Software development through collaboration
White paper
October 2008

Rational software

In tune with IBM Jazz and IBM Rational Team Concert enterprise development tools.

Jean-Yves Rigolet, software engineer, Rational software, IBM Software Group

Contents

Discordant software development

As enterprises adapt to evolving marketplaces, the evolution of business applications has had an ancillary effect of giving rise to disconnected or loosely linked teams. Teams are often organized around run times, roles and technology, data, or architecture, but not around the people who make up the team. The development of business applications has never been easy, but projects shouldn't fail because of a lack of visible progress or a vast array of technology and development styles that are too expansive to contain.

IBM started the IBM Jazz™ project to merge evolving business requirements with the latest technological advances in social networking. In 2006, a group of experienced IBM i and IBM System z® tool developers formed the Enterprise Advanced Technology work group. The group's goal was to explore ways in which Jazz technology could add value to IBM enterprise tools and provide a framework for organizing teams around people, artifacts and processes.

The crescendo of Jazz

Business application development began on the mainframe. Developers built applications while sharing a common system where they could directly edit, compile and deploy code. With the common system as the center for business application development, project development workflows became the first collaborative development environment.

Soon, new sets of development tools based on new languages emerged to help create PC applications. While developers' productivity increased, so did system complexity, since developers had to import and export artifacts between tools, hindering teamwork and collaboration.

Highlights

The multilanguage IDE ushered in new collaboration tools and changed how developers work today.

In early 2000, a new form of integrated development environment (IDE) changed all that: the multilanguage IDE. An extensible platform for building development tools, Eclipse led the multilanguage IDE pack as an open source project available to the entire development community. The broadband boom helped Eclipse gain popularity through the Internet and augmented two-way collaboration using downloads and uploads. Soon, many new collaborative tools proliferated—including instant messaging, Web conferencing, file sharing, wikis and blogs—making teamwork easier and accelerating the software delivery cycle. These tools also pushed new development trends and changed how the IT industry and developers work today. Development teams that used to work at a single location are now collaborating as smaller teams that are geographically distributed around the globe.

The Enterprise Advanced Technology group has carefully monitored these shifts and analyzed new business orientations. In response, IBM is building an environment that can meet the needs of the individual and lift the constraints on team productivity. With the Jazz project, developers can leverage teamwork and interactive concepts to design and deliver products within a collaborative development environment.

You may not be familiar with mainframe and midrange systems, but you rely on their functions every day.

Composing applications on mainframe and midrange systems

Although many people are unfamiliar with mainframe and midrange systems, we all rely on them every day. Do you know what's happening behind the scenes when you withdraw cash from an ATM, buy gasoline for your vehicle, or await your monthly paycheck? These transactions are computing without interruption on IBM i and System z servers all over the world, every day.

Highlights

Applications that were originally written to run on first-generation computers are attempting to coexist with the latest technologies.

Over the years, developers added more applications and more power to these systems to accommodate growing business demands. Such additions have resulted in a huge variety of artifacts that were originally written to run on the first computers but today coexist with the latest technologies—such as Java™ Platform, Enterprise Edition (Java EE) applications, asynchronous JavaScript and XML (AJAX) assets and Web 2.0 artifacts. A close examination of artifacts on a given system can show us the entire IT history of a single company.

The lifetime of an application running on an IBM i or System z server can be between 10 and 40 years. Your monthly payroll system may have been built on programs that were written before you were even born. This can present real maintenance challenges in terms of finding the history behind an application and bringing new employees into the workforce.

If we look at the payroll example—the application that may have been coded before you were born—it is likely that the requirements for your locale have evolved since your payroll application was first coded, compiled and deployed by your company. It is also likely that your company experienced structural or IT infrastructural changes. All the changes that occurred over time inevitably had an impact on the application's original source code, which means your company ended up with thousands of interrelated resources that all needed to be managed simultaneously. This endeavor is monumental in itself.

Even when developers are using the same language, they still have to navigate one another's organizational systems.

But to make developers' work even more complex, the applications they have to work with have evolved over time. And even when developers are using the same language, they encounter a plethora of code organizations in which current and former developers have kept their artifacts according to need or design. The variety makes it difficult to find artifacts and maintain them.

Highlights

The development problems that exist in a small-enterprise context also exist in a large-enterprise context, but they're bigger.

Development teams are often organized around run time, roles and technology, data, or architecture.

Another problem developers have to face is a variety of data organizations and file systems. When partitioning large projects into smaller components according to people, geography or development platform, project managers have to coordinate the work with teams and team members.

If all of the problems mentioned above exist in a small development context, they exist in an enterprise development context—but they're bigger.

