How collaboration in software delivery improves productivity for small to medium businesses

Business is never without risk. That's why success can be thrilling, along with its financial rewards. Many risks are obvious, such as the depth and breadth of the competition, or the dangers in missing deadlines or profit projections. But what about less obvious forms of risk, or those considered so unlikely that they go ignored by decision makers? As we've seen in the recent gulf oil spill, an otherwise successful multinational corporation ignores potential risks at the peril of its workers, the environment, and its public image. Or, as in the case of Toyota in 2010, ignoring safety warnings can lead to a very public crisis of confidence, including law suits, lost revenue, and yet another damaged reputation.[1]

In the area of software development and delivery, the rapid pace of change—owing to constant shifts in technology, project requirements, interim company objectives, personnel, and other factors—creates a uniquely risky environment for project managers. What's *soft* in software is its insubstantial nature, that it's merely information, a set of instructions to manipulate machinery that will essentially do what the software tells it to do. When the environment for creating these instructions is under constant change, outcomes become less predictable.

We see three categories of risk in software development. First, there is the risk that a software project will completely fail. That is, the project will not be completed because of technology or management hurdles, or most often a combination of these. Second, there is the risk that the cost of a software project will exceed its benefit to the business. Third, there is the risk of unintended behavior—as in the recent case of Toyota, where the risk posed by a software glitch was not only ignored by the manufacturer, it was simply missed by the National Highway Traffic Safety Administration, the regulatory body that oversees the auto industry. As reported in *The Washington Post*, the agency "was woefully unprepared to decide whether engine electronics might be at fault," and that "NHTSA officials told investigators that the agency doesn't employ any electrical engineers or software engineers."[2]

The business of software is risky not only for the auto industry, but also for the medical, electronics, financial services, communications, aero-defense, and a host of other industries. As systems and devices of all sorts—from jet fighters to toothbrushes to Wall Street trading systems—become smarter and more interconnected, software development and delivery has emerged as a critical business process that needs to be managed as effectively as possible. That includes the ability to address and manage the risks assumed through innovation and ever-more competitive business models. Yet the nature of software's benefits, especially the competitive edge that success can deliver, often blinds managers to the risks that lie just beneath the surface of whiz-bang graphics and user interfaces.

One long-held approach to reducing risk in software development and delivery involves the use of automated tools: specialized software that helps developers manage most aspects of the project life cycle. Just as chainsaws replaced axes and allowed teams of forest workers to be more productive, tools for software development have progressed from language compilers

to those that support discrete stages and specialties within the software development process. Tools for capturing and managing requirements, changes, testing, and deployment, for example, have been used for more than 15 years to support team specialties and roles, and their use has certainly improved productivity. But improvements have typically been measurable only at the individual level, while improvements regarding team collaboration have been minimal.

Collaboration is our next challenge in managing the risks inherent in the intangible nature of software design and development. This will require both better process platforms and better architectures that explicitly confront these uncertainties in open and measurable ways. Recent models for software and systems architecture—including systems of systems, service-oriented architecture (SOA), and reusable middleware—have begun to improve the structure of what gets delivered. Modern governance, CMMI, agile methods, cloud computing, and improved toolsets are providing a better platform for the development process itself. But there is much still needed in both contexts of architecture and platforms.

This paper describes the economic benefits of simplifying software tool implementation for software development and delivery teams through solutions such as Presto from Black Diamond Software. Our goal is to illustrate incremental improvement to a team's risk and productivity profiles through a range of scenarios taken from actual business cases. The software economics discussion is based on Walker Royce's work for the Rational® organization at IBM; the examples provided for each scenario come from Randy Howie's work with customers and the Black Diamond team.

The nature of software projects

The best thing about software is that it is soft (i.e., relatively easy to change) but this is also its riskiest attribute. Should we think of software as something designed and manufactured like a bridge that is engineered and then produced according to an organized plan with a division of labor? Or is software more like a movie, a collaboration involving a team of craftsmen and emerging from the naturally creative process of artistic people?

Unlike bridge construction, most software does not deal with natural phenomena where laws of physics or materials provide a well-understood framework. Hence, like a movie, most software is constrained only by human imagination; the quality of software is judged more like a beauty contest than by precise mathematics and physical tolerances. Also like a movie, you can seemingly change almost anything at any time: plans, people, funding, requirements, designs, and tests. "Requirements" rarely describe anything that is truly required. Nearly everything is negotiable, and quality is best determined through the eyes of the audience.[3]

Unlike a movie, however, the consumer cannot usually see all of the capabilities of a given piece of software in order to judge it. In making online purchases, for example, consumers can see the list of products in their shopping basket but not their personal data, which has been stored. Is that data secure? Often it is not—data security cannot be visualized and is difficult to ensure. These "behind the scenes" behaviors are part of the reason that software is so inherently risky.

