

Brought to you by

**InfoQ**  
Enterprise Software Development Community



**IBM**

# Scaling Agile With C/ALM (Collaborative Application Lifecycle Management)

A Scenario - based  
approach for  
overcoming  
the challenges  
of team  
collaboration



Brought to you by



© Copyright 2009 IBM Rational

All rights reserved

No part of this publication may be reproduced, stored in or introduced into a retrieval system, or transmitted, in any form, or by any means (electronic, mechanical, photocopying, recording or otherwise), with the prior permission of IBM.

# About the Authors

**Carolyn Pampino** is a member of the C/ALM leadership team at IBM Rational's Lexington lab, working closely with the Jazz team leads to define the Collaborative ALM road map and strategy. Working from the 'outside-in' Carolyn defines user scenarios to identify product integration opportunities. She is co-author of a Redbook titled Collaborative Application Lifecycle Management with Rational products. She was a founding member of the team defining strategies for Rational and Tivoli product integrations, and contributed to Rational's acquisition of Build Forge. Prior to IBM, Carolyn was the Director of Product Management, Development, and Competitive Intelligence at BroadVision, Inc.

**Erich Gamma** is a Distinguished Engineer at IBM Rational Software's Zurich lab. He is the technical lead of Rational Team Concert and is a member of the C/ALM leadership team. He was the original lead of the Eclipse Java development environment and was a founding member of the Project Management Committee for the Eclipse project. Erich is also a member of the Gang of Four, which is known for its classical book, Design Patterns - Elements of Reusable Object-Oriented Software. Erich has collaborated with Kent Beck on developing JUnit, the de facto standard testing tool for Java software, and on writing the book contributing to Eclipse: Principles, Patterns, and Plug-ins.

**John Wiegand** is a Distinguished Engineer at IBM Rational's Beaverton, Oregon lab and Rational Chief Architect. John is responsible for defining the architectural and implementation aspects of Jazz as a platform for use in products across the software lifecycle. Prior to John's current assignment, he was the technical lead for the Jazz project. John was the principal architect for the Eclipse Platform infrastructure and played a central role in the development of Eclipse and VisualAge for Java. John is a former member of the Eclipse Foundation Board, and played a key leadership role in establishing Eclipse as a successful open source project. John strives to enable teams to deliver high quality products on-time - pioneering, with Erich Gamma and others, an approach to software development called "The Eclipse Way".

# TABLE OF CONTENTS

- 1 Overview: Wasted words, or key to understanding?
- 2 A scenario tells a thousand stories
- 3 The WWW (Web We Wove)
- 4 ALM Ecosystems – one size fits no-one, or haute couture?
- 5 Open Sesame: A phrase uttered by a genie, or IBM Rational strategy?

## Chapter 1

### Overview: Wasted words, or key to understanding?

- 1.1 Integrations enable collaboration
- 1.2 A programmable C/ALM web
- 1.3 A concrete example in the form of a scenario
- 1.4 One example, generally applicable

## Scaling Agile

Let's face it; most software development teams indulge in tribal behavior. If you're working in a large enterprise the divisions between the business, development and testers can be deep enough that these three groups reject the idea of being on the same team. Even within teams you can have deep divisions. For example, developers loyal to traditional methods, and those committed to agile approaches. Each group has their own culture, their own rituals and their own tools. Development and testing teams fluctuate between war and peace but for the most part have managed to come to understand each other. Developers lob builds over the fortress wall, testers shoot defects back at them. Unless, you're part of an Agile team, but even in Agile shops some testers (e.g. integration, system, and deployment testers) are likely to be outside the agile "whole team."

But when it comes to "software teams versus the business," it's a completely different story. People representing the business speak a different language that developers and testers struggle to understand. The testers quickly discovered an ally for shooting arrows at the mighty fortress of development. The developers however, just think they're crazy. After all, what they want and what is realistic is often so far apart, it's easier to just not talk. So the business throws requirements at the development teams. The development team tosses them in the trash. The test team validates the requirements against the builds and shoots defects at the development team. Much to everyone's dismay, Agile and Agile@Scale suggest these three groups collaborate as a cohesive unit.

Many books on Agile are written, and yet there are still many project managers left scratching their heads, wondering, "how am I going to get my team to do that"?

This eBook is dedicated to all of the functional and dysfunctional organizations that are eager to break down the organizational and cultural silos, and become a finely tuned software delivery machine. The eBook describes a scenario that is common to many development teams. Throughout the scenario, we encounter collaboration challenges followed with examples of how these teams can interact without investing enormous effort in cultural change. We believe each team member deserves to use a tool that best suits their needs, that enables them to collaborate across the software delivery disciplines. All of this can be achieved in the open using open Internet inspired standards.

**Agile@Scale addresses one or more of the following scaling factors:**

- Team size
- Geographic distribution
- Organizational distribution
- Regulatory compliance
- Environmental complexity
- Enterprise discipline

## 1.1 Integrations enable collaboration

Without integrations across the lifecycle teams are left to operate in silos. When silos form, trust deteriorates, and software delivery suffers. However, software delivery teams and their software development environments are fluid. Rather than provide a 'one-size-fits-no-one' solution, the Jazz Integration Architecture follows the model of the World Wide Web where sites are loosely coupled through the linking of resources and thereby making repository boundaries disappear.

- **Data integration** via linked artifacts across repositories using RESTful interfaces
- Leads to increased **collaboration** among team members who link, navigate and track the status of delivery team artifacts
- And enables **automation** such as real-time reports and queries
- Resulting in increased **transparency** for everyone

By providing C/ALM integrations in a loosely coupled way, development tool vendors can provide you with the freedom to choose the combination of products that best suits your needs.

## 1.2 A programmable C/ALM web

Many organizations cite the need for traceability across disciplines, but for the most part, you can't do this today because your tools don't support it. There's no doubt that members of the business, development and testing organizations know their disciplines and have tools that help them perform their work. However, in many cases, each discipline uses their own tool with proprietary APIs, and the data is locked away in the tool. The key to agile development, or any software development for that matter is not the collaboration within an individual discipline, but collaboration across disciplines.

On agile teams, the feature is not considered done until testing is complete and defects are fixed. To accomplish this goal, however, teams need traceability and transparency across disciplines. For example, having a test case linked to the development work item, provides insight to when testing is complete, that is, all the test cases for the work item passed. Defects discovered during test execution should have relationships to both testing and development to inform both teams on the quality of the software.

Integration across the application lifecycle is perhaps the most important challenge facing software delivery teams. While it makes sense to consolidate our repositories where possible, it is also true that important project resources already exist in a disparate set of repositories across teams and organizations. For most, migrating sensitive enterprise data into a single ALM repository and retraining the entire software delivery workforce on new tooling is not enticing, and may not be an option.

Rather than provide a one-size fits no one solution, many organizations prefer to allow each stakeholder to have a user interface that suits their needs. The key to doing so is to unlock the data housed in these repositories and enable loosely coupled data sources to share and present information in a cohesive way. Many attempts to resolve this challenge have been made by software development tool vendors and even by teams creating home grown solutions.

It turns out that an architecture already exists that achieves this goal, the internet . It is resilient. Users can surf from site to site. The ability to get and put data is fundamental in its design. It scales. It's global. Using the Web as inspiration and metaphor, the Jazz project was born. The Jazz project includes the following:



- **Collaborative Application Lifecycle Management (C/ALM) Scenarios** –the C/ALM scenarios work from the outside-in by providing real-world, role and task based user experiences that explore end-user goals and their needs to access data throughout the lifecycle. They are designed to validate the Jazz Integration Architecture, Jazz Foundation, and OSLC specifications.
- **Open Services for Lifecycle Collaboration (OSLC)** – to unlock the information buried within development tools, open and agreed upon interfaces are needed that allow different tools to share and exchange the data that they produce. By agreeing on common specifications for lifecycle resources and the services to access them, the community behind OSLC seeks to eliminate traditional barriers between tools and open the door to new forms of collaboration.
- **Jazz Integration Architecture (JIA)** – a set of inter-connected technologies and specifications, consisting of reference architecture, API specifications, a set of common services and tool building blocks. At the center of JIA is the Jazz Foundation Services which provides services to enable groups of tools to work together. Powering much of JIA are standard RESTful APIs and standard resource definitions which enable participating tools to easily share data.
- **Jazz Foundation** – an implementation of the Jazz Foundation Services, and optional toolkits to aid in the construction of Jazz applications.

The Jazz project integrates and coordinates the architecture (JIA), the services (OSLC), and the C/ALM scenarios to allow the coordinated flow of information essential to successful development. Moreover, since the tools and the foundation expose RESTful APIs, it is possible for development teams to customize their integration to best suit their choice of development methodology.

## 1.3 A concrete example in the form of a scenario

This eBook uses a scenario that specifically highlights integrations among Rational Team Concert, Rational Quality Manager, and Rational Requirements Composer. These integrations empower teams to collaborate across the application lifecycle.

- [Rational Requirements Composer](#) provides a platform for collaborative requirements definition that enables business analysts, client stakeholders and software development teams to define requirements using a variety of techniques, and collaborate with each other using a suite of platform capabilities.
- [Rational Team Concert IBM Rational Team Concert](#) is a team-aware software development platform that integrates work item tracking, builds, source control, and agile planning. Rational Team Concert interoperates with other products by providing Visual Studio integration and connectors for ClearCase and ClearQuest.
- [Rational Quality Manager](#) provides a centralized test management environment to help increase the efficiency and quality of software delivery by mitigating risk and lowering cost through collaborative ALM for test planning, workflow control, tracking and traceability, and metrics reporting capable of quantifying how project decisions impact and align with business objectives. Rational Test Lab Manager, which is an extended component of Rational Quality Manager, helps to improve the efficiency of the test lab environment and optimize its utilization, cutting workload and saving on test infrastructure.
- [Jazz Foundation](#) provides a scalable, extensible team-collaboration platform that integrates tasks across the software lifecycle. The platform also provides useful building blocks and frameworks that facilitate the development of new products and tools.

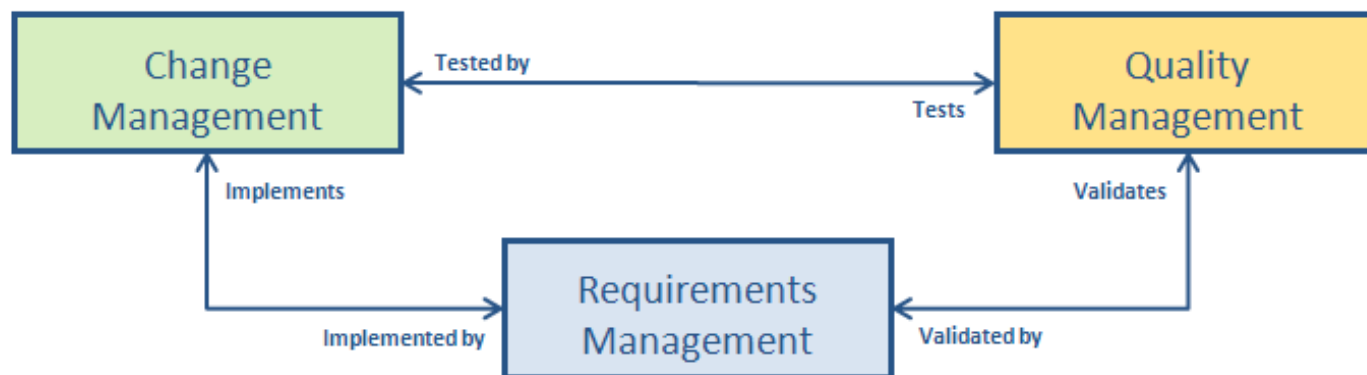
Note this is not a one-size-fits-no one solution. The products are developed and shipped independently with the integrations built in. This allows you to have an iterative approach to products adoption.

