IBM Software
Thought Leadership White Paper

August 2013

Smarter quality management

The fast track to competitive advantage

IBM

Organizations that create and deliver software—whether for their own IT operations, for the packaged applications market, or as the core of their final product, as in the systems space—must grapple not only with today's tough economic climate, but also with increased complexity in their processes and supply chains. Many factors serve to complicate software delivery, but competition lies at the heart of this complexity.

Here are a few examples. In the products arena, customers demand more from the software components designed for the latest hardware, often with requirements that change rapidly, even as software projects are underway. Keeping track of changes while meeting aggressive (and unaltered!) deadlines is difficult, if not impossible. In the IT space, more businesses are focused on their operational software for capturing and providing value to their customers and lines of businesses. E-commerce websites compete to improve customer relations and simplify online business; businesses that create highly optimized supply chains supporting a fast, efficient ecosystem of partners quickly rise in the marketplace.

What does this competitive environment mean for businesses seeking to deliver high-quality products and services? Certainly, effective quality management and continuous testing create opportunities to deliver key business benefits, such as improved market share, higher customer satisfaction, and increased brand equity. But top quality in the completed product cannot serve as the single guiding principle by which products are produced and delivered. Time to market is also key; costs and risk factors must also be part of the balancing act. Get these things wrong, and you may face unsustainable costs, missed windows of opportunity, unhappy customers, even a massive recall or the complete failure of a system at a critical moment. Get these things right, and you can achieve a positive operational return on investment from efficiencies gained in development activities.

One of the biggest challenges related to quality management is how to invest intelligently to minimize risk, given economic constraints. However, figuring out a) how to relate quality to business outcome and b) what constitutes the right level of quality for individual products is not always clear.

This paper introduces a practical approach to quality management (QM), continuous testing, and service virtualization that helps reduce time to market without sacrificing quality in the outcome. The underlying concepts presented here will be familiar to software project managers, especially those with QM experience, but we will explain some fundamentals as we go along to ensure all readers seeking these benefits can understand the essential processes involved.

The nature of software development

Here's one way to understand the "soft" in software: it is relatively easy to change. But for software designed for the commercial space, where the competitive pressures described above govern a software project's success or failure, the "soft" feature happens to be one of its riskiest attributes. That's because software projects are seldom designed and manufactured as in traditional engineering projects—a bridge, for example. While a bridge is engineered through traditional planning and architecture based on the laws of physics, then produced according to an organized plan with a division of labor, software is, at its essence, simply information. Its development typically resembles a more creative process than one bound by the laws of nature. Walker Royce, IBM Software Group/Rational's chief software economist, compares software production to movie production: a collaboration involving a team of craftsmen and emerging from the naturally creative process of artistic yet technical people.

Over the past two decades, this unique feature of software has been understood and embraced by iterative development practitioners, who now tend to develop software in stages called "iterations"—with many of these practitioners following agile development practices outlined in the Agile Manifesto. Each iteration delivers a working, functional version of the software under development, so it can be reviewed, tested, and vetted by stakeholders and other teams seeking adherence to the original project requirements. This allows project managers to make smaller, incremental course corrections during the project lifecycle—thus ensuring the final deliverable is close to expectations—as opposed to having separate teams work according to a plan, assemble various components near project end, and discover major failures due to integration or deployment complexities.

For testing teams, the iterative development process integrates quality management and continuous testing across all stages of the project work flow, as opposed to relegating test activity to the end of the project. We will describe the role of iterative development-based quality management and continuous testing more fully in future sections.

Quality management in the software development lifecycle

What is the role of the testing, or QM, team during the iterative lifecycle? What do they test for, and how do they know what is changing from one iteration to the next?

As noted above, traditional software testing usually occurs late in the lifecycle, after multiple coding teams have spent much time and effort to deliver their components toward the complete project. Because these traditionally managed projects proceed according to strictly described requirement sets, and various component teams focus on their portions alone, it is up to the testers to discover the discontinuities and malfunctions as these components are assembled—then it's the testers who must deliver the bad news that much rework has to be done in order to get the project back on track.

Iterative software development techniques improve on that scenario by introducing test teams to the process much sooner. A relatively modest, first iteration may only address 15 percent of the full set of project requirements, but as a functional module of working code, the completed iteration can be demonstrated, and continuously tested. So any defects discovered by test teams at this early stage have a proportionally small impact on the larger development team, who make the fixes, then proceed to the next iteration where more of the requirements can be incorporated into the working version (iteration) of the software.