If we don't carefully manage software production, we can lull ourselves into believing that what we see is what we get and ignore the risk lurking beneath. We can end up operating in malignant cycles of change that result in massive amounts of scrap, rework, and wasted resources. And worse, we can smother the business results that are the purpose for the software in the first place.

Software project managers must combine techniques that support the creative process with those used in conventional engineering projects. The process must be agile enough to support change and other aspects of complex teamwork via a set of tools that enable monitoring and measurement of that process.[4]

The question is, what techniques can we use to manage what is essentially an intellectual endeavor?

Agile development and the challenges of software delivery

While a variety of industries are finding software delivery to be challenging, it has nevertheless become an essential part of their business operations. The needs of large-scale businesses have spawned niche markets for small to medium size suppliers, where turnaround time is brief and the margin for error is slim. Companies at this level must quickly meet their software challenges in order to deliver quality components to their major customers.

Over the past decade, many smaller companies competing in niche markets, or smaller IT organizations enabling their businesses through software, have adopted agile software delivery techniques as a way to meet deadlines more efficiently. Much has been written on agile development,[5] and a full discussion of the concepts and benefits of agile is beyond the purpose of this paper. Suffice it to say that agile techniques have allowed software project teams to move from a "development activity" focus to a "delivery of results" focus, which is the whole point of the process.

Alongside this evolution toward a more agile software delivery process is the complementary market for software delivery tools, noted earlier. Tools for testing, requirements management, change management, security and compliance, assembly and delivery, etc. allow software teams to be more productive and thus more valuable to their respective business, since each class of tools helps automate a critical portion of the software project life cycle.

But the specialization of software development roles and the interrelated data these roles must create and maintain have introduced new areas of risk into the equation. Let's consider the risk associated with specialization across the life cycle.

Life cycle challenges

The next advance in software tools will be improved collaboration across these roles throughout the project life cycle. To date, most traditional tools offer little to aid collaboration as teams work to deliver results. The hand-off from one phase—one tool, so to speak—to the next is managed via manual processes that are error prone and time consuming. Consider the impact of this manual process in just one area of the development life cycle: change management. Some form of change management practice is vital for coping with dependencies among hundreds, or thousands, of units of software code, each of which must be created or reused, tested in context, and retested whenever a change is introduced that "touches" any given unit.

Figure 1 shows the potential impact of change on a very small part of a typical project in the mid-market.

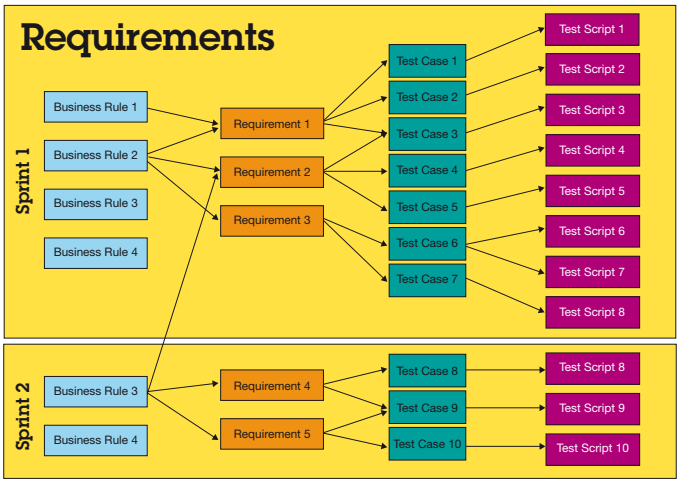

Figure 1: Requirements begin with "business rules" (shown in light blue boxes at far left). These become actual "requirements" (deep yellow), leading to "test cases" (teal) with corresponding "test scripts" (purple) at far right. A change in business rules will have a cascading, multiplying effect on requirements and testing that can overwhelm teams relying on non-automated processes for life cycle collaboration

Figure 1 illustrates how enormous the impact of change can be. A small, benign change in one component may cause malignant changes across numerous components. And because many different people, each playing multiple roles, must react to a change exactly when it happens, it's easy to see how change causes risk on software projects. If the combined explosion of change and communication to team members is not managed, risk "leaks" are created, the effects of which permeate many components and can linger throughout the life of the software.