In addition to supporting a team's ability to select and integrate their toolset, the C/ALM project enables delivery of important new features. Some examples:

- Surf the Collaborative ALM web of artifacts. Users don't always care about the tool that created the data, they just need access to the information. Therefore links appear on artifacts just like any other link. By hovering over a link, users can get a rich set of information about the target at the end of the link, helping them to make decisions on what to do next. Clicking, of course navigates to the artifact at the other end of the link.
- With In-context Collaboration users link to (or create) software artifacts in other software development repositories without leaving their primary tool. Several examples of in-context collaboration are provided.
- Mash it up! Dashboards and widgets provide transparency across the disciplines. Throughout the lifecycle team members are dependent on each other's status to perform their own tasks, and yet they have very little insight into that work. Dashboards and widgets give team members instant access to each other's status.
- C/ALM Queries: Answers to a team's more interesting questions. In addition, a set of Collaborative ALM queries are provided that give users additional insight regarding their linked artifacts. For example, developers can use a dashboard widget to see which stories have failing test cases. This query finds all open Stories with links to Rational Quality Manager test cases, where the test status is failed. This helps a team determine when the "Story" is done – it is done when the tests pass. In fact, in an integrated team it is the tester that sets a Story to the done state.
- Enterprise Reporting – Identify what works and what doesn't by collecting real-time data from a wide variety of development tools to track project health throughout the life cycle.
- Learn more about the Collaborative ALM project and supported scenarios at [jazz.net](http://jazz.net)

## 1.4 One example, generally applicable

The challenges presented in this eBook are independent of the tools or process a team uses. We selected a specific scenario and a set of tools to provide a concrete, real-world example; however, the challenges presented can be resolved by a wide variety of tools.



**FIGURE 1.** RELATIONSHIPS BETWEEN SOFTWARE DEVELOPMENT DISCIPLINES

The scenario can be abstracted into relationships across three important disciplines: requirements management, change management, and quality management. The abstraction is an important realization in our quest to integrate software development ecosystems. Rather than looking at the solution as a monolithic software application, we view it as a set of services. C/ALM solutions can act as providers and consumers of software delivery services that come from a variety of vendors. This is an open and flexible approach that invites all software delivery tools vendors to provide public, and open APIs for providing and consuming services.

In addition, tools that integrate the Jazz Foundation provide the additional benefit of a common UI framework, link-types describing cross-discipline relationships, rich hovers, queries, dashboards that host cross-repository widgets, and much more.

In this eBook we present one representative approach and example for how a team might address and resolve a particular challenge. We acknowledge different teams will choose different tools or emphasize different aspects of the same tools we chose for our illustration. Our example and the specific tools we chose to highlight are not intended to be “The Only Answer,” rather, our presentation is intended to illustrate a generally applicable issue or problem that can be solved by tools vendors in an open, and resilient way.

**A scenario**  
*tells a thousand stories*

## Chapter 2 A scenario tells a thousand stories

- 2.1 Requirements: dirty word or diamond in the rough?
- 2.2 Sprint Backlog: Creative Exercise or Reflection of Reality?
- 2.3 Testers: Third wheel or Magic Number?
- 2.4 "The Build": A present from an eccentric relative, or something I want?
- 2.5 Defects: Odd game, fiction or fact?
- 2.6 Sprint Review: Rorschach test or view of Reality

## 2 A scenario tells a thousand stories

The ideas presented in this section are generally applicable to any team. To provide concrete examples of working software in the context of a team, we will use one of the C/ALM scenarios to illustrate the utility of scenario-based development (Figure 2). This scenario assumes an integrated team with analysts, developers and testers interacting within the same iteration. The scenario also provides the structure to communicate the overall value of Collaborative Application Lifecycle Management (C/ALM). Throughout the scenario we come face to face with common collaboration challenges and present an approach for overcoming them.

The challenges the team encounters are as follows:

- Requirements: How a team establishes a shared vision using a rich definition.
- Sprint backlogs: How a team can align their shared vision with a dynamic, up-to-date agreed upon plan.
- Testers: How teams incorporate the test plan.
- The build: How do testers know the quality and contents of a build coming from development?
- Defects: How to link test execution results with defects, How can developers reproduce test failures easily?
- Sprint Review: Throughout the sprint the team can assess their progress and, using live-data, assess when they are done, done, done.

According to the scenario, our team is a geographically distributed agile team using Scrum. Any management process could have been applied, but for this example, we will use Scrum. In addition, our team integrates a set of tools that forms their C/ALM delivery environment.

For more information on scrum see:

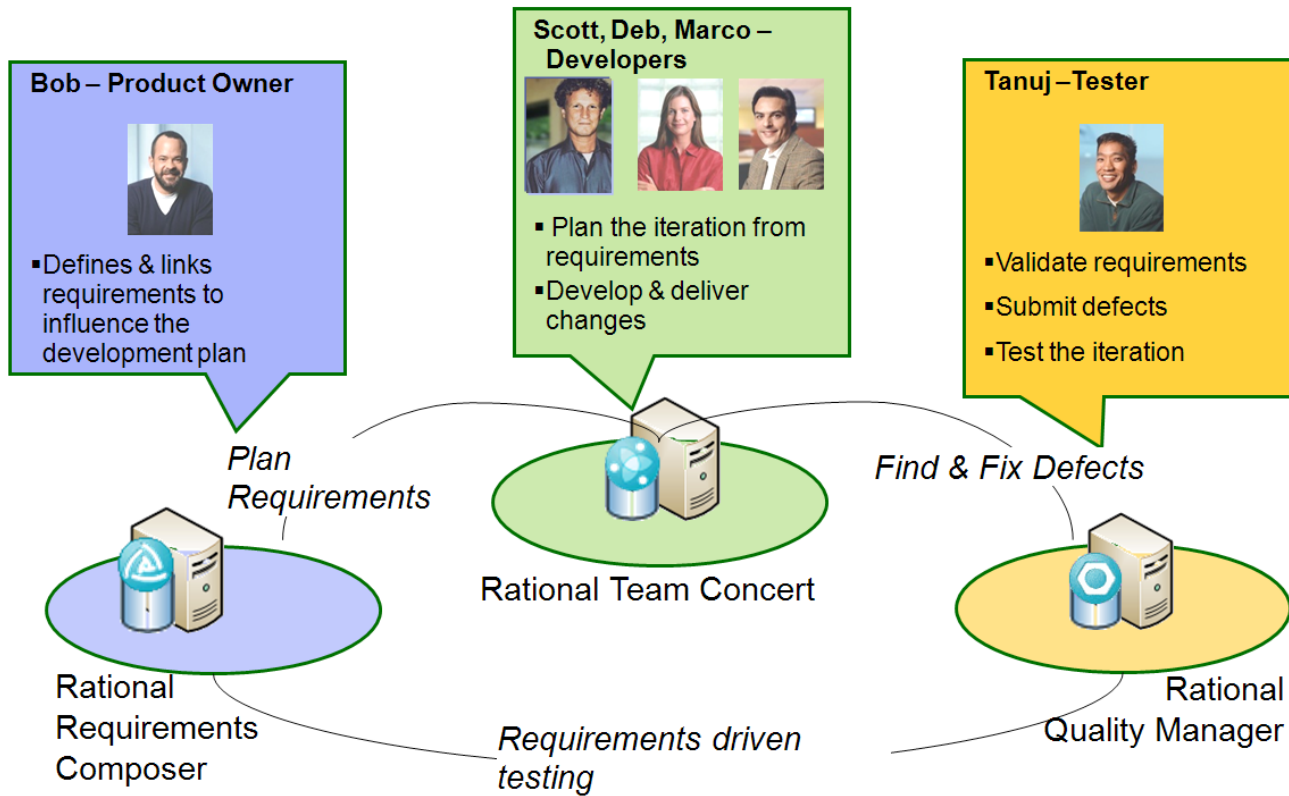
[Scrum Adoption in China](#)

[Scrum at Dutch Railways](#)

[Kanban and Scrum](#)

[Scrum and XP Minibook](#)

[Scrum Checklists Minibook](#)



**FIGURE 2.** A SPRINT IN THE DAY OF THE LIFE OF A DISTRIBUTED AGILE TEAM



Our team and the tools they use are as follows:

- **Bob:** Product owner, responsible for representing the business and stakeholder needs. His goal is to effectively communicate his vision for the development team and to prioritize the backlog items. Bob collaborates with Scott and Tanuj on the product and sprint backlogs. Bob uses Rational Requirements Composer to define and manage his requirements. (For non-agile teams, the role is typically represented as an “analyst” or “business analyst.”)
- **Scott:** Scrum Master, responsible for coordinating the development team, tracking impediments, running the daily stand-ups and moderating the sprint planning session. He uses Rational Team Concert together with the Scrum process template for managing the product backlog and for planning and tracking sprint backlogs. Scott collaborates with everyone on the sprint backlog.
- **Marco and Deb:** Team lead, Developer, responsible for developing and building the software. They use Rational Team Concert for sprint planning, source code management, work-item and defect management, and continuous builds. They collaborate with Bob & Scott on the sprint backlog and with Tanuj when he submits a defect. There are other developers on the team, but they play the same roles as Deb or Marco and can be considered as role instances.
- **Tanuj:** Tester, responsible for exploratory testing and an integral member of the team. He uses Rational Quality Manager to manage his test plans, cases, scripts and to track and report against test execution status. Tanuj collaborates on the product backlog with Bob and Scott, and he also collaborates with Deb on fixing a defect.

Let’s see their collaboration challenges and how they overcome them.

## 2.1 Requirements: dirty word or diamond in the rough?

Agile teams don't like the word requirement, and frankly when we think of the days of terse, incomprehensible, un-prioritized, ever changing, line-item requirements, neither do we! It's important to involve members from the business because they have the best understanding of the goals for the software that is to be built. When done well, requirements provide a rich vision of what the enterprise and all of its constituencies require from software applications. In this section we'll show you why we believe "requirements" are like diamonds in need of a little polish.

### 2.1.1 Collaborating toward a Shared Vision

Bob is the product owner responsible for defining the needs of the business and its stakeholders. Bob listens to many stakeholders, each one with their own specific needs, finds commonality and negotiates a concerted approach to defining needs. Unfortunately, it can be really hard to summarize this rich vision in a set of line-item requirements, crammed into a single row in a spreadsheet representing a 'backlog' ... especially when Bob "sees" things, – insights that arise because Bob is so conversant with the "Big Picture," – insights that he visualizes as images and models. To effectively collaborate with his software development team, Bob needs a requirements definition tool.

In an ideal agile world, that tool would be a story card. Bob's team is geographically distributed and working in multiple time zones. To provide an electronic depiction of his interactions, Bob chose Requirements Composer, (Figure 3) which has features to help him capture his ideas in the form of sketches, storyboards, business processes, rich text, glossaries and use cases.