The number of iterations required by any software project depends on many factors, of course, such as the complexities of the team's supply chain, the complexity of the software under development, the physical location of team members (are they geographically distributed, perhaps internationally? or are they co-located under a single roof?), and the competitive demands that determine optimum time to market. During any software delivery process, "When do we release?" is a key question for the business with no simple answer. The business must consider project-specific variables, such as the cost of delays, the opportunity value of early delivery, marketplace quality expectations, and the costs associated with defects. Ultimately, the delivery strategy will be based on the actual or perceived importance of each variable.

The optimal time to release

The optimal time to release is when the total risk exposure is minimal, typically around the time where the risk associated with competitive threats starts to outweigh the risk reduction associated with further quality improvements, as illustrated in Figure 1.

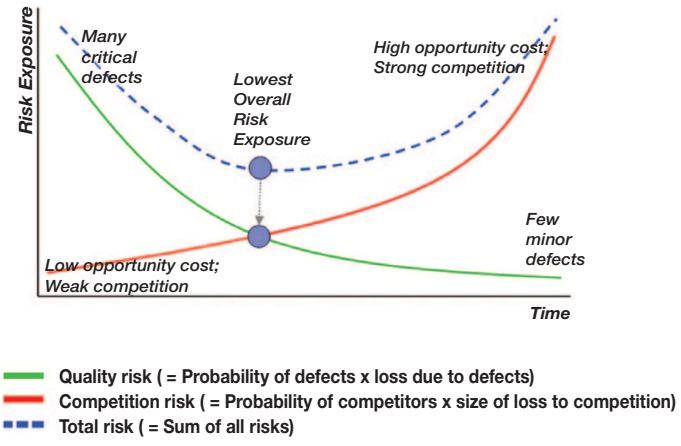

Figure 1. Minimal risk exposure is when opportunity cost and competitive threats outweigh risk reduction related to quality improvements.

The best time to release varies widely based on your software delivery strategy and your target market. Software delivery for both IT (internal business systems) and smart products (products using embedded software, including "system-of-systems" design) is typically dominated by one of two motivators, depending on the organization's target market: time to market (schedule driven), or quality impact (quality driven).

Schedule-driven delivery implies "deliver on time, regardless of other factors" and is often used in industries where "Time to market is king." Consumer electronics is one good example, as well as automotive, segments of the medical industry, and other markets, where product teams try to gain a first mover advantage over their competitors, (almost) regardless of the risk associated with inadequate quality. It should also be noted that schedule-driven delivery is not limited to the systems space (i.e., the many embedded software devices industries). Many IT development teams use schedule-driven delivery when trying to enhance their end user experience and increase market share, taking away from their competitors, often risking quality in the process.

Figure 2 represents the risks associated with schedule-driven software delivery. The green line represents the risks associated with the delivery of your product being reduced over time. The red lines represent the risk of competitors stealing your market away increasing over time, as well as risks associated with opportunity costs.

The intersection is the point in time where the sum of both lines, i.e. the total risk, is the lowest. As seen in Figure 2, this point moves to the left as the environment you are in is more competitive in nature. (Notice that the intersection point is moving up as well.)

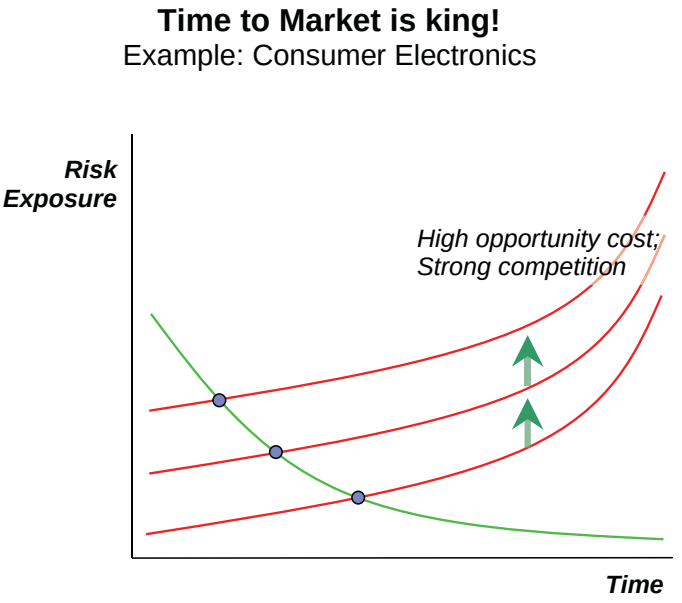

Figure 2. The blue dots show possible release points, with points of minimal risk moving forward as competition intensifies (red lines).