Challenges for smaller teams

The difficulties associated with change management plague software development and delivery teams of all sizes. But there is a major difference between large and small software organizations when it comes to their risk profiles. While large software organizations can "amortize" major investments in tools and training over a period of several years, and absorb early failures as nominal growing pains over the course of successive projects, smaller companies do not have this luxury. Automated tools and modern development methods are equally critical for these smaller, niche players, but their investments need to be realized in the shortest of time-frames—typically by the end of the current project. Failure is often not an option: the current project may be their last.

A word on open source tools: Free at what cost?

Tool automation can improve the productivity of individuals, and, certainly, open source tools (i.e., "free" software) can provide much of this automation. However, open source tools do not minimize the impact of change described in Figure 1. That's because they do not integrate easily, do not improve team collaboration, and therefore do not reduce the primary project risks associated with modern development practices. True, there are open source tools designed to help some teammates collaborate on certain tasks. But these do not provide team-wide collaboration across the application development life cycle. In fact, open source tools can create a barrier to future collaboration improvements.

A grounding in software economics

The agile methods and tools described above emerged as a significant, though partial, answer to the unique challenges of software "engineering." For small to medium size companies competing in niche markets, or for smaller IT organizations enabling their businesses through software, agile practices help team members be more productive, and in the aggregate, team productivity should rise as tools automate each specialized area of the life cycle.

But we can do much more.

The underlying goal is improved *productivity*. Productivity can be measured in time savings and translated clearly into monetary value. But it's just as clear how these savings can be eroded during a project if risks are not properly managed. For example, a project can appear to be on track, or even ahead of schedule; but a missed requirement discovered at some downstream point in the project can mean revisiting and, at worst, recreating units of code that had been settled earlier. Furthermore, the most valuable forms of productivity are based on team collaboration, which extends beyond improvements made by individual team members. Support for effective, efficient collaboration must be the cornerstone for productive teams, yet collaboration—which requires proper tooling, processes, and team configuration—must itself be managed like any other risk element within a project.

Improved productivity at both the individual and team level is possible when processes are appropriately scaled. As we'll illustrate below, a full range of improvements across an organization can be achieved over time, as teams collaborate and the business is willing to scale efforts toward greater business results. But the kernel of improved *software economics* lies in the first steps to combined individual and team productivity.

What do we mean by "software economics?" Good economics means efficient management of finite resources toward an optimal result. At the individual and team levels of an organization, improved productivity is an immediate and tangible aspect of good software economics. Improved productivity means more or better output per dollar (or euro, pound, yen, rupee, or yuan). This improvement can be channeled into business value in many dimensions, such as cost savings, better quality, earlier time to market, more predictable delivery, increased market share, enhanced market reputation, and even greener operations.

But for most organizations, this improvement is more aspiration than reality. Each software delivery practitioner today costs American companies about $200,000 per year, including salary, benefits, capital, overhead, profit, and other direct costs. Even a 5% productivity improvement is therefore worth $10,000 per year per person. That is substantial financial leverage. In lower cost countries, the numbers change, but the leverage is still significant.

Let's quantify this further. The IBM® Rational organization and its business partners have observed a scale of organizational improvement that spans individual productivity through the use of effective software tools, through team and process improvements, all the way to increased business value at the company level. For this discussion, we will focus on improvements that are possible through an optimized tool adoption strategy, as mapped out in Figure 2.

The key message from Figure 2 is that a range of incremental, measurable improvements can be achieved. As a company's business model matures, so should its software development and delivery process. The more significant improvements, like systematic reduction in complexity and major process transformations, require more significant financial and time