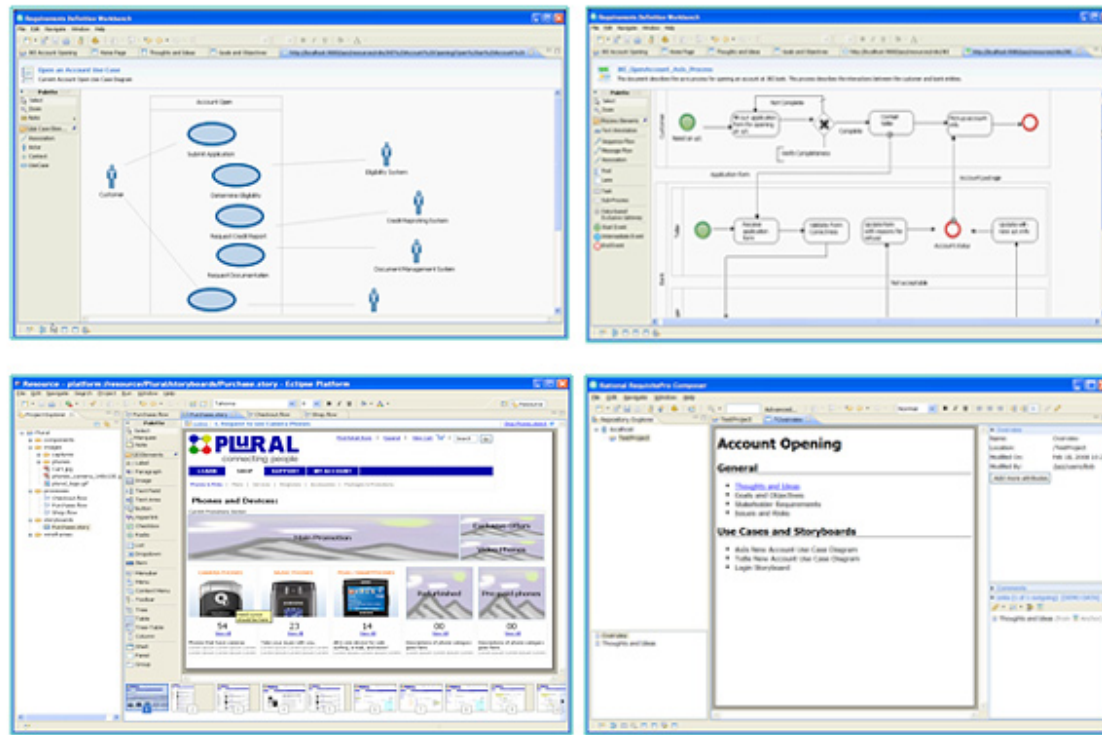


FIGURE 3. BOB “SEES THINGS” USING RATIONAL REQUIREMENTS COMPOSER

Using Requirements Composer, Bob collaborates with the business domain experts until he is ready to initiate a dialog with the developers. By using the Web interface provided by Requirements Composer, developers and stakeholders can be geographically distributed, while seamlessly accessing and commenting on the data. Bob can review and respond to the comments as they are submitted, or conduct reviews until the team comes to a shared vision.

This dialog addresses multiple goals. One, Bob needs to confirm his ability to articulate the business vision, ideally using the same models and artifacts he used to extract that vision from the domain experts. Second, Bob needs feedback - a healthy dose of reality in terms of time and cost to develop – that he can use to negotiate and prioritize activities with the business side of the house.

## 2.1.2 Requirements are diamonds in need of a little polish

A simple example of web-based stakeholder collaboration using comments and reviews was used to show the value of communication across silos, specifically enabling communication between members of the business silo and the development silo – with Bob as our facilitator.

This represents a major leap forward in business-development relations, and it's great that the tool allows everyone to review and comment on the requirements, but what is the relationship between these models and artifacts (often annotated drawings at this point) and the developer stories on the product backlog?

## 2.2 Sprint Backlog: Creative Exercise or Reflection of Reality?

Big plans, little plans, shifting plans, stagnant plans they all have the potential for becoming creative exercises rather than reflections of reality. Many Agile teams have learned to execute against realistic plans by working with public task boards and moving a story and its related tasks through their phases to completion.

If you are part of a geographically dispersed team, or even a co-located team coordinating with other co-located teams in the context of a larger project, traditional approaches are inadequate. Planning in such circumstances tends towards an exercise in creative writing. In addition teams have multiple sources of information to work with – defects in one tool, requirements in another, resource information in another – making it challenging to coordinate a plan.

Through the use of in-context collaboration for linking artifacts across disciplines and the use of dynamic plans where the data is coming directly from the work-items owned by the team members, we'll show you how the plan can be a reflection reality.

### 2.2.1 From vision to execution: updating the product backlog

Bob's development team uses Rational Team Concert's planning feature to manage the backlog. Bob's challenge is to link the rich set of requirements he defined in Rational Requirements Composer with a set of work-items prioritized on a backlog maintained by Rational Team Concert.

This separation between requirements and development plans is a common challenge that all teams must overcome. Using in-context collaboration the product owner, Bob, will bridge this gap by linking requirements and development work-items without having to leave Composer, the tool with which he has become accustomed; nor will Bob be required to learn Team Concert features in order to make this transition.

Bob has two choices. He can link his requirement to an existing work-item in Team Concert, as shown in as shown in Figure 4, or he can create a new work-item.

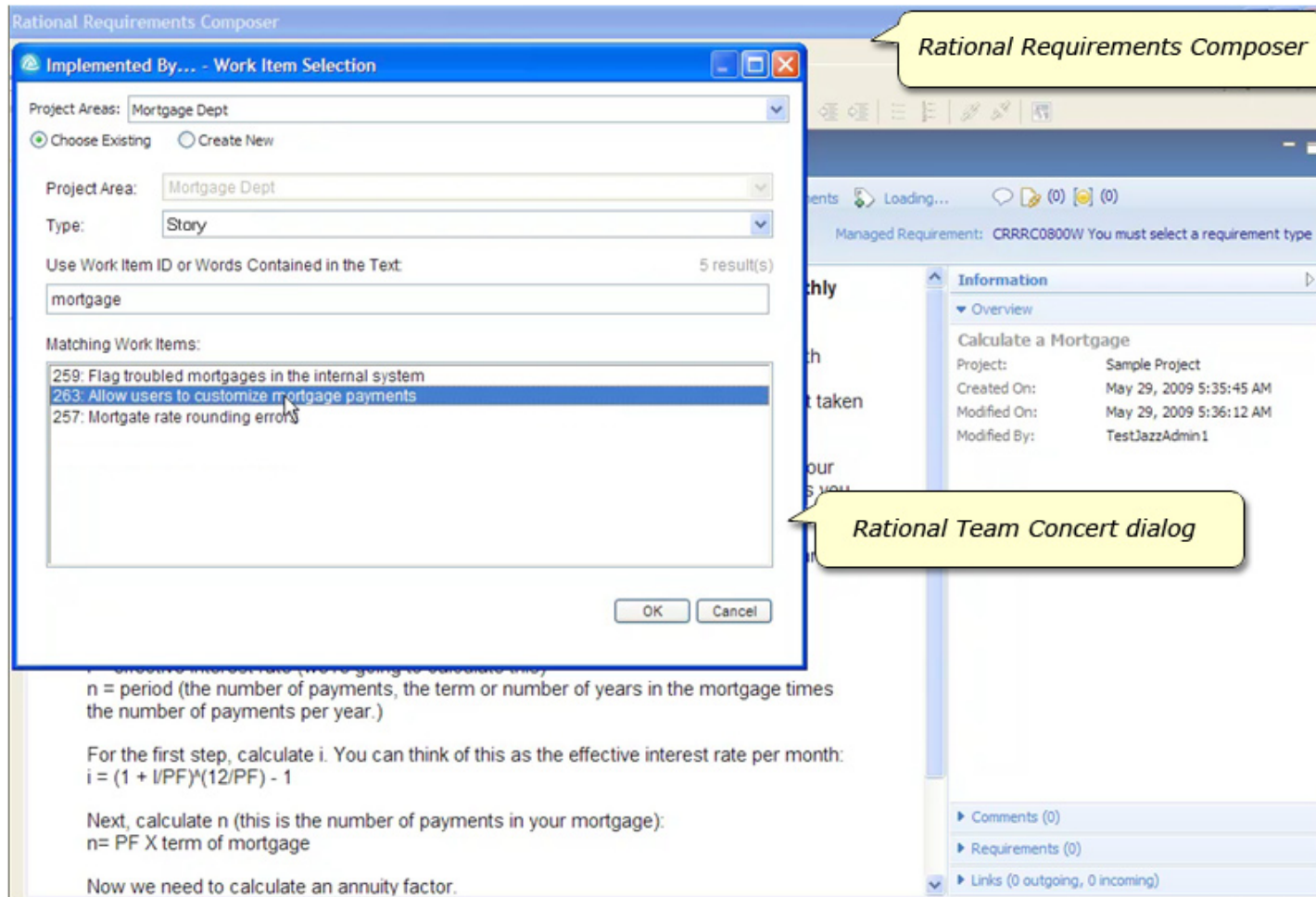


FIGURE 4. BOB USES TEAM CONCERT'S 'LINK PICKER' TO LINK TO AN EXISTING WORK-ITEM

Notice, Bob did not leave the Requirements Composer user interface. Instead, a dialog with information coming from Team Concert came to him! When he types a keyword, such as 'mortgage' the matching Work-items field is automatically populated with work-items managed by Team Concert. When he clicks the ok button, links are created between the two artifacts that identify the relationship.

Bob's requirement is now linked to a story work-item type on the backlog in Rational Team Concert. Clicking the link opens the work-item in the Rational Team Concert Web user interface. Bob can rank the stories on the back log, or use the mini editor, which is shown in Figure 5, to change the properties, summary or description of each of the stories. Once he's finished prioritizing the backlog, Bob saves the plan and exits the browser.

This use of delegated user interface greatly reduces the number of times Bob has to change tools, simplifies the integration points between tools, while leaving all of the power and semantics of each tool available to the users. The two repositories are communicating using RESTful interfaces and this interchange is an implementation of the Open Services for Lifecycle Collaboration change management specification. Whether he's linking or creating work-items, the dialog in Composer looks like all the other dialogs to which Bob is accustomed. Additional information is provided in Section 3.5, How it's woven



## 2.2.2 Sprint Planning, Goals and Backlog

Scott is the Scrum master responsible for making the team as productive as possible. He understands Scrum, is accustomed to using a public task board, but now his team has grown and is geographically distributed. Scott chose to use Rational Team Concert for planning, source control and continuous builds. He likes the idea of having dynamic transparent plans and team dashboards that are driven from the data produced by the team.

To conduct a geographically distributed planning meeting Scott and his team mates use the Web user interface to view the product backlog. The team uses the mini-editor to view the description of each story of interest. The mini-editor, shown in Figure 5, allows the team to quickly view and update each item without leaving the view of the backlog. This provides a context for their discussion and helps assure that decisions made in one story do not contradict decisions made when working with a second.