Teams with different tunes

When it comes to building and running business applications, there is always the challenge of matching the right people to the right work. In some cases there is a division between people with business knowledge and people with technical expertise. For development, further divisions frequently occur around the system structure.

Although companies vary dramatically according to size, culture, history and other criteria, IBM has identified four primary organizational tendencies that affect the way development teams are organized (or disorganized, in some cases).

Organizing around run time

Despite the emergence of new ways to limit friction between run times—like using model-driven architecture, platform-agnostic languages like Java, or business-oriented languages like Enterprise Generation Language (EGL)—application developers continue to be grouped according to the systems they build on. Education and experience naturally link people together, but creating teams based on these commonalities can make it difficult to adapt to emerging technologies.

Highlights

Organizing around roles and technology

Whether the role is development, quality assurance, building and deployment, or a combination of all three, the tools, languages and platforms that have to be mastered can be very different and are often handled by separate teams, sometimes even in different locations.

Organizing around data

In cases where data has evolved over time—like from a traditional System z file system to an advanced relational database—the way to access the data, and even the access routines, is usually handled by another individual or team that is strictly dedicated to data.

Organizing around architecture

Stacks of applications, using different technologies and running on different platforms, need to be constantly verified and maintained. To manage verification and maintenance, teams are often organized around architecture and their modernization approaches.

Harmonizing with Jazz technology

IBM Jazz technology can help enterprises address the challenges of developing on the IBM i or System z platform.

IBM i and System z platforms are extremely powerful and can efficiently and reliably support large and diverse development organizations. Nonetheless, these platforms can present challenges. Jazz technology addresses these challenges from within the tools used by team members involved in enterprise application lifecycles.

The intent of this paper is not to give an in-depth view of Jazz technology itself, but rather to reveal how this technology can help overcome the challenges of enterprise application development. Because successful enterprise application development requires that the people, the artifacts and the processes work together, in concert.

Highlights

Creating cohesion and facilitating improved software development projects, Jazz can help maintain individual developers' productivity while supporting cross-organizational collaboration.

The chords of Jazz

Jazz provides the technology to address many IBM i and System z challenges. By offering an extensible framework that dynamically integrates and synchronizes people, artifacts and processes, Jazz can create cohesion and facilitate improved and accelerated software development projects.

In developing the Jazz platform, the Enterprise Advanced Technology group first investigated the entire enterprise development stack in the IBM Rational® Software Delivery Platform to determine how the new technology could add value. This investigation involved digging inside some of IBM's most important development tools, such as IBM Rational Business Developer, IBM Rational Application Developer, IBM Rational Developer for System z and IBM Rational Developer for IBM System i™ software.

Next, the team considered the developers themselves. A new technology would need to help maintain individual developers' productivity while still supporting their team roles and their collaboration with dispersed team members. It would also have to support the way developers work today but still provide a framework that would allow them to adapt and enhance their unique processes over time.

The IBM group explored developers' current challenges and sought to solve them. To understand some of the challenges, consider again our monthly payroll system example.

Highlights

The simplified system contains three main components: one to deal with updating employee absences, another to calculate actual payroll, and the third to handle legal declarations. Let's say that the first component—the one that updates employee absences—was recently developed using a Java EE Web application. This application runs on distributed servers that access back-end services created in EGL with data stored in an IBM DB2® relational database that employees can access to edit and display their absences. Absences-related data and inputs from other sources are then used by the payroll calculation process that is executed on the System z platform and launched monthly by a job command language (JCL) batch script using COBOL code. At the end of this process, paychecks are printed and some payroll information is collected and provided to a midrange IBM i application in charge of collecting tax declarations.

In our payroll example, we can see how a development system with many moving parts and multiple stakeholders can create a software delivery cacophony.

With that development context in mind, consider what happens in a typical day for the team in charge. After receiving a request from one of the application stakeholders to delete an absence-related code, the project manager identifies the affected parts of the system and the different teams responsible for them. To do so, he or she first uses an impact analysis tool like IBM Rational Asset Analyzer software to discover the potential effects. Then, according to the results, he or she informs the teams in charge of the affected components that those assets have been impacted.

Extra vigilance is required if an organization is going to overcome the lack of standardization and transparency in its software development and delivery system.

This kind of complicated structure for delivering software is vulnerable to many errors, mostly human. On top of that, the coordination of the work being done can be another source of problems. Even assuming there is only one team working on the Web application with a team leader who will distribute the work, the task of deleting the absence-related code from the Web application can still involve many team members. And to make their collaboration even more difficult, team members might use different development tools, different bug tracking systems and different code repositories. This lack of standardization and transparency requires the team leader to be extra vigilant and follow the different aspects of the work carefully.