Quality issues are often magnified in a schedule-driven lifecycle, given that software contractors frequently get paid on a time and materials basis, regardless of the quality of software they deliver. In many cases, you may even end up paying extra for the delivery team to fix their own defects, so the potential costs of defects to the end user can add up quickly. And here's an interesting statistic: According to the Carnegie Mellon Software Engineering Institute, "Data indicate that 60 - 80 percent of the cost of software development is in rework."[1]

Quality-driven delivery can also be costly but for different reasons. As shown in Figure 3, the more critical any defects might be regarding quality, the longer it takes to get to an optimum release point.

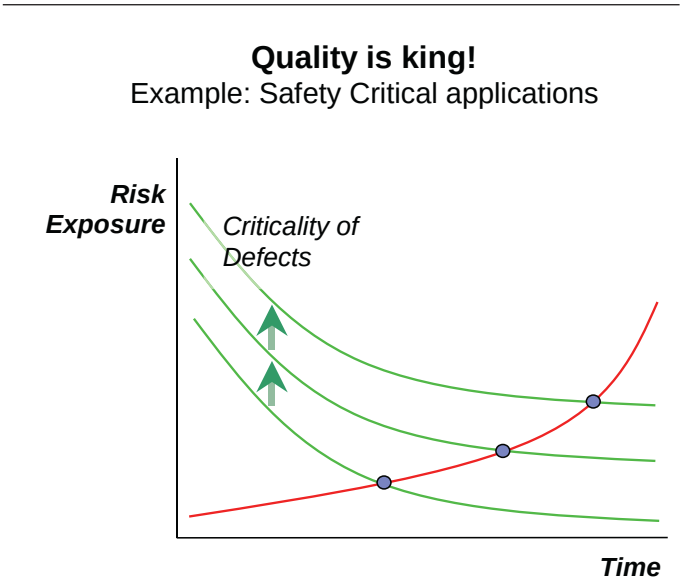

Figure 3. The more critical the implications of defects are, the more time it takes to get to the lowest risk point where release is possible.

The release timing for this approach is governed by achieving the right quality, moving the optimal time to release further to the right—but how do you define that? Zero defects is practically impossible to achieve, given that there is no way to determine how many defects still exist in a piece of code or the

probability of detecting those defects in use. A target based on "defects fixed" might be more realistic—but it's still impossible to know the number of remaining defects in the product.

Risk-driven delivery implies delivering your software when the risk is minimal. But in practice, we always need to release "early"—earlier than we can. Which typically implies increasing the risk, right? At least this is a commonly held view, but is it always the case?

Within the risk-driven model, the optimal release time is when risks are sufficiently reduced (not completely eliminated) and time to market has not been wasted. In other words, some time is needed to reduce the most significant risks, but the company cannot afford to address every known risk because the opportunity to beat the competition is fleeting.

So the question is, how can we get to this point sooner? How do we compress the release date from the optimal intersection (shown as a blue circle in Figure 4) to a point earlier in time?

We cannot simply cut the time requirement, because as we move left on the green line, the risk goes higher. But what if we could compress the curve described by the green line—push it down, so to speak? Then we could not only deliver sooner but lower the overall risk as well. The intersection point will move down and to the left. This improved scenario is shown in Figure 4.

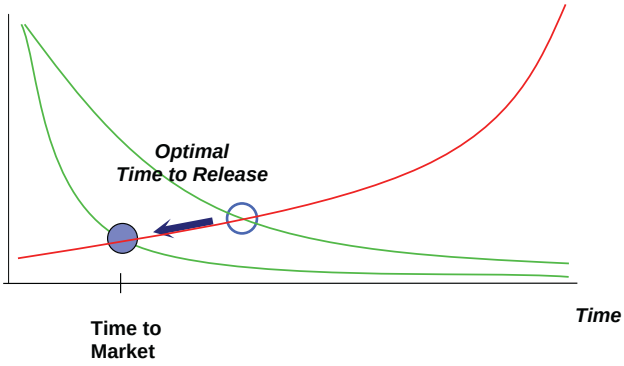

Figure 4. To deliver early, at an improved quality, reduce your risk at an earlier point in the lifecycle.