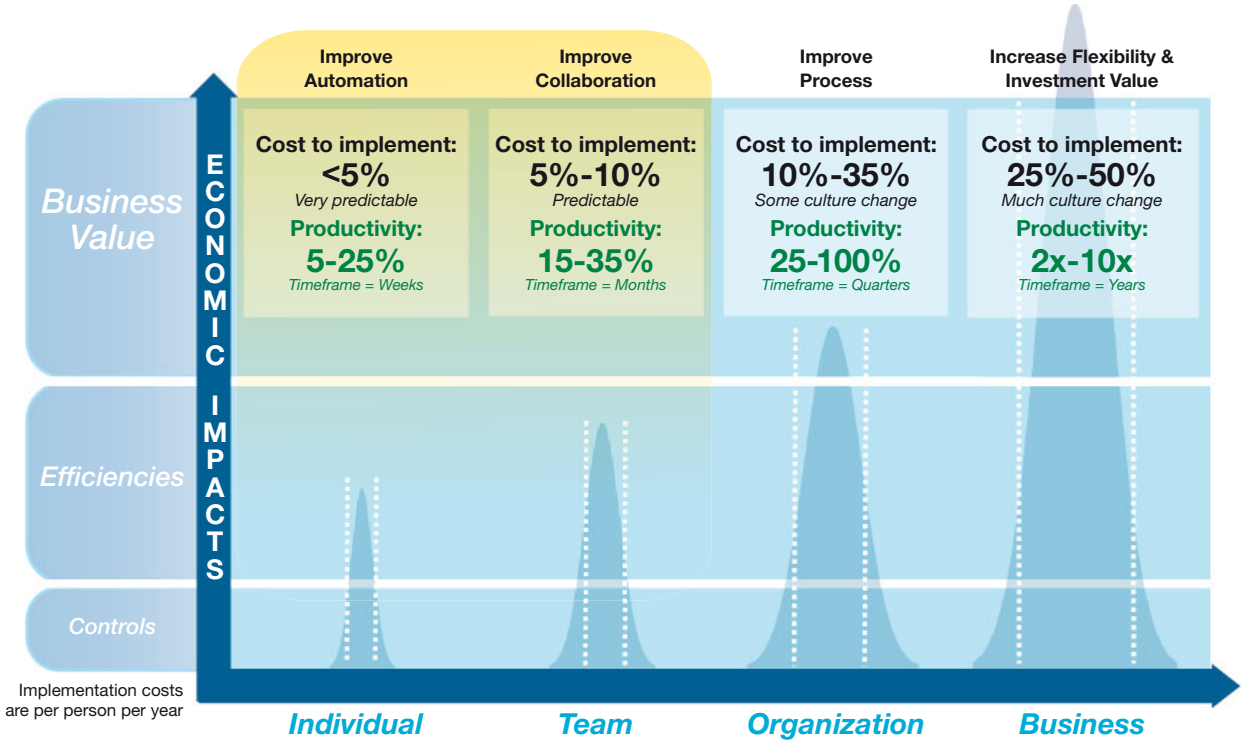

Figure 2: Expected productivity returns from investment in application life cycle management technology

investments and the range of outcomes has more uncertainty. These are covered by the business value improvements graphed at the right of the scale in Figure 2, and they tend to be broader organizational initiatives that require changes to the "culture" of the organization. The smaller improvements highlighted in yellow are more predictable and straightforward to deploy. These include skill improvements, and automation improvements targeted at individual teams, projects, or smaller organizations.

The remainder of our discussion will focus on an optimum approach to changes at this end of the scale, where teams gain productivity through the transition from individual tools to Collaborative Application Life cycle Management (CALM) tools. The question is, how can tools be implemented so that economic improvements can be realized and measured?

The move toward collaboration

Reducing risk and improving productivity are the fundamental reasons tools are used by software development and delivery teams. Improved productivity is achieved by improving the output of the people building software. Risk is reduced by clearly describing and focusing on the desired business result and minimizing the impact of change upon the result.

Over the past twenty years, the software industry has offered application life cycle management (ALM) tools to the marketplace that promise automation for individuals and the tasks they perform, but they do not provide the cross-team support known as "collaborative ALM," or CALM. In some cases, a suite of tools was offered that could only support CALM with significant investment in integration. Yet the marketplace rarely used the Rational suite in its entirety, nor did they widely subscribe to the services necessary to create CALM. Further, only larger businesses that could amortize the investment could afford the cost of integrating the suite to achieve CALM. Businesses could buy tools that automated tasks within their ALM implementations, therefore ending up with a collection of ALM tools that automated areas of the ALM process but were not themselves truly integrated.

The IBM Rational organization made this selection process easier by acquiring and developing a suite of tools that partially paved the path to full integration. While these tool choices resulted in productivity improvements, substantial risk reduction did not come out of the box and still required clients to integrate tools and processes to adapt them to their specific context.