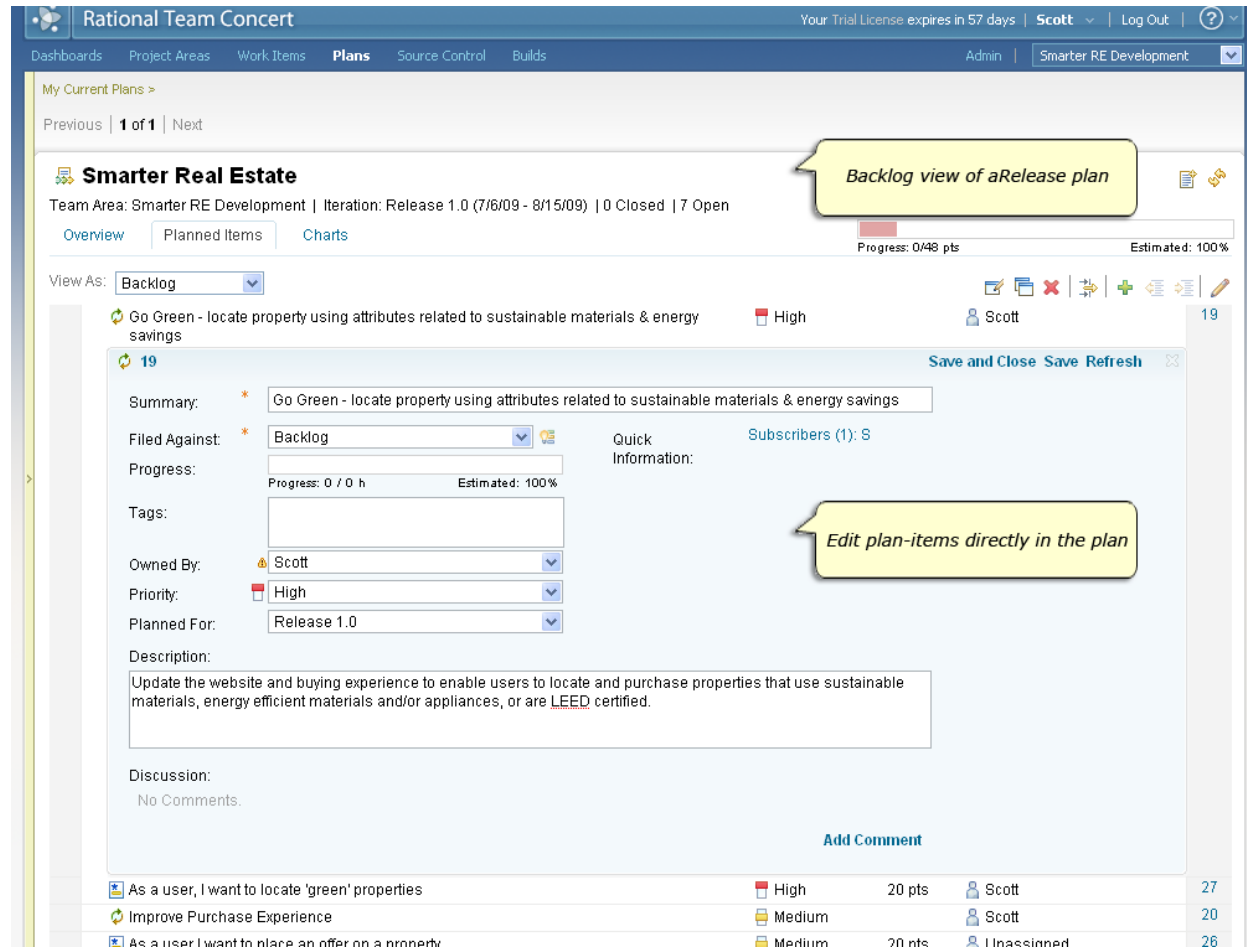


FIGURE 5. REVIEWING THE BACKLOG WITH THE MINI EDITOR OPEN



The team comes across the story created by Bob in the previous section and sees the link to a requirement. Clicking the link, as shown in Figure 6, opens the Requirements Composer Web user interface allowing the team to discuss the sketch Bob has provided. Note how this conversation provides many of the advantages of having an on-site customer. The integration and traceability offered by the tool allows the team to interact with the same models and have very similar conversations as they would have if a human user was in the room.

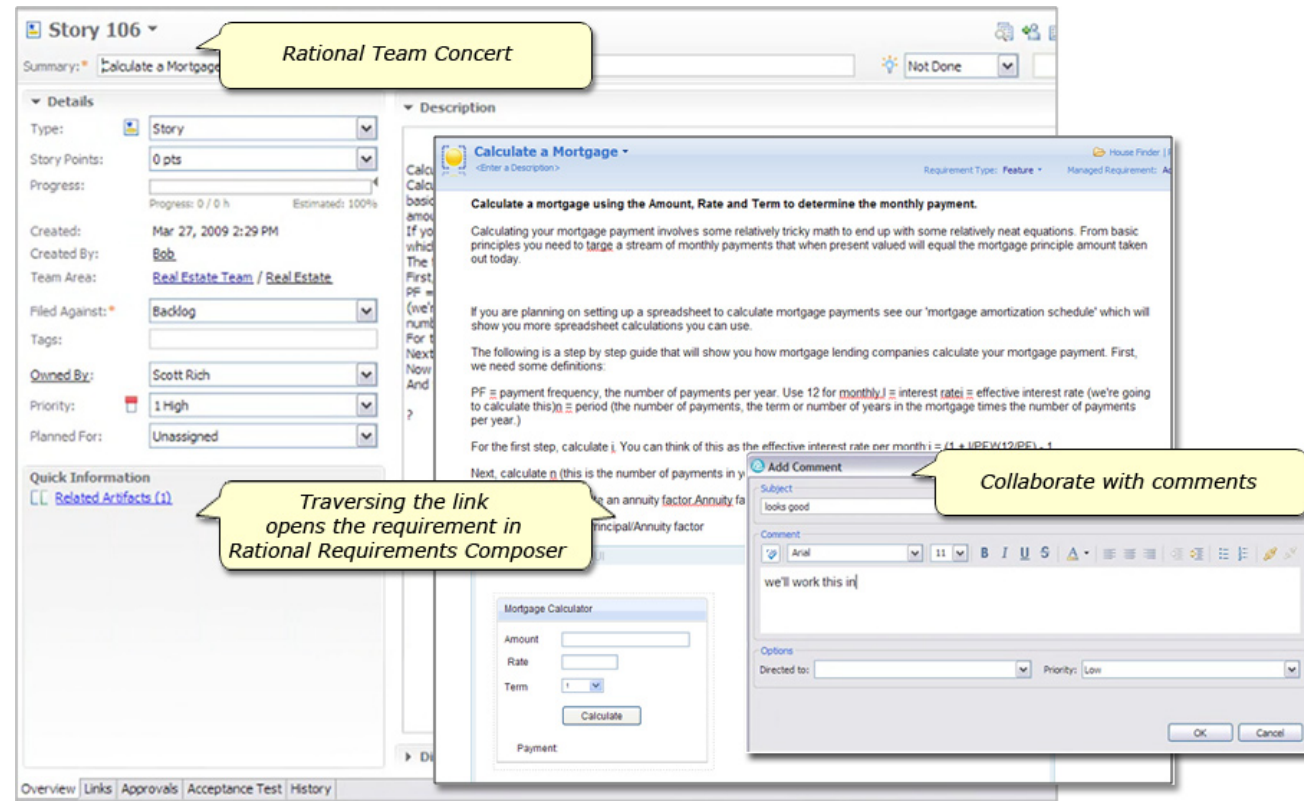
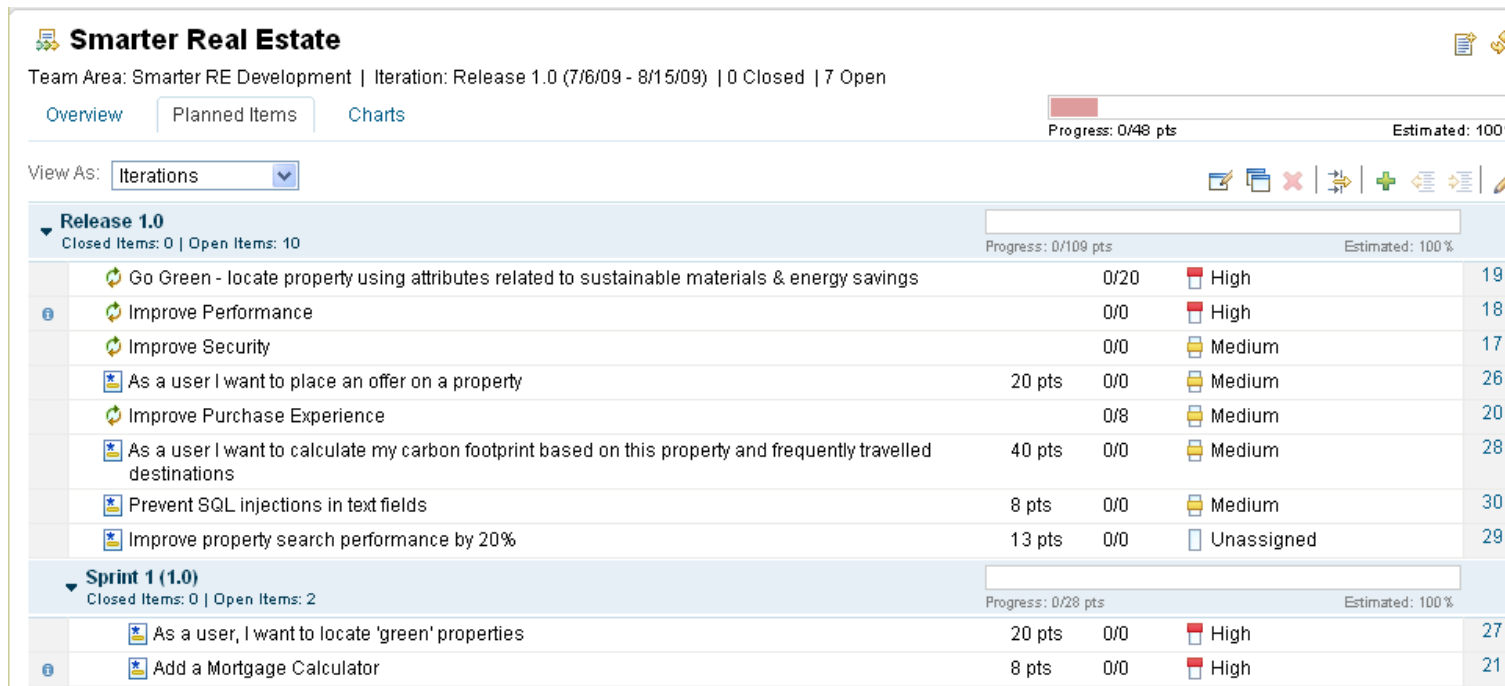


FIGURE 6. TRAVERSING A REQUIREMENT LINK FROM RATIONAL TEAM CONCERT

After reviewing the requirement, the team jointly agrees on the number of story points and Scott saves the changes. As the team moves through the backlog they can easily see the number of stories and story points they are committing to for the sprint. Upon reviewing a sufficient number of backlog items, the team is ready to create the sprint backlog.

Next Scott creates the Sprint plan and the team collectively agrees upon the goals and themes of the sprint, all of which are captured in the plan. Using the planned items tab (shown in Figure 7), Scott drags and drops items from the Product backlog onto the Sprint backlog.