Highlights

After successfully removing the absence-related code from the Web application, team members must update the rest of the payroll system to align with this new business requirement. There are no direct dependences between these two components, so both operations could be done concurrently, but does anyone on the team know about the changes? Again, lack of transparency means knowledge of changes is uncertain.

Without an efficient tracking system, individual developers can continue to make changes to an application without anyone knowing.

Changes can continue in the same erratic way for a long time until someone tries to build the newly updated payroll system. This is when problems that were lying dormant in the system might be discovered. Many types of problems could arise: from a record-length definition incompatibility between a COBOL program and a JCL calling the program, to any other integration problem within the entire system.

IBM Rational Team Concert stores traditional artifacts and more recent artifacts in one place, enabling team members to easily access and share them.

Jazz sets the tone for IBM Rational Team Concert software

Enterprise applications have to handle many different artifacts, and the tools provided to developers must operate efficiently and transparently. Handling enterprise development styles means that Jazz technology–based products like IBM Rational Team Concert software can store traditional artifacts (such as COBOL, RPG, PL/I and High Level Assembler [HLASM] programs, and JCL and Control Language [CL] batch scripts) as well as more recent artifacts (such as EGL programs; HTML, JavaServer Pages [JSP] or JavaServer Faces [JSF] Web pages; Web Services Definition Language [WSDL] or any Java EE artifacts) in one place and share them between teams and team members. And Rational Team Concert can store these artifacts in a way that is independent of the tool being used.

Highlights

This aspect of enterprise development also helps build applications. Rational Team Concert enables developers to build applications that execute on IBM i and System z platforms, independent of where the Rational Team Concert server resides.

Rational Team Concert features work items, developer builds and SCM tools that think and work in unison.

In the IBM i and System z contexts, IT people have established elaborate, consolidated processes over the years to support software configuration management (SCM). To accommodate these processes and extend the capabilities of the team, Rational Team Concert features integrated work items, developer builds and SCM tools that think and work in unison. The Rational Team Concert SCM component allows artifacts to be developed locally in the development environment client using IBM Rational Developer for System z or System i software, or it allows artifacts to be developed remotely on the enterprise file system using the client for management only. The SCM component can also be used during all phases of the artifact lifecycle, including editing, testing, building and debugging in a cross-platform development context.

Providing the base technology and infrastructure to define process-aware operations, Jazz technology can be easily extended to fulfill the needs of an enterprise development environment. And new components can provide enterprise-specific work items, such as tasks, defects, enhancements and plan items, as well as the processes for creating them. Enterprise stakeholders can create work items using the technological entry forms and templates that enable those users to get a more transparent view of complex IT systems and their original developments.

Highlights

The process adviser framework in Rational Team Concert allows tool developers to implement their company's or team's unique development practices.

To help teams communicate the status of artifacts, Rational Team Concert provides tools for creating, versioning and exchanging update information.

At the heart of Jazz is a process infrastructure tightly linked to all the Jazz components that help development teams follow well-defined processes. Part of this infrastructure is the process adviser framework, where tool developers can extend the Jazz platform to implement their company's or team's own development practices and business policies. Building these controls can be a challenge for tool developers because the programming languages provided by the development platforms are generally not accessible to developers for testing. And when they *are* accessible, there is no common way to build and provide these controls to the platform.

Rational Team Concert offers a framework that can ease the development of process adviser tools to allow users to implement controls over enterprise languages. To make things easier, Rational software furnishes adviser examples based on programming guidelines for the most common enterprise programming languages, such as COBOL, RPG, PL/I and EGL.

As enterprise development organizations create artifacts, they generate large volumes of documents detailing which artifacts are being promoted, which are being deployed and which are being removed from production. These documents casually flow between teams, but their information is critical, particularly when it needs to be used to address security or legal compliance regulations.

The solution is to provide a simple but persistent method for creating, providing versions of, and exchanging the documents between teams and among team members. The Rational Team Concert SCM component can handle the document versioning, and the software's process infrastructure can handle the different flows.

Highlights

Tools based on Jazz technology can be used to simplify the process of updating the code in our payroll example.

To help communicate the status of application updates, tools based on Rational Team Concert trigger e-mail notifications to inform team members of their individual tasks.

Rational Team Concert takes the stage

Let's look back at our payroll example. The work could be simplified by using a new set of tools based on Jazz technology and designed for enterprise development. First, the project manager starts with the requirement for removing an absence-related code. In spite of the fact that it is a simple requirement, it would be inappropriate to make the request through an informal, untracked communication method such as e-mail or instant messaging.