Risk-driven delivery offers a practical improvement over these two extremes (i.e. schedule driven vs. quality driven) because it more cost-effectively balances quality versus time-to-market considerations. A risk-driven strategy is a refinement of a quality-driven approach that optimizes risk exposure against development cost and time.

For the remainder of this discussion, we will assume that the software delivery lifecycle is based on a risk-driven approach. We will explore how to bend the green curve shown in Figure 4 downward and to the left, for reduced time to market without compromising the risk profile.

Understanding quality management: It's more than simply testing

If a faster reduction in risk is the goal, how do you achieve it? In traditional testing practices, testing is considered a late stage activity, squeezed between an often-late development handoff date and an immovable ship date. Not only does this practice fail to yield the benefits of incremental, iterative development techniques explained earlier; it also minimizes, or at best reduces, the amount of time spent on quality assurance, and makes fixes all the more difficult unless you're willing to compromise the release date.

As noted earlier, iterative development techniques greatly improve this situation by having functional units tested continuously throughout the lifecycle, rather than leaving the testing phase until project completion.

And quality management takes this improvement a step further.[2] Quality management, which is the implementation of best practices to proactively reduce risk throughout the whole lifecycle, is a risk reduction mechanism in its own right. By choosing quality management practices with the potential to deliver a positive ROI within a relative short amount of time, you can justify risk reduction measures from not only a quality standpoint but also a financial standpoint.

Approaching quality from a full lifecycle perspective should not seem like such a radical idea. After all, testing a product is an engineering task just like development: the requirements need to be analyzed by the test architect, its testing strategy has to be defined, the relevant test benches and test frameworks need to be built (designed and implemented), etc. These engineering development processes are very similar to the ones followed during product development. In fact, product development and quality management can be viewed as two parallel development threads emanating from the same requirements, with many synchronization points between the threads, up to the point of the testing activity itself where a verdict is made based on meeting expectations or not (as expressed in the test cases). This applies to both the traditional development process as well as agile methods (with variants such as test-first development and test-driven design). The important point is that QM is a thread that must run in parallel to development, especially in agile development.

Although a complete discussion of QM is beyond the scope of this paper, we can show how QM reduces risk—and thus allows earlier software delivery without compromising quality—by demonstrating how one of the quality management best practices can improve iterative testing within the software development lifecycle.

- **Continuous testing and service virtualization to avoid the "big bang"**

  It has already been shared that in traditional development practices teams typically defer testing until late in the development lifecycle. But, why does this happen? The logical explanation is that teams simply can't test critical business scenarios end to end until all the components have been developed and deployed in a test environment. However, when all of the "pieces" are brought together for the first time, many

organizations experience what is referred to as the "big bang" and a large number of defects are discovered. Some of the defects are so major teams are forced to rethink their application architecture or design decisions and return to the drawing board. Some may in fact consider starting over. So, it might be said that deferring testing until later is injecting unnecessary risk and could delay software release.

As a way to address this risk, Agile techniques, like test driven development, force teams to write tests before a single line of code is developed. However, to what level do these tests validate the application as a whole? Are the developers really writing tests which validate the dependencies between application components? One technique your organization may have undertaken is to create mocks or stubs for simulating missing functionality, but this ad hoc practice is not a very good use of a developer's time or your money. Organizations will also soon realize that this approach to enabling continuous integration testing is not scalable and cannot be easily incorporated into one's automated build and deployment process. Yet many believe the combination of continuous testing and continuous deployment is a critical need. So, how do teams make the unavailable available for earlier and continuous testing without having to write ad hoc stubs?

This point is where service virtualization comes in. Service virtualization simulates the behavior—functionality and performance—of select components within an application to enable end-to-end testing of the application as a whole. By creating and deploying virtual components to simulate the missing functionality, functionality provided by components unavailable due to a number of reasons, development teams can achieve continuous testing of the application end-to-end much earlier, discover defects sooner, and reduce project risk so teams can release software more often.