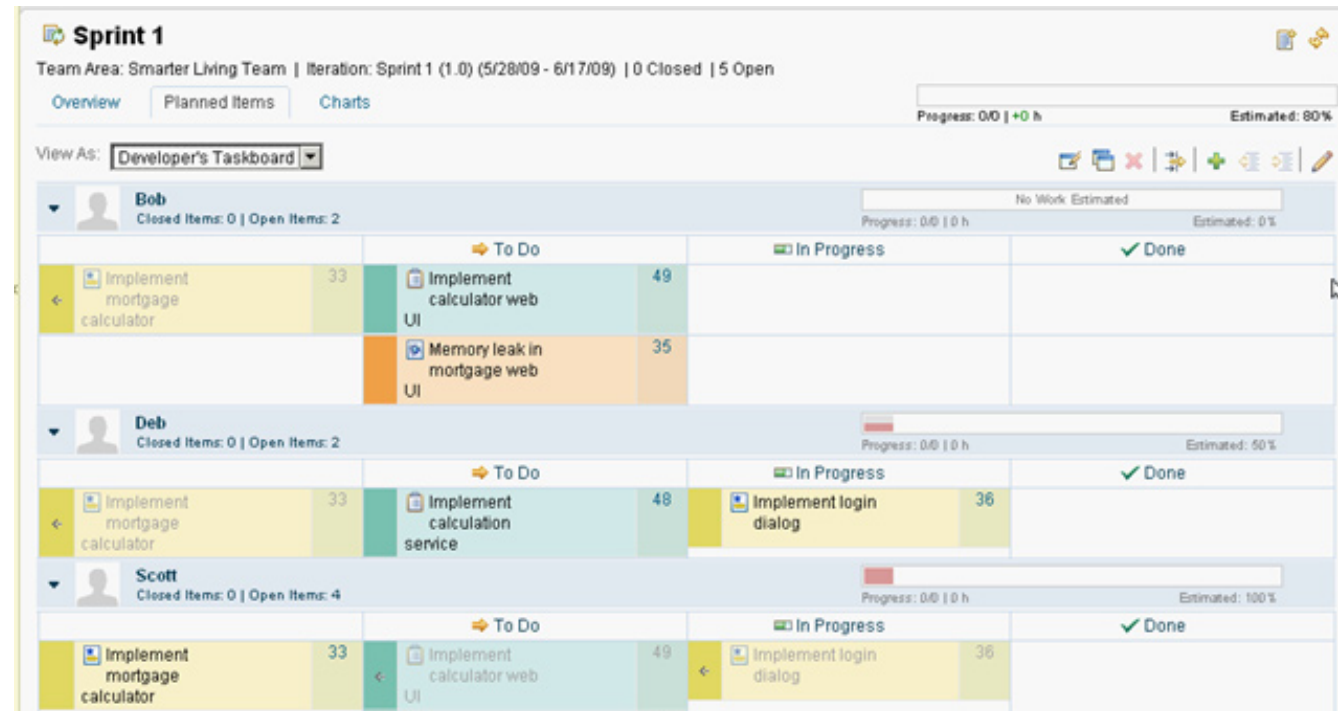


As the sprint backlog comes together, the team may have second thoughts or additional insights regarding each of the work-items. They can easily drag items into the sprint backlog, or drag them back onto the product backlog, until they reach an agreement. Scott saves the Sprint backlog, and the team is ready to get to work.

**FIGURE 7. SCOTT DRAGS AND DROPS ITEMS ONTO THE SPRINT BACKLOG**

## 2.2.3 Working with the Task board

Marco and Deb prefer using the Task Board view of the Sprint (Figure 8). With an electronic task board, teams can be geographically distributed and remain up to date with each other's progress. The task board is simply another view of the sprint backlog that Scott created. The data is the same, allowing each team member to choose the view that best suits their needs.



**FIGURE 8. MARCO AND DEB WORK WITH THE DEVELOPERS TASK BOARD**

As Deb works on the Story provided by Bob, she can navigate the requirement link to view Bob's sketch and requirements information. Doing so helps Deb determine which tasks are needed to complete the story. Using the task board view in Team Concert she decomposes the Story into developer Tasks.

The next time Scott views the plan he will see the progress made by the team. As each developer takes ownership of and begins working on their tasks, everyone will be able to see their progress. When they conduct their daily stand-ups they can collectively review the plan status, whether they're in the same room or distributed in multiple locations. The plan cannot only be reviewed but the state of work items can be updated as well.

## 2.2.4 Sprint backlogs are a reflection of reality.

We showed a simple way to link a sketch, a depiction of what the business sees as a requirement, to a story, a unit of work recognized by the development team, and place it on a product backlog. An example specific to Scrum teams was used to show the value of linking across silos, specifically enabling communication between members of the business silo and the development silo – with Bob as our facilitator.

The teams come together during the planning meeting to review and refine the items on the backlog. Requirement links on work-items provide additional detail which the team can navigate by clicking the link. The team can collectively agree to the items for the Sprint backlog, and can work with any number of plan views, such as the backlog or developer task board view. Collectively the team creates a shared vision for the software and these two groups that typically operate in silos can finally begin to collaborate!

## 2.3 Testers: Third wheel or Magic Number?

Mom always warned me about threesomes. If a third friend came along, she'd warn me that one always gets left out. On the other hand, there's a lot of folklore regarding the magic embodied in the number three. So far we have seen how to break down the silos between the business and development and now we need to explore how we can add a third set of colleagues – testers – to the mix.

Agile teams look to integrate testers as an integral part of the team, part of which involves “concurrent testing” which brings testers closer to development and requires tighter collaboration. Testers can provide valuable insights as we try to understand requirements, they can help the team define useful and valuable developer tests, and they can pave the way to smooth integration, regression, system, and deployment testing.

Yet, typically there's little transparency in the test effort. Every self-respecting tester knows what they've done and what they haven't, and that's great, but what about everyone else on the team? All too often test scripts and the execution results are either in the heads of the test team or buried in static documents where some of the text is up to date, some is stale, and no one can determine the difference. Knowing when testing is 'done' becomes one of the harder questions to answer.

Agile best practices suggest that we include test teams in the planning meeting. But how else can we involve them in the development process in productive ways? What testers and development teams need is a tool that can manage the test effort and link it to development efforts, all within the context of development sprints. We hope to convince you that it can be done in a manner that will lead to the magic power of the number three.

**Concurrent testing adopts testing throughout an iteration, concurrent with development. This prevents teams from compressing testing into a separate activity at the end of an iteration or release. Concurrent testing reinforces the concept of feature teams working in parallel.**

***Practice: Concurrent Testing***

### 2.3.1 Linking requirements and testing

With Quality Manager, test plans move from being dusty old documents to active, dynamic plans reflecting the current status of the test effort. To ease the transition from dusty to dynamic, a Quality Manager test plan is organized like a document with configurable Table of Contents that describes key aspects of the plan.

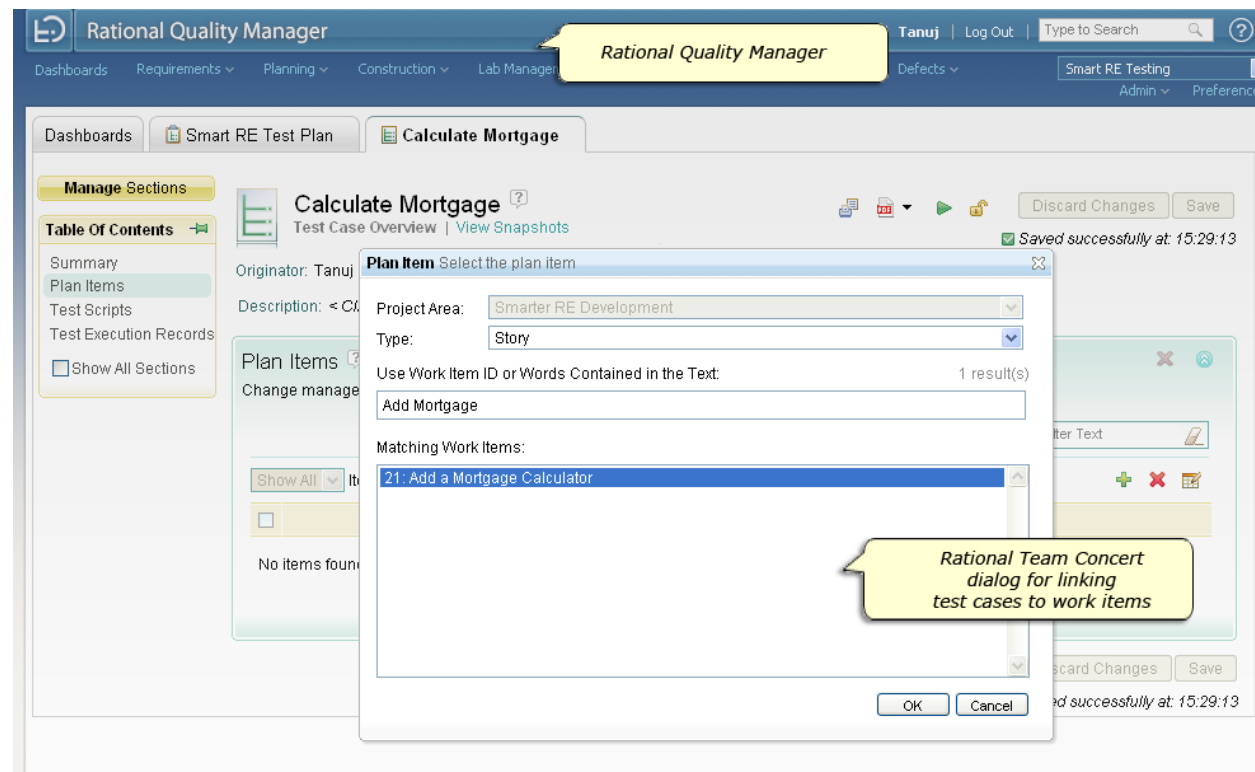
The plan can be as simple or complex as needed for any given project. For example, a test plan can have a section for linking to the requirements defined by Bob in Requirements Composer. Linking to requirements helps Tanuj understand what to test. A test plan consists of a list of test cases that the team will need to construct and execute. To align the test effort with the business needs, Tanuj ensures that every requirement has at least one test case. When creating these links, Tanuj has a similar in-context collaboration experience as Bob did when he linked his requirement to a developer work-item. When Tanuj chooses to “Link to” an existing requirement, a dialog appears within Quality Manager that contains the user interface and semantics for selecting requirements from Requirements Composer. Tanuj chooses the same requirement that Bob worked with in the first part of this scenario. Just as Bob did not have to leave Composer to create a link, Tanuj does not have to leave Quality Manager to create a link. Both users can collaborate in-context of the work they are completing.

### 2.3.2 Linking test cases with developer plan items

Now Tanuj knows what to test (by linking to requirements), but next he needs to know when to test. He can do this by linking test cases to development work-items in Rational Team Concert. Sure he was included in the Sprint Planning meeting and that helps, but plans are dynamic. To stay up to date with the development effort, he can link his test cases to work-items in the Rational Team Concert Sprint backlog without leaving the Quality Manager user interface.

Once again, notice the dialog that appears as illustrated in Figure 9. Tanuj is in Quality Manager, but the Team Concert user interface for selecting work items appears in the dialog. Tanuj searches the Team Concert repository, selects a work-item and links to it.

At any time, Tanuj can hover over the link to see the status of the work-item in Team Concert, which is shown in Figure 10. The status information is live and coming from Rational Team concert providing, Tanuj with real-time information about of the work-item, such as its status, owner, and when it is planned for implementation.



**FIGURE 9.** TANUJ LINKS TEST CASES WITH STORIES IN THE SPRINT BACKLOG

At the same time, when Scott uses Rational Team Concert, he can see which Sprint backlog items have links to test cases and which do not. On his team, stories are complete when testing is complete. Therefore, Scott and Tanuj can work together to ensure every Story has at least one test case. The link is also beneficial to development. It helps to determine whether the tests associated with the Story have passed and to assess the “Done” state of the story.

The screenshot shows the Rational Quality Manager interface. A story titled "Story 21: Add a Mortgage Calculator" is selected. A rich hover menu is displayed over the story, providing detailed information:

- Status:** New
- Summary:** Add a Mortgage Calculator
- Details:**
  - Type: Story
  - Filed Against: Backlog
  - Story Points: 8 pts
  - Progress: [Progress bar]
  - Team Area: Team 1 / Smarter RE Development
  - Creation Date: July 9, 2009 3:52 PM
  - Created By: Scott
  - Tags: [None]
  - Owned By: Unassigned
  - Priority: High
  - Planned For: Sprint 1 (1.0)
- Quick Information:**
  - Subscribers (1): S
  - Parent: 20
- Description:** Enable users to calculate mortgages directly from search results page, and from the property page. Calculate monthly payment based on size of loan, rate, term.

The background shows a "Calculate Mortgage" test case overview with a table of plan items. The table has columns for "ID" and "Story 21: Add a Mortgage Calculator".