Today's Collaborative ALM, or CALM, is more squarely targeted at risk reduction by eliminating the errors that can occur through poor integration or through manual transfer of code artifacts under development. In order to reduce risk the following must occur:

1. The tools that provide automation along the application life cycle must work on a common data architecture.

2. Dependencies between ALM tasks must be managed with traceability between data elements.

3. Whenever a change occurs, the right people along the trace must be notified of the change immediately.

In the next section, we will present a range of case-based scenarios that demonstrate incremental improvements toward an optimum, productive CALM tool environment.

Tool implementation: A range of scenarios

To this point, we have provided a perspective on agility and tooling as they relate to the fundamental parameters of software economics, risk, and productivity. Now let's examine the impact of tools choices on a real project through a range of scenarios.

We have chosen an example business and project taken from one of Black Diamond's clients. This successful 90 million dollar business had reliable but stable net income of about five percent and wanted to grow by venturing into a new market. The vision for net income growth relied on a web-based product, for which their business case showed a net income growth of 20 percent if successful. Black Diamond was asked to help this company arrive at a budget. A team of nine people was needed to build the first release, including contractors from Black Diamond. The average loaded hourly cost of the team was $70 per hour and based on initial but gross estimates a project budget of $1.134 million was established.

Project Staffing Cost Analysis

Description	Value	
Average fully loaded hourly cost	$	70
Cost/week	$	25,200
Cost/month	$	100,800
Cost/sprint	$	75,600
Cost/release	$	378,000
Cost/project	$	1,134,000
10% of project cost	$	113,400
Cost per team member per month	$	11,200
10% of team member per project	$	1,120

Business A

Description	Value	
Revenue	$	89,418,750
Net income	$	4,773,750
Target net income growth		20%
Target net income	$	954,750
Minimum weeks to recover cost		62
Minimum weeks to recover cost + 40%		86
Cost as a percentage of net income		23.8%
Cost + 40% as a percentage of net income		33.3%

- Project represents a significant cost to the company
- Over a year of at expected increased income to pay for the cost of the project
- At the average cost overrun for a project of 40 percent, project consumes one-third of net income and will take almost two years to pay back

Table 1: Project parameters for Business A

Given the initial budget it would take over a year of the target net income increase to pay for the project. The profile for this company, Business A, and its project are outlined in Table 1 above.

With the business results as a basis, let us examine the choices Business A faced as it decided how to outfit its team with development tools—the goal being to increase team productivity and save costs. We will quantify the results in terms of the company's balance sheet, which, in some cases, will make it easy to see that the introduction of tools can defeat the business goal. Each scenario includes an estimated risk for cost overrun.

Description	Best Case	Worst Case
Estimated Staff Cost	$ 1,134,000	$ 1,134,000
Risk of Project OverRun (40%)		$ 453,600
Totals:	$ 1,134,000	$ 1,587,600

Table 2: Using no tools to manage risk of project cost overrun.

Scenario 1. Do nothing—using no ALM tools at all

The risk we've discussed in this paper is widely referenced by the Standish Group's series of CHAOS reports[6] which as early as 1994 found that 25 percent of software projects fail to complete and those that complete do so with an average 189 percent budget overrun. In the past 15 years the report has been widely analyzed, praised, referenced, and disputed. While it is not in the scope of this paper to appraise the Standish Group's findings, our own empirical experience over the past twenty years has taught us that software project estimation is imperfect and sometimes extremely difficult. This, combined with the inherent risk we have discussed, means that most projects end up over original estimates. Therefore, any business relying on a software project to boost results would be negligent not to plan for an overrun. We believe a reasonable assumption is a 40 percent overrun, which according to some estimates is a conservative number.[7] Therefore if a business attempts to execute a software project with no ALM tools they have no opportunity to achieve any of the productivity improvements illustrated in Figure 1 and face a planning risk of 40 percent cost overrun. Table 2 shows the best and worst case outcomes regarding the "Do Nothing" scenario.

Scenario 2. Improving individual productivity

The majority of businesses, like our example business, invest in automation at an individual level, as shown in Figure 2. These tools help with version control or change management, or requirements management, but they do not represent a full-suite of integrated CALM tools.

Today a major category of non-integrated ALM tools are those available freely as downloadable open source projects. These tools are attractive, since they are free and available via the Internet. Let's examine the impact on our example project if the team chooses open source.

First, as described in the CALM discussion earlier, the team can expect individual productivity improvements. The best and worse case impacts are shown in Table 3. But as we'll see below, these benefits do not include substantial risk reduction.