One must also look at the value service virtualization can bring to an organization when considering the cost of quality. Testing is expensive and that cost is increasing as applications become more complex. You need not look beyond the cost of provisioning a test environment to see how service virtualization can save money and time. Test teams spend a large portion of their time dealing with testing interruptions—interruptions like waiting for test environments to be available, recycling test environments, or deploying the latest build so they can begin testing.

In fact, this could account for 40 percent or more of the tester's time. Service virtualization decreases the amount of time testers spend on dealing with interruptions, Taking it to the next level, teams that incorporate automated and continuous testing as part of the deployment process receive constant feedback on the quality of the latest release. This approach also frees up the tester's time and allows them to increase the level of testing by executing more value-added activities like exploratory testing—which means testing applications to determine how they actually work and where significant defects are discovered.

- **Improved collaboration between the QA team and other stakeholders:** From talking to customers, we learned that on average, a tester spends only about 60 percent of the remaining time performing actual testing, test planning, or test reporting. The other 40 percent is related to activities that are collaborative in nature, such as clearing up the requirements with domain experts or business analysts, or exchanging emails and phone calls with members of the development team. This gets worse in distributed organizations. If you could track and manage the collaboration, it will not only reduce your risk associated with lack of communications and misunderstandings, but also reduce the time for collaborative tasks by 20 - 50 percent.
- **Automated reporting:** Creating a report, especially one that goes to high level management, requires data collection from many sources, sometimes from teams that are in different time zones, and then formatting this data appropriately. If you could automate this activity, your team will probably use it more often and take the appropriate decisions in "real time," thus reducing your risk. How much would this save you?

These are some of the quality management best practices, each of which contributes to risk reduction and therefore increased quality and reduced cost. Now, let's consider the overall impact of quality management on the defects density and the cost of fixing them.

The overall business impact of quality management

Quality management best practices center on quality synchronization points across the whole development lifecycle. We have discussed the benefits of continuous testing enabled using service virtualization, and we have briefly noted collaboration between stakeholders and the QA team, as well as automated reporting. Other best practices include: allowing quality professionals to contribute to the team effort from the very beginning of a project; the integration of practitioners doing the testing as part of quality management; and the use of consolidated quality dashboards.

Together, these quality management best practices can benefit the overall business in measurable ways. Using CMMI[3] as a proxy for the maturity of the development process, Figure 5 shows that the overall business impact of quality management is quite compelling.

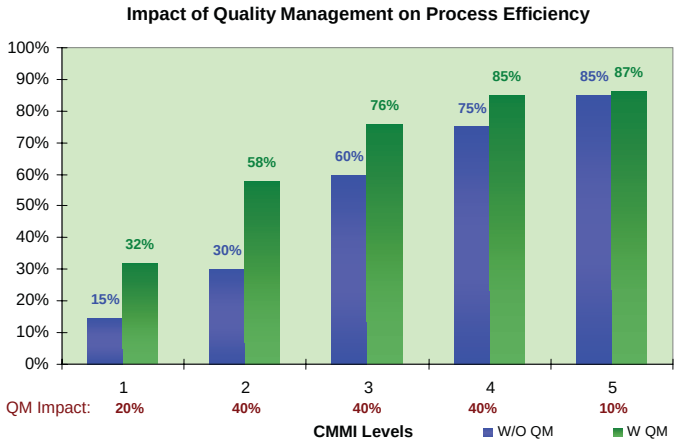

Figure 5. Graphing percentage of defects detected (Y axis) against an organization's software development maturity level (X axis).

Let's assume an organization at CMMI level 2, with 1000 defects detected during functional testing. Figure 5 shows that on average, without QM practices, about 30 percent of the defects are being detected in functional testing (the left, blue bar), and therefore the total number of defects is 3300. However, by applying QM practices, the defect detection rate increases to 58 percent (the right, green bar), therefore detecting 1914 (58 percent of 3300), or 914 more defects.

As fixing defects during User Acceptance Testing (UAT) is about seven times more expensive than during unit/integration test, and assuming a fix cost of US$120 per defect during unit/integration test, fixing 914 defects in UAT is already increasing the cost by over half a million dollars!