**FIGURE 10.** RICH HOVERS PROVIDE INSIGHT INTO ARTIFACTS BEHIND THE LINK



### 2.3.3 Bob is curious about coverage

To demonstrate the power of the number three, Bob, working in Composer, can see which requirements have links to test cases. Bob also has a widget on his dashboard that shows him all requirements without a link to a test case in Quality Manager. This gives Bob immediate insight into the test coverage for his requirements. He has a similar widget that shows him all requirements without a link to a work-item in Team Concert. This lets Bob see the status of both the development and test teams enabling a rich collaboration across the teams.

### 2.3.4 There is magic in the number three.

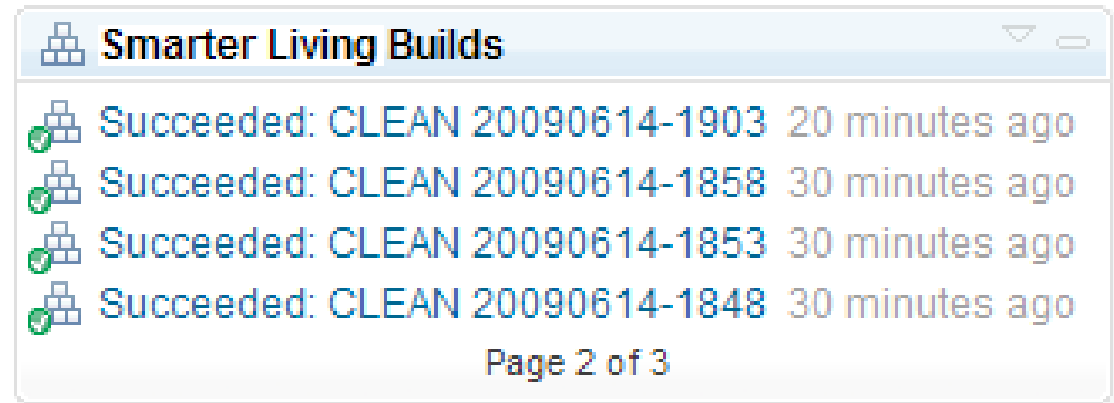
The requirements, development and test efforts are linked and aligned giving all members of the delivery team the ability to sprint as a 'whole team.' The increased transparency enables concurrent testing, and reduces wait time -- one of the obstacles of lean development.

This cross-discipline linking frees each team member to use a tool tailored to their needs. In addition, all members of the team can collaborate, in-context, with the toolset providing non-intrusive integration. It is very important to note that information is not copied, everyone is viewing the same, live, data and therefore each is viewing a real-time reflection of reality, not yesterday's or last year's data.

## 2.4 “The Build”:

### A present from an eccentric relative, or something I want?

You know which relative we’re talking about. The one that, year after year, sends the gift where you wonder, do I want to open this? To testers, the build coming from development can be like that present from the eccentric relative. The perpetual question for testers is, “is the build ready for test?” This is quickly followed by “What features are implemented?” and “What defects are fixed?” Hopefully we’ll persuade you that builds can be presents worth receiving..

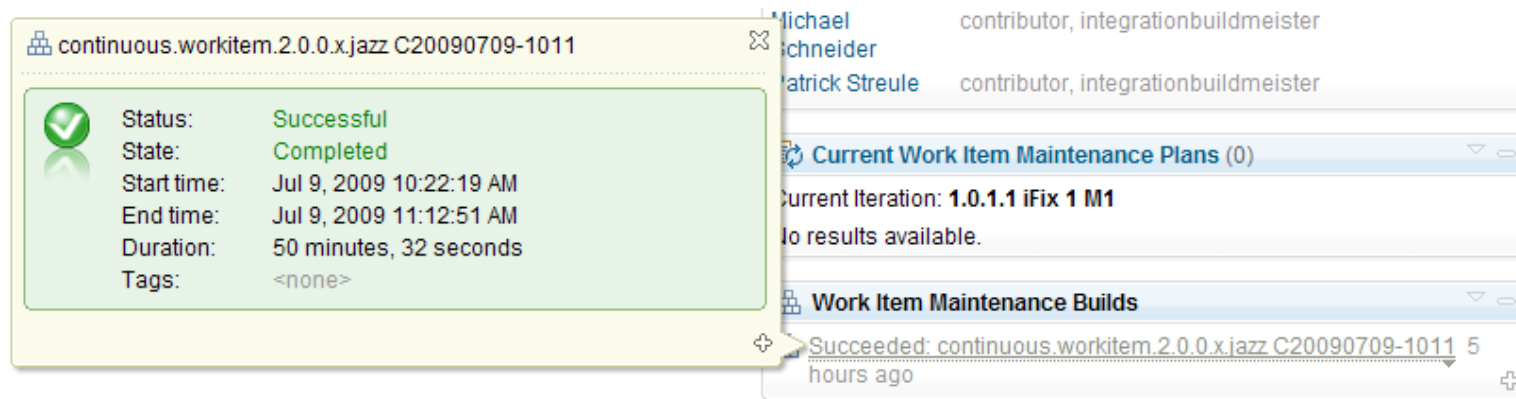


**FIGURE 11.** A BUILD FEED WIDGET IN RATIONAL QUALITY MANAGER

### 2.4.1 The build bridges development and test

Tanuj uses widgets on his dashboard to monitor the status of the builds in Team Concert. Figure 11 shows a feed of Team Concert builds.

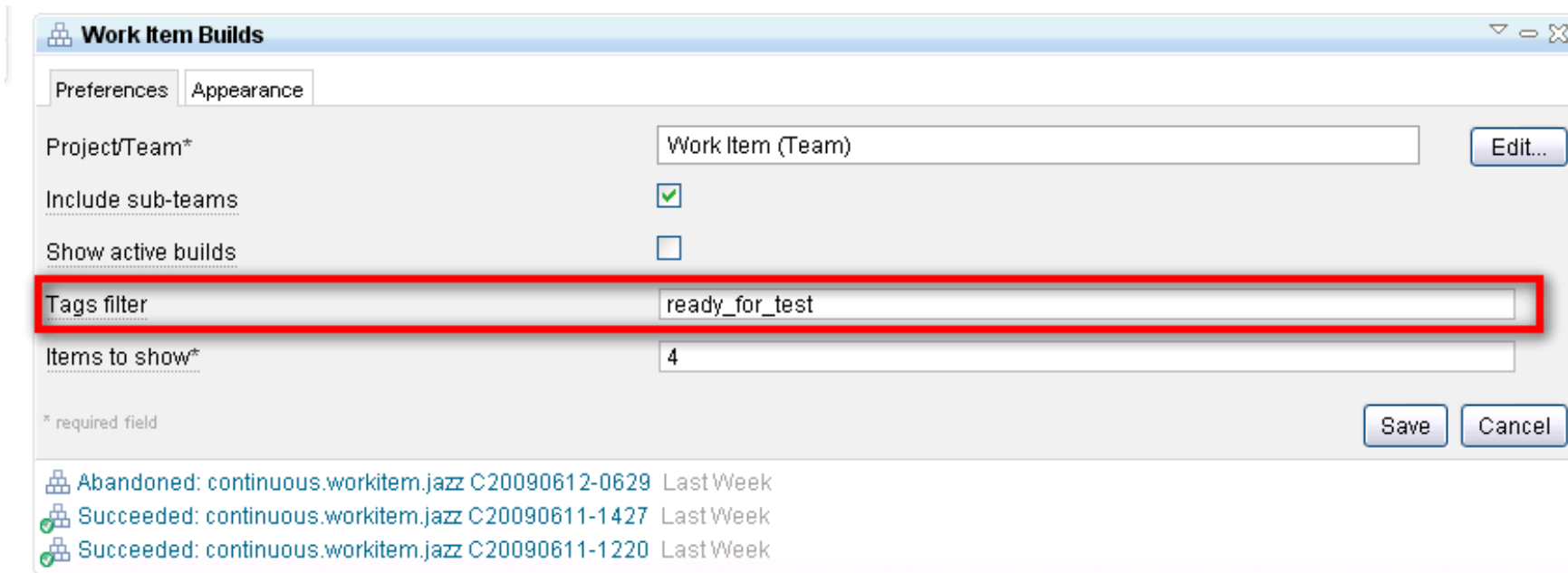
At any time Tanuj can click on the link to view the contents of the build or use the rich hover which is illustrated in Figure 12.



**FIGURE 12.** RICH HOVER SHOWING THE BUILD STATUS FROM A DASHBOARD WIDGET

The build results are presented in a web user interface. Here Tanuj can see what changes went into the build thus helping him determine what to test.

Figure 13 shows another form of in-context collaboration, where developers tag a build ready for test and testers can filter the builds to those builds that have this tag.



The screenshot shows a 'Work Item Builds' dialog box with the following fields and values:

- Project/Team\*: Work Item (Team) [Edit...]
- Include sub-teams:
- Show active builds:
- Tags filter: ready\_for\_test (highlighted with a red box)
- Items to show\*: 4

Buttons: Save, Cancel

\* required field

Build list:

- Abandoned: continuous.workitem.jazz C20090612-0629 Last Week
- Succeeded: continuous.workitem.jazz C20090611-1427 Last Week
- Succeeded: continuous.workitem.jazz C20090611-1220 Last Week

**FIGURE 13. FILTERING BUILDS USING TAGS**

In addition, Quality Manager provides a feature for running a 'test sequence' when a build completes. The test sequence can be used to automate traditionally manual tasks such as test environment setup or tear down, or test case or suite execution. The sequence is automated, captures an audit log, and can be triggered by the completion of a build running in Team Concert. This automation removes the manual and error prone process of deploying a build into a test lab. Just as agile teams have automated the build process, testers can now automate the deployment of builds into the test lab.

## 2.4.2 The build is package you can look forward to receiving

By viewing the build as a bridge between development and test, we begin to provide clarity into the status, contents and quality of the build. Providing this clarity is fundamental to developer and tester relations. Agile teams promote the idea of automating everything. By adding test suites to the test sequence testers can finally automate smoke and regression tests, thus freeing themselves to spend more time on exploratory testing. In doing so, the build becomes a package the testers are eager to receive. But once the build is deployed, what happens when the testers begin to test?

## 2.5 Defects: Odd game, fiction or fact?

Hopefully by now you've had the unique opportunity to play the game 'whack-a-mole.' It's kind of like finding and fixing defects, wouldn't you say? Upon defect discovery, a tester logs a defect report including a detailed set of steps needed to recreate the defect. In turn, the developers can't recreate the defect. Despite our efforts to improve defect reports and reproducibility it remains a challenge that creates more friction than lean, mean software delivery machines need.