The missing man condition

Once tools are selected and purchased they must be installed and configured for the team to use them. Further, if the tools are not collaborative, integration may be necessary to approach some level of collaboration. Large companies often hire or have in-house separate teams dedicated to this purpose so as not to interrupt the progress of software development. However, it is obvious that our example company cannot afford such a strain on personnel resources, which would significantly compromise the business case. Organizations who do this end up using members of the software development team to install and configure the software. This leads to the "missing man" condition. Table 3 accounts for the costs of missing resources by making conservative best and worst case assumptions about the time commitments of the missing team members.

Hiring a contractor to do the installation and configuration is seldom a solution, since that expense typically raises the cost of installation and configuration even higher than the missing man condition. Further, using team members who are not fully trained or experienced in tool assessment can lead to compounded risks, such as improper installation or inadequate knowledge transfer.

The time to learn

To be clear, open source or other non-integrated tools can help *individual* team members improve productivity. In this case, the risk of a project overrun improves by five percent over Scenario 1. While this is a step in the right direction, tools designed to automate specialties within the application life cycle, including open source tools, are not provided in a way that aids team collaboration.

Description	Best Case	Worst Case
Estimated Staff Cost	$ 1,134,000	$ 1,134,000
Risk of Project OverRun (35%)		$ 396,900
Cost of Open Source Tools	$ –	$ 20,000
Increased Productivity (20% - 5%)	$ (226,800)	$ (56,700)
Architect Time Lost on Tools (1 wk - 3 wks)	$ 2,800	$ 8,400
Developer Time Lost on Tools (1 d/wk - 2 d/wk)	$ 25,200	$ 50,400
Decrease in Team Velocity (-2%)	$ –	$ 22,680
Totals:	$ 935,200	$ 1,575,680
Improvement:	17.5%	0.8%

Table 3: Implementing open source ALM tools.

The new element introduced with the addition of tools is the time required to learn, implement, and train team members regarding the new technology. The "missing man condition" causes one or more team members to be "lost" to focus on the introduction of tools, which causes some economic impact on the project.

Of course, the Internet continues to provide a wide assortment of training material for novice tool users. Video, downloadable demos, and Q&A wiki websites are widely available as an alternative to traditional class based training, which appeals to smaller companies on tight budgets and deadlines.

The best solution is for teams to move to standard, collaborative platforms, where the cost of learning can be reduced over time. But, as noted earlier, small businesses most often do not have time on their side.

Scenario 3. Using Jazz-based tooling

With business rules traced all the way to test cases and test scripts shown earlier in Figure 1, dependencies become overwhelming for development teams who use no tools to automate processes. These dependencies also present significant, time-consuming challenges for teams using tools that facilitate work only at the individual practitioner level, because non-integrated tooling necessitates manual handoff of artifacts between development roles during the project life cycle. When one team member makes a change, every artifact in the chain must be tested, again, to ensure consistency.

What does this mean in terms of agile development practices? Think about the need to manage requirements throughout a project's life cycle. At the most obvious level, this means ensuring that a customer's expectations are met. But it also means ensuring that any changes made to project artifacts (as a result of various course corrections, or, frequently, changes made by the customer mid-stream) do not adversely impact dependent artifacts as a result. The coding team needs to notify the requirements management team of all changes made, and the RM staff need to carefully rectify those changes against previously established baselines. And this sort of "manual handoff" must occur between other teams as well, including the testing team (who, in this case, must invoke regression testing methods to ensure overall stability) and the analyst and architect, who must ensure system-wide integrity through separate software models or related artifacts, all of which must be manually updated.

Several years ago, the IBM Rational organization launched the Jazz™ initiative to improve the flow of software artifacts from one tool to the next during the project life cycle. Just as good jazz musicians seamlessly elaborate, improvise, and collaborate during a live performance, practitioners using the IBM Rational Jazz platform are able to focus on their specific contributions to the life cycle without spending excessive time in the inter-phase handoff to the next set of practitioners. As described on the Jazz.net website, "Jazz products embody an innovative approach to integration based on open, flexible services and Internet architecture. Unlike the monolithic, closed products of the past, Jazz is an open platform designed to support any industry participant who wants to improve the software life cycle and break down walls between tools."[8]

Description	Best Case	Worst Case
Estimated Staff Cost	$ 1,134,000	$1,134,000
Risk of Project OverRun (20%)		$ 226,800
Cost of Jazz Tools	$ 68,000	$ 85,000
Increased Productivity (35% – 10%)	$ (396,900)	$ (113,400)
Architect Time Lost on Tools (1 wk – 3 wks)	$ 2,800	$1,134,000
Developer Time Lost on Tools (1 d/wk – 2 d/wk)	$ 25,200	$ 50,400
Decrease in Team Velocity (-2%)	$ –	$ 22,680
Totals:	$ 833,100	$1,413,880
Improvement:	26.5%	10.9%

Table 4: Implementing Jazz-based tools from IBM Rational.

