to plan the implementation. Unless management KPIs are integral to enforced regulatory compliance, they may need a little boost to help them gain traction. The best approach is to ensure that the KPIs offer value to the management audiences, and that the benefits can start to be realized quickly. Results materialize once management starts to use process KPIs to gain traceability forward through a process and an audit trail looking back, and when software projects move to a state of lower cost of maintenance and improved robustness at runtime.

Roll out the implementation on a project-by-project, or team-by-team basis. Organizations that attempt an enterprise-wide program of project- and code-conformance tend to not get beyond their own bureaucracy, and consequently are thwarted in their efforts. A leaner approach that emphasizes the success of a project by empowering a pioneering team is more likely to generate enthusiasm and success, with the added benefit of fostering development personnel who will cross-pollinate their next project with the valuable lessons learned.

Finally, remember to leverage your Rational infrastructure. Management KPIs usually don't come from developer IDEs or from tools gathering dust on some shelf. They come from hardened infrastructure components like Rational that are secure, that can survive the scrutiny of an external audit, and that yield consistent answers.

Notes

1. For more on dynamic languages, see Gary Pollice's articles in the June, July, and August 2007 issues of *The Rational Edge*.
2. See Scott Ambler's governance column in the upcoming November issue of *Dr. Dobb's Journal* for a discussion of how Agile approaches provide even greater opportunities for governance.

5. For example: Total Quality Management (TQM), as defined by the International Organization for Standardization (ISO), widely used in manufacturing, government, and service industries; ISO 16949 (Quality Management Systems: Automotive).


7. CMMI is used in the context for its characterization of capability levels. It may be worth noting that the author agrees with others (Hillel Glazer, Mike Konrad, James Over) that Agile and CMMI are not adversarial approaches.

8. To borrow from the work of Gary Pollice ("Does Process Matter?," *The Rational Edge*, August 2006), the emphasis of this discussion leans more towards enterprise process than team process, but is applicable across this spectrum right down to the micro-processes he describes.


10. *Software Configuration Management Strategies and IBM Rational ClearCase: A Practical Introduction* (2nd Edition) by David E. Bellagio and Tom J. Milligan is a valuable guide for understanding the constructs available to standardize ClearCase usage and gain the most benefit in development governance.

© Copyright IBM Corporation 2007

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)