Instead, using Rational Team Concert, the requirement details can be fully explained in the context of the request. The request details can include related tasks along with the deadline for when the new code needs to be operating in production. To do so using Rational Team Concert, the project manager creates one or more work items directly within the client that could involve an application like IBM Lotus Notes® software or a simple Web browser.

While using the same tool, the project manager asks another team member to provide more details about the payroll system components that would be affected by the changes. The project manager then reassigns the associated work items to the teams responsible for the various updates.

Via enterprise development tools based on Rational Team Concert, e-mail notifications inform the teams of their newly assigned work items and of updates to existing items. The developer or team leader in charge of EGL code learns of the new task and reviews the team workload displayed on the team central view. He or she then reassigns the work item to the appropriate developer. By knowing the entire context of the assigned task, the EGL developer can quickly start working on it. Upon reading the task description, the developer has access to references—presented through hyperlinks—to the EGL program and to records that need to be updated.

Highlights

Rules and controls established in Rational Team Concert can warn the developer that one of the programs is flawed and must be corrected.

Using these hyperlinks to load and open the EGL editor on the program, the developer can now begin his or her work. From this perspective, the EGL developer is in a familiar environment where content assistance and other code productivity tools can help him or her complete the task. And after clearly documenting the EGL code update performed in the task editor, the developer changes the work item status to *resolved* to reflect its completion.

Now that it is associated to the task, the EGL code update waits to be delivered to the other members of the EGL team. It is presented in the *pending changes* view until the developer requests delivery.

During delivery of the pending update, the process mechanism applicable to the EGL development team at this stage of the project is executed automatically. And the rules and controls defined are processed against the code being delivered. Through the automated adviser, the control settings can warn the developer that one of the programs is incorrectly named and should be changed. This can save the development team from having to later correct the naming and repeat steps.

For all the teams involved in this update, the work is almost finished. The project manager, who subscribed to the different work items involved in this project, is notified of every change. Rational Team Concert, through an Eclipse client or a browser-based interface, provides a set of tools that augment the visibility of the project—from simple notifications to highly specialized dashboards and reports. The most used feature is the iteration plan, where all contributors can see the exact status of the work for an entire project.

Highlights

The newly updated system is about to be integrated into one cohesive unit; it will then be compiled and tested before being deployed to the requested systems on which the components will run. To simplify this highly complex process, the construction of a heterogeneous and cross-platform system is directed by a single conductor, Rational Team Concert.

Rational Team Concert takes all the updated code from a single stream and provides a functional map of the entire payroll system.

Scheduled or manually requested, the integration build performed by Rational Team Concert in our payroll example takes all the updated component code from a single stream, provides a coherent functional map of the entire payroll system, and lists the programming languages used (COBOL, EGL and JSP). The affected applications are recompiled and unit tested, and, before they're deployed to either an IBM i, a System z or a distributed system, the build results are automatically delivered to the subscribed contributors. During this operation, the functional system map makes snapshots for a later use, like for code maintenance or recovery.

Highlights

You can take better advantage of your IBM i and System z platforms by using Rational Team Concert to manage and transparently share development artifacts.

Benefits for IBM i and System z developers

By leveraging Jazz and Rational Team Concert technology inside the enterprise space, siloed developers can collaborate; share and exchange data with their colleagues to ease the software delivery process; control what they deliver; and get early feedback on what they plan to deliver.

Jazz can help consolidate multiplatform development, including IBM i and System z technology, by potentially improving IT governance, reducing development and maintenance costs, and enhancing the quality of delivered software. And Rational Team Concert for IBM i or System z features can be used out of the box or tuned to fit enterprise development styles, governance and IT infrastructure without breaking current operating procedures.

By using Rational Team Concert to manage and share all development artifacts in a common and transparent way between teams, team members and stakeholders, you can more effectively leverage the strength and scalability of your IBM i or System z development architecture, regardless of your staff's global location or time zone. These platforms offer robust quality of service to help make work more effective with multidisciplinary teams and business partners.

As Jazz technology–based products are deployed for enterprise environments, application development in large organizations has the potential to become streamlined and transparent, and software delivery can be made more predictable than ever before.

For more information

To learn more about IBM Rational Team Concert software, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/awdtools/rtc

To learn more about IBM Jazz technology, visit:

www.jazz.net

About the author

Jean-Yves Rigolet is a software engineer with more than 18 years of application and development tools shipping experience in a variety of programming environments and platforms. He is currently working at the IBM France Software Laboratory on Rational Team Concert for IBM i and System z. You can reach Jean-Yves at rigolet.j@fr.ibm.com.

RAW14040-USEN-00