The key new ingredient here is greatly improved *collaboration*. In the context of software development projects, collaboration not only allows smooth transition from one phase of the iterative life cycle to the next; it also allows the generation of documents, process measurements, and other aspects of modern, agile project management to occur without overburdening team members or slowing the process. In the requirements management example described earlier a Jazz-based project features automatic notification of a change to the requirements management system, which in turn notifies testing and analytic software tools of changes to be examined and/or baseline inconsistencies to be rectified. Such a collaborative environment does not eliminate the human element at work, but rather makes the work easier to perform since the manual, and error-prone, data entry task is eliminated.

As shown in Table 4, the implementation of Jazz tooling offers a significant improvement over both the non-tool and the non-collaborative tool environment.

Yet, for all its collaborative value, the Jazz platform still requires training and implementation time, which for small to medium size software delivery organizations can mean the difference between making or missing a critical deadline. Our fourth and final scenario offers an answer.

Scenario 4. Using Jazz-based tools in combination with Presto

With a team poised to take advantage of a fully collaborative development environment, including a full set of tools to support all team members, software development organizations are well on the way to eliminating risk. Remember that the risk to a software project is ultimately a risk to the business-especially small to mid-sized businesses whose projects frequently represent a sizable opportunity in the life and health of the company.

While tools that support individuals, collaboration, and project success are typically touted as “agile” in today’s software development arena, tools need to support business results and

not jeopardize them through the time involved with their introduction. Furthermore, tools need to keep a project on track and not overrun the project budget because they're too costly.

In these scenarios, we have attempted to quantify risk in terms of cost within the context of a specific software development project. The greater the cost related to staffing, tool procurement, project overrun, and other budgeted items, the greater the risk to the business. Therefore, the next improvement in the use of CALM methods through Jazz-based tooling involves the elimination of risk in implementing and configuring Jazz tools themselves. How do we address this risk? A key strength of Jazz is that it's an open, extensible platform, which allows a growing "ecosystem" of partners to contribute point solutions for specific industry needs or other needs not addressed by current Jazz offerings. IBM partner Black Diamond Software now offers a Jazz configuration environment, called "Presto," which is designed to speed and simplify the process of introducing Jazz technology.

Presto reduces risk and increases productivity in the following ways:

1. Missing-man condition is so diminished that we can eliminate it from the economic analysis shown in Table 4.

2. The 100+ steps necessary to properly implement and configure the Jazz tools are now automated.

3. Many undocumented nuances of the installation and configuration process are addressed and simplified.

4. The possibility that an incorrect installation will impede the benefits of Jazz collaboration has been eliminated.

5. Automatic backup and security for the collaboration data is included.

6. Presto appliances can be easily updated to quickly incorporate updates to, and new releases of, Jazz products.

Description	**Best Case**	**Worst Case**
Estimated Staff Cost	$ 1,134,000	$ 1,134,000
Risk of Project OverRun (20%)		$ 226,800
Cost of Presto	$ 35,000	$ 35,000
Increased Productivity (35% – 10%)	$ (396,900)	$ (113,400)
Totals:	$ 772,100	$ 1,282,400
Improvement:	31.9%	19.2%

Table 5: Implementing Jazz-based tools along with Presto.

Essentially, Presto delivers Jazz tools already implemented and configured in a virtual appliance. A virtual appliance is a complete, functioning virtual computer—including the operating system, devices, memory, and configuration needed to successfully use Jazz tools and quickly begin reaping the economic benefits of CALM. Because Presto greatly diminishes the effects of "the missing man condition" for Jazz tools, it can shave weeks off a software project team's learning curve.

As described earlier under "the missing man condition," tools require time and effort to be incorporated into the life cycle of the project team. And the cost here can be significant—anywhere from days on a small single project to months on a corporate wide initiative. This number increases (non-linearly) as you increase the number of tools and need to work out the integrations.

Designed to enhance the economic benefits of Jazz, IBM's most advanced software development technology, Presto actually *includes* the Jazz tooling as part of the package. This offers significant economic benefits, as shown in Table 5.