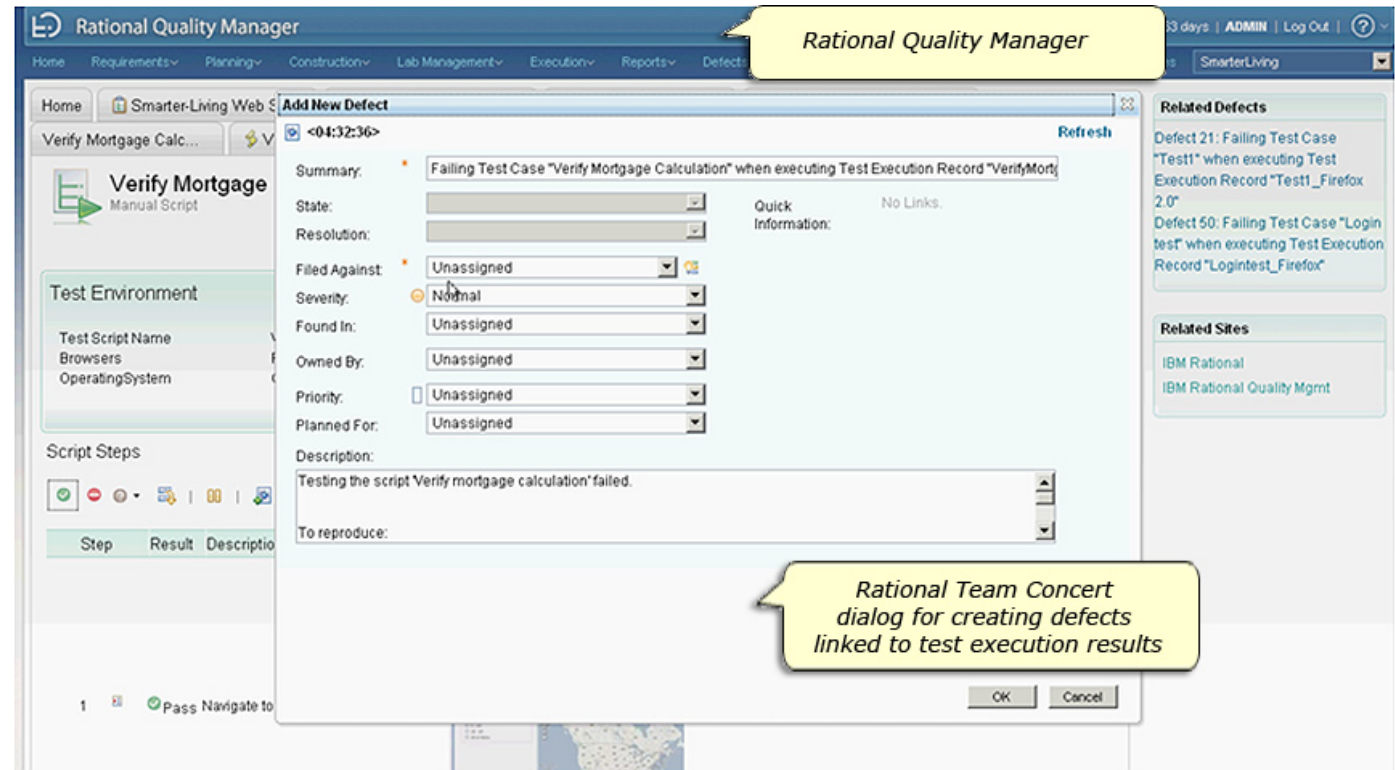
When a tester executes a test, the results are captured by Quality Manager, which is very important information to the testers. But the key question is how can you share it with the developers when reporting defects? Next we'll demonstrate why finding and fixing a defect doesn't have to feel like playing 'whack-a-mole.'

### 2.5.1 Finding and submitting a defect

In Rational Quality Manager, a test case has a test script, which can be a manual set of steps, or an automated test script which agile teams prefer. For each test case there may be one or more test execution records. A test execution record binds a script with a particular test environment. For example, conducting the test using three different browsers would constitute three test execution records.

The approach to submitting defects is similar to those that we demonstrated as Bob linked requirements to development work-items. Tanuj can create a new defect or link to an existing defect managed by Team Concert. The OSLC change management service is called and presents the user interface in a dialog with which Tanuj interacts.

As shown in Figure 14, the dialog is primed with work-items of type defect. When Tanuj clicks ok, a defect is linked to the exact step in the test script where the failure occurred. In addition, the defect in Team Concert has a link back to the execution result in Quality Manager. At any time, Testers can navigate the link to view the defect. Submitting the defect required only 3 clicks. 1 to open the dialog, 1 to set the “Filed Against” attribute, and 1 to click ok. When compared to the task of opening a separate tool, logging in, and laboriously detailing every step performed to reproduce the defect, the testers experience is profoundly improved!



**FIGURE 14.** USING QUALITY MANAGER TO CREATE A DEFECT IN TEAM CONCERT

## 2.5.2 Knowing when the test team is blocked

When the defect is submitted by Tanuj, Scott can notice it from several locations in Rational Team Concert: the events view, the recently submitted query, or using a new widget which shows “Defects blocking Test.” This widget presents the results of a C/ALM query that leverages the links between Rational Team Concert and Quality Manager. The query presents all open defects in Team Concert with “Blocked by” link type. In this case, Scott can see which defects are blocking the test team, and triage them appropriately. By acting on them immediately, Scott can reduce the wait time for testers and ensure the highest quality and team productivity.

The screenshot shows a Rational Team Concert dashboard with a 'Testing' tab. A widget titled 'Defects blocking tests (1)' is active, displaying a table with the following data:

Status	Resolution	Summary
New	Defect	Failing Test Case "Login test" when executing Test Execution Record "Logintest_Firefox"

The 'Details' section for the selected defect includes:

- Type: Defect
- Filed Against: Backlog
- Severity: Normal
- Found In: Unassigned
- Team Area: Smarter Living Team / SmarterLiving
- Creation Date: May 29, 2009 1:13 AM
- Created By: Scott
- Tags: Unassigned
- Owned By: Unassigned
- Priority: Unassigned
- Planned For: Unassigned
- Estimate: Unassigned
- Time Spent: Unassigned
- Due Date: Unassigned

The 'Quick Information' section shows: Subscribers (1): S, Blocks Test (1), and Affects Test (1). The 'Description' field contains: Testing the script 'Invalid login test' failed.

A callout box points to the tooltip with the text: *Rational Team Concert dashboard with a C/ALM query and rich hover*

**FIGURE 15.** DEFECTS BLOCKING TESTS WIDGET, AND A RICH HOVER OVER A LINK



### 2.5.3 Recreating the defect

Deb is a member of Scott's team and is responsible for fixing the defect. When she receives the defect, she has a rich description of the test which was automatically provided by Rational Quality Manager via the integration. She also sees there's a link to the execution result. Figure 19 illustrates the use of the Affects and Blocks link types.

There is another nice collaboration at work here, when a tester files a defect against a particular build, a developer can easily reconstruct the source set-up that matches the state of the tester. When submitting a defect, Tanuj sets the Found In field as shown in Figure 16.

Next Deb opens the build result, and from the build result traverses the link to the snapshot, as shown in Figure 17.

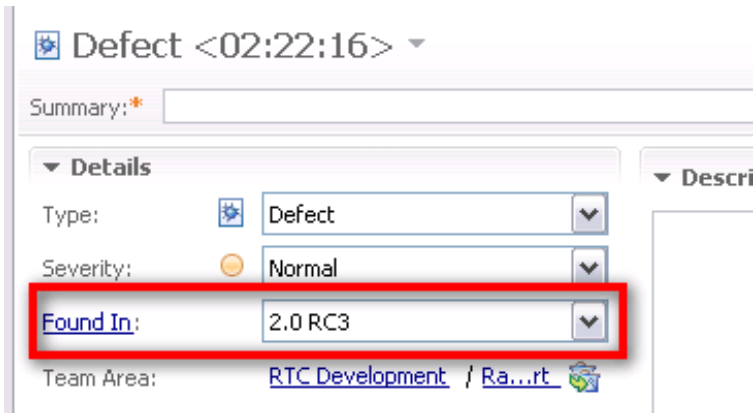


FIGURE 16. FILING DEFECT AGAINST A MILESTONE BUILD

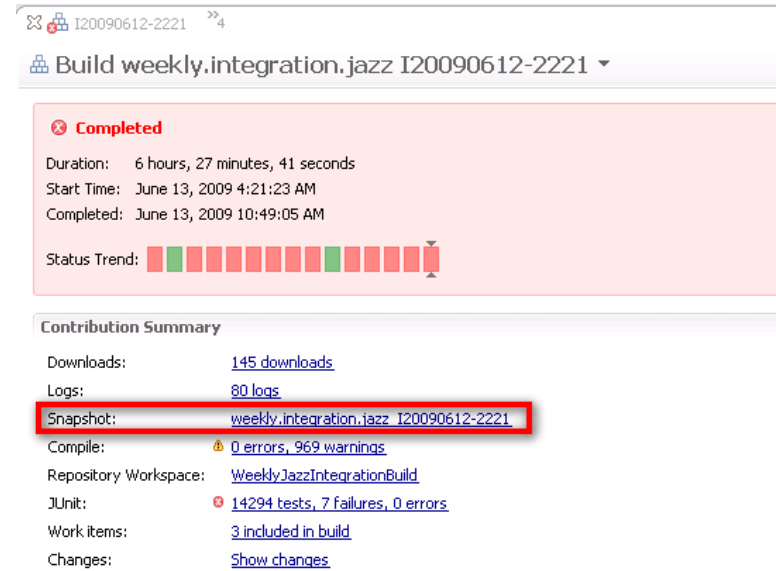
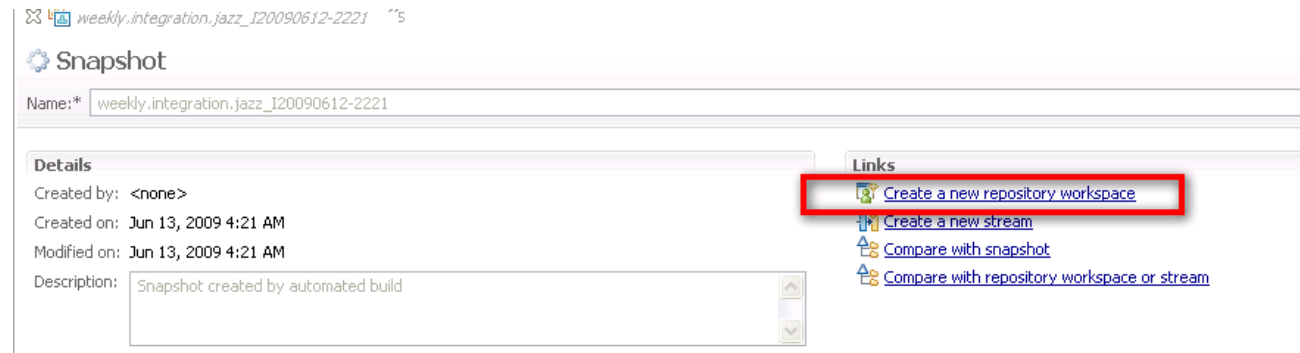


FIGURE 17. LINK TO SOFIGURE 17URCE SNAPSHOT

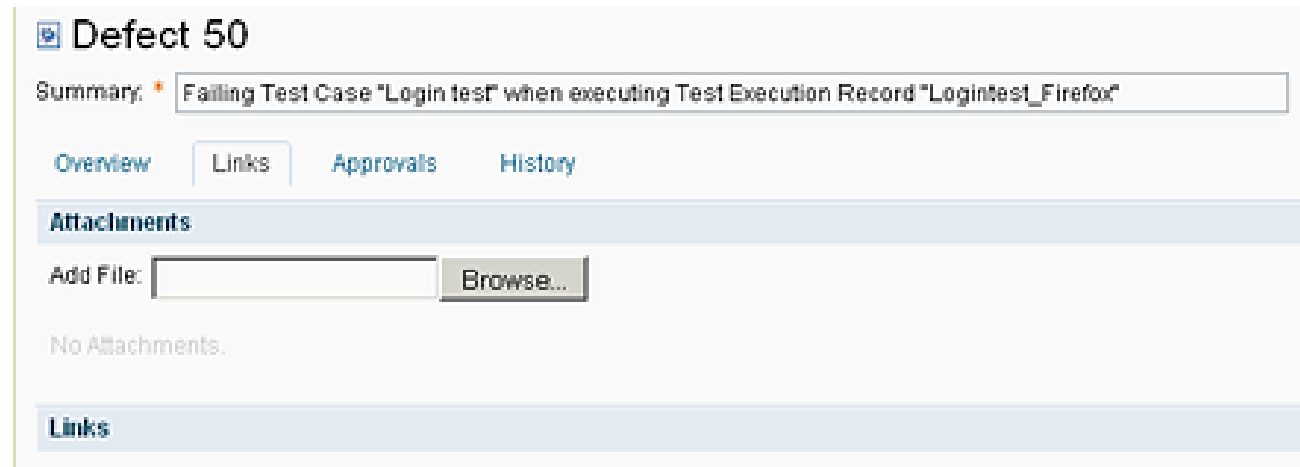
On the snapshot user interface, Deb clicks the link to create a workspace, which is highlighted in Figure 18.