And this does not even take into account the reduction in the number of defects that result from applying QM practices in the first place, which makes the savings even more significant. This also does not take into account less tangible savings, such as increased quality, customer retention, and other implications of quality as a differentiating asset.

As most software development teams are around CMMI levels 2 or 3, the benefits and the savings described above apply to most of the industry. But as development teams become more mature, apply QM practices, and move up to CMMI levels 4 and 5, the focus shifts into less obvious—but for some, even more important—benefits, such as reduction in the number of defects that are introduced in the first place, measured improvements around planning and execution of quality related activities, customer retention, and leveraging quality as a differentiating asset.

Better quality + lower cost = improved competitiveness

In this paper, we have described several improvements to methods used by software teams in the design, testing, and deployment of software for systems or IT. Teams may use these quality management and continuous testing methods to deploy that software more quickly, while mitigating quality-related risks throughout the lifecycle. The multidisciplined practice of quality management is breaking the functional and organizational silos that are so common in today's companies. It encourages an analytical process that's closely integrated with the development lifecycle.

Analyzing the market and best practices shows that business outcomes can be optimized, and that smart improvements within the realm of proven best practices, continuous testing, collaborative test planning and automated reporting—a combination of disciplines that defines quality management—can help address the need for increased innovation with more competitive products and services to your customers.

The IBM® Rational® organization is ready to demonstrate these techniques to you. With straightforward adjustments to your investments, deployment practices, and tooling, we can help you realize these benefits within a time frame that best suits your business's needs.

We look forward to working with you!

For more information

To learn more about the IBM Rational quality management offerings, please contact your IBM marketing representative or IBM Business Partner, or visit the following website:

- **ibm.com**/software/rational/offerings/quality/ or
- **ibm.com**/software/rational/servicevirtualization/

Additionally, IBM Global Financing can help you acquire the software capabilities that your business needs in the most cost-effective and strategic way possible. We'll partner with credit-qualified clients to customize a financing solution to suit your business and development goals, enable effective cash management, and improve your total cost of ownership. Fund your critical IT investment and propel your business forward with IBM Global Financing. For more information, visit: **ibm.com**/financing

About the authors

Moshe Cohen is the Market Manager for IBM Rational quality management offerings. In his current role, he works closely with customers, including managers and practitioners, to drive IBM Rational quality management offerings in both the IT and embedded systems spaces. Prior to this, he was with Telelogic, defining and driving its Model Driven Testing solutions. He has extensive hands-on experience in the specification, development, and testing of C3I medical and telecom applications, including technology adoptions and driving process improvement programs. He received his EE and M.Sc in mathematics and computer sciences, both with honors, from Beer-Sheva University in Israel.

Allan Wagner is a Technical Marketing Manager and Evangelist with IBM Rational, driving thought leadership, strategic initiatives, and tangible solutions around the adoption of Rational ALM for IT and manufactured products. Focusing specifically on quality management and testing during his ten years of practical field experience, Al has assisted, mentored, and enabled both internal IBM and external customer teams to address their IT application infrastructure, development, implementation, and operations challenges. Al is a frequent conference speaker on topics of software quality products, principles, and techniques and has authored numerous papers.

© Copyright IBM Corporation 2013

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
August 2013

IBM, the IBM logo, ibm.com, and Rational are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at **ibm.com**/legal/copytrade.shtml

This document is current as of the initial date of publication and may be changed by IBM at any time. Not all offerings are available in every country in which IBM operates.

THE INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING WITHOUT ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY OR CONDITION OF NON-INFRINGEMENT. IBM products are warranted according to the terms and conditions of the agreements under which they are provided.

[1] See the Carnegie Mellon Software Engineering Institute's CEO and founder's message at http://www.sei.cmu.edu/about/message/

[2] For a more complete discussion of quality management practices, download the paper "Value-driven quality management for complex systems: Six strategies for reducing cost and risk" at http://www14.software.ibm.com/webapp/iwm/web/signup.do?source=swg-rtl-spsm-wp&S_PKG=wp_RQM_VALUEDRVN_071510

[3] Capability Maturity Model Integration. CMMI is a staged approach to process improvement that defines incremental levels of maturity in software engineering organizations. For more information see the Software Engineering Institute (Carnegie Melon University) website at http://www.sei.cmu.edu/cmmi/

Please Recycle

RAW14273-USEN-02