Outfitting each team member with Presto is as simple as starting the client appliance and performing some simple configuration steps. A repository server manages project artifacts and allows team members to collaborate as necessary to support the development process. Using Presto, team members are ready to play their respective roles in the application life cycle and help ensure that the team as a whole can deliver software to the expected business results. On average you can save up to 70 percent with Presto compared to purchasing standalone tools.

Conclusion

Software is developed and delivered under conditions that are more common to creative endeavors than to traditional engineering; the possibilities are nearly limitless and change happens rapidly. Yet risk lurks beneath the surface of software projects because of enormous complexities and the difficulty of visualizing software capabilities. Automated tooling for managing and measuring software projects is the answer, but you need to incorporate tools into the team's work environment with the lowest possible cost and risk to avoid compromising the business results of software projects.

The key is the understanding the relationship between a) time and money, and b) testing and requirements dependencies. Once requirements become too numerous to track manually (and this frequently occurs in projects of relatively small scope), it becomes easier for project managers and decision makers to see the benefits of an automated toolset, such as IBM Rational Jazz, that supports cross-team collaboration. A fully collaborative development platform such as Jazz allows progression from individual productivity to team productivity.

Extending the core Jazz environment, new technologies like cloud computing, software as a service, and virtualization solutions such as Presto by Black Diamond Software offer a significant move forward in dealing with software project risk, while providing even higher levels of productivity.

About the authors

Randy Howie is CEO of Black Diamond Software, an IBM premier business partner specializing in software application development, installation, productivity tools, and processes. With nearly twenty-five years in the areas of engineering management, project management, and advanced technology, Randy joined Black Diamond Software from Artificial Intelligence Technologies, where he was Vice President of engineering. Randy is also co-author of "The Five Mandates of Software Development," and he holds a B.S. in Mechanical Engineering and a M.S.E. in Engineering, both from Carnegie Mellon University.

Mike Perrow is a writer and editor on the brand strategy team for IBM Rational. He is the founding editor of *The Rational Edge* online magazine, which covered processes, IT and business alignment, and related technologies from Rational Software Inc., and later as part of IBM. He holds degrees in English and writing from the College of William and Mary and from the University of Massachusetts, Amherst.

For more information

To learn more about acquiring and configuring an IBM Rational Jazz solution for your business via Presto please contact your IBM marketing representative or IBM Business Partner, or visit:

- The Presto site: www.bds.com/presto
- Download the White Paper: "The Presto Manifesto—Business results over software process, cost control over craftsmanship, team success over individual achievement"
- Contact Black Diamond at (203) 431-9600 or by email: sales@bds.com

Additionally, financing solutions from IBM Global Financing can enable effective cash management, protection from technology obsolescence, improved total cost of ownership and return on investment. Also, our Global Asset Recovery Services help address environmental concerns with new, more energy-efficient solutions. For more information on IBM Global Financing, visit: **ibm.com**/financing

[6] For access to the complete series of CHOAS reports by the Standish Group, see http://www.standishgroup.com/. Or you can find excerpts from these reports at a variety of academic computing-related web sites, such as http://www.projectsmart.co.uk/docs/chaos-report.pdf

[7] The 40% estimate for cost overrun as a result of using no tools at all is based on a combination of our own experience and a variety of anecdotes from our partners over the years. For further validation of this baseline statistic for "doing nothing," see Greg Alleman's blog at http://herdingcats.typepad.com/my_weblog/, and especially his entry from 2007 at http://herdingcats.typepad.com/my_weblog/2007/06/project-failure.html

[8] http://jazz.net/about/about-jazz-vision.jsp

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
December 2010

[1] This movie analogy is borrowed from Walker Royce, from his IBM white paper "Improving software economics: Top 10 principles of achieving agility at scale," at http://www.ibm.com/common/ssi/fcgi-bin/ssialias?infotype=SA&subtype=WH&appname=SWGE_RA_RA_USEN&htmlfid=RAW14148USEN&attachment=RAW14148USEN.PDF

[2] http://www.latimes.com/news/local/la-fi-toyota-recall8-2009nov08,0,6120294.story

[3] http://www.washingtonpost.com/wp-dyn/content/article/2010/02/22/AR2010022204887_2.html?sid=ST2010022204995

[4] Royce, Walker, "Successful Software Management Style: Steering and Balance," IEEE Software, Vol. 22, No. 5, September/October 2005

[5] You can begin your exploration of agile development principles with the Agile Manifesto, for example, at http://agilemanifesto.org/