Once her workspace is created, Deb is ready to recreate the defect. She reviews the defect report, and if needed can traverse the link to view the execution result, test script and test case. The link is shown in Figure 19

By traversing the link, Deb can see exactly what happened and can use the same steps to recreate and debug the problem. This removes any doubt as to how to recreate the defect and contributes to overall developer productivity. When the defect is fixed, Deb delivers the code and marks the defect as resolved.



**FIGURE 18. CREATE A WORKSPACE FROM A SNAPSHOT**



**FIGURE 19. A DEFECT WITH LINKS TO TEST EXECUTION RESULTS**

## 2.5.4 Knowing when defects are fixed

In this team, the process set-up in Team Concert defines that only testers can mark a defect 'verified.' Therefore, Tanuj leverages a cross-repository widget on his dashboard to show which of the defects he submitted are marked as resolved. Alternatively, hovering over the link in the execution result provides up-to-date information about the status of the defect. When the build containing the fix is deployed to the test lab, Tanuj can re-run the test execution record for that defect, and update the defect accordingly.

## 2.5.5 Recreate defects from fact

When a developer reviews the defect, the link to the test execution is provided. The developer can navigate the link and view the very same script the tester used in finding the defect. If the test is a manual test, the defect is linked at the exact step where it occurred in the script. This lets the developer review the exact same steps the tester used to recreate the defect. This reduces the frustration felt by both developers and testers when defects cannot be reproduced.

In addition, the testers and the developers need to keep track of the status of the defects and testing effort. To help them, new widgets are provided in Team Concert and Quality Manager. The testers using Quality Manager can see which Tests are blocked by Defects. Developers using Team Concert can add a widget to their dashboard that shows "Defects blocking Test." At a glance, developers can see new defects submitted by the test team and respond accordingly.

Imagine no more countless hours training testers on how to create a good defect report. No more wasted hours attempting to recreate a defect. No more wondering whether the test team is blocked, or when defects will be fixed. Defect handling can be grounded in reproducible fact.

## 2.6 Sprint Review: Rorschach test or view of Reality

By linking the business, development and testing artifacts, teams can align on sprint backlogs and sprint execution. By managing plans and execution results teams have an up-to-date reflection of reality to help them make decisions throughout the sprint. The combination of using links across artifacts, rich hovers, and cross-repository widgets hosted in user dashboards provides unprecedented team awareness.

The links between requirements, development and testing establish transparency and agreement across these two traditional silos. By seeing that the requirement is linked to a work-item on a development plan, and hovering over the link to see the status of the work-item, product owners become active participants on the team. By seeing what defects are blocking test, development leads can actively triage defects to keep the test team productive. If change to any of the artifacts is necessary, the impact can be noticed as it is being considered, and the agreement can be collaboratively revised – because the link is in place. The traceability between requirement, development and testing can help to reduce thrash, build trust, and establish a shared vision for execution.

This live data helps the team understand when they are done, done, done. They can compare their result to their sprint goals, confirm that all requirements are implemented and tested, and defects are fixed.

## Chapter 3 *The WWW (Web We Wove)*

- 3.1 The web Bob wove
- 3.2 The web Scott and his team wove
- 3.3 The web Tanuj wove
- 3.4 The secret web of a defect
- 3.5 How it's woven

### 3 The WWW (Web We Wove)

The story we told is simple but powerful. It may help if we summarize the links created by each of the team members.

#### 3.1 The web Bob wove

Without leaving the Composer user interface, Bob linked his requirement to a "Story" work-item in Rational Team Concert, thus establishing a relationship between them. The image in Figure 20 illustrates the link types between resources: the work-item implements the requirement; the requirement is implemented by the work-item.

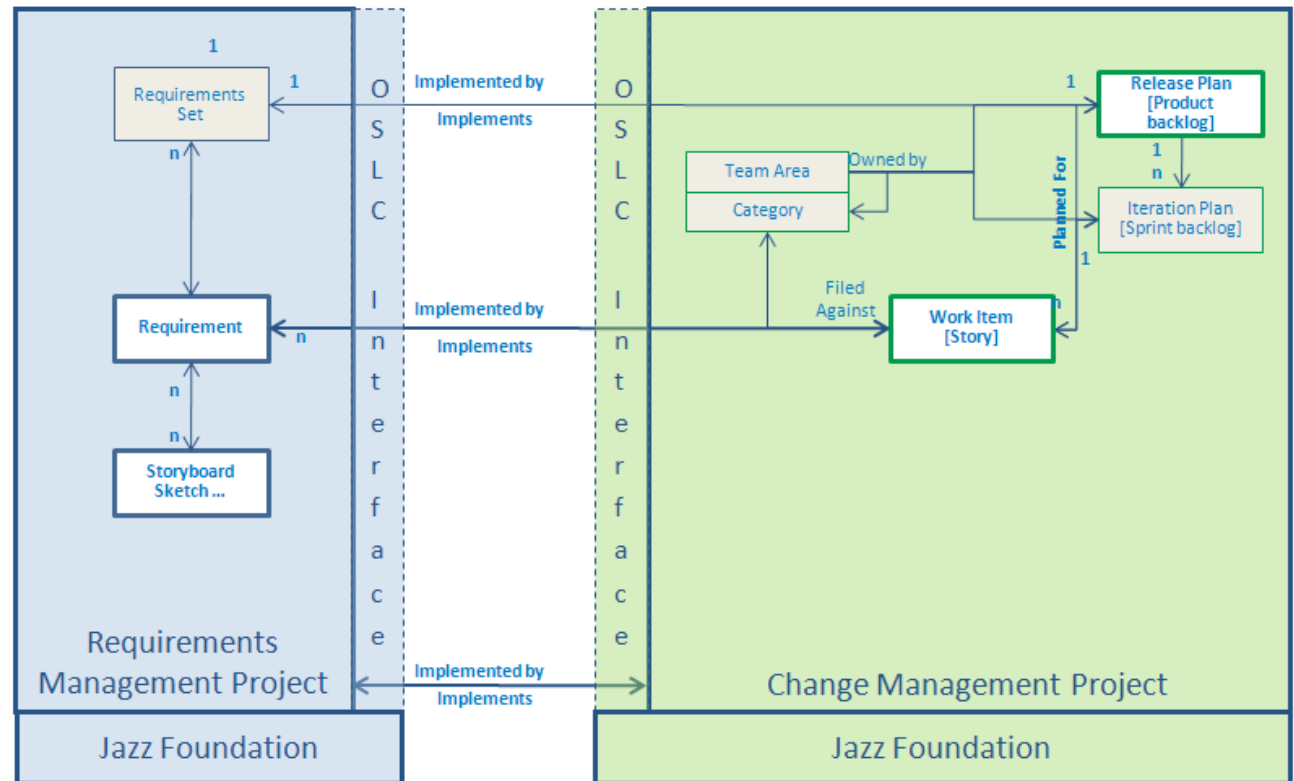


FIGURE 20. A REQUIREMENT IS LINKED TO A WORK-ITEM ACROSS DEVELOPMENT TOOLS

### 3.2 The web Scott and his team wove

Scott's team navigated the links that Bob created between the requirements and stories. By viewing Bob's sketches and storyboards, they had clearer insight into the amount of work involved. Here, Bob's requirements were treated like a resource at the end of a URL. The development team simply navigated the links and viewed the requirements.

As shown in Figure 21, Scott and his team worked with the Product and Sprint backlogs, and decomposed story work-items into Tasks. These are all local to Rational Team Concert.

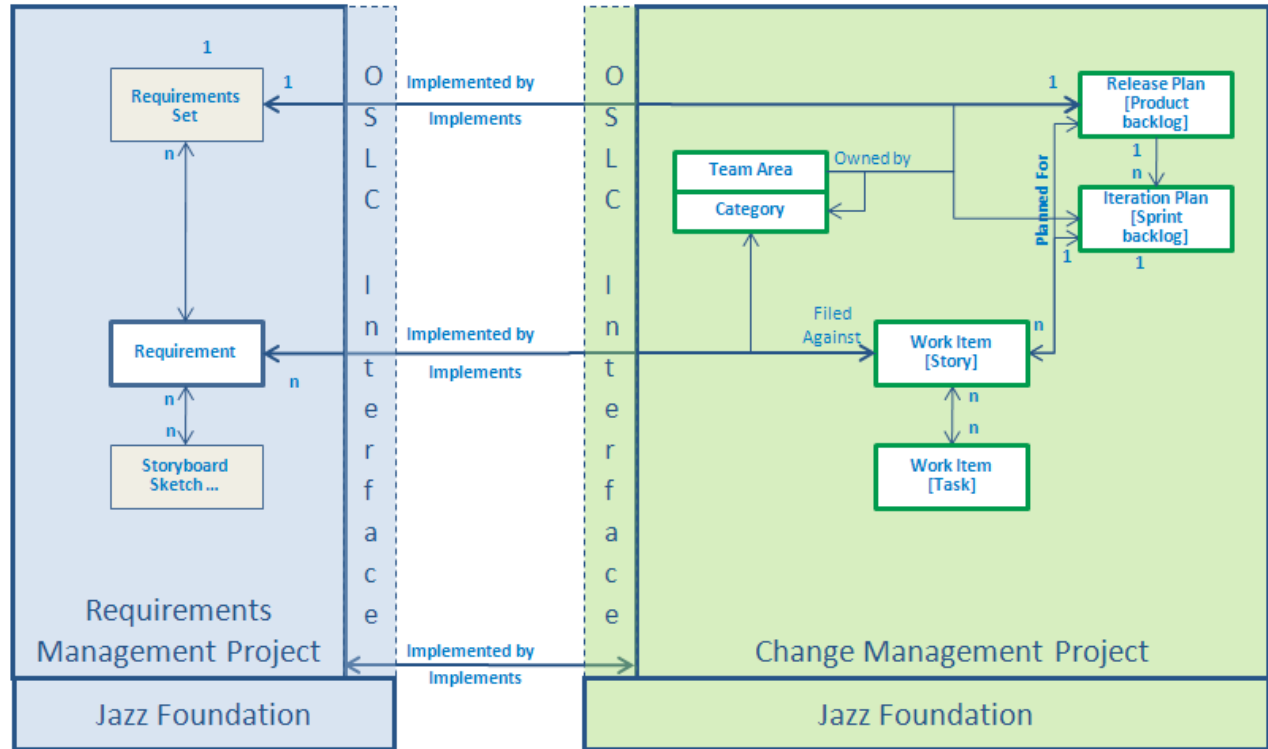


FIGURE 21. ARTIFACTS LINKS WITHIN AND ACROSS TOOL BOUNDARIES

### 3.3 The web Tanuj

#### wove

Tanuj creates test cases as part of the test plan, and has links to requirements to provide clarity on what to test. Test cases also link to the planning-level work-items providing clarity on when the features will be available to test. Tanuj uses a tool tailored for his discipline and also has unprecedented alignment with the product owner and development team.

Figure 22 illustrates how a single test case can have a “test” link to a work-item in Team Concert, and a ‘validates’ link to a requirement in Composer. This gives the tester immediate access to the “what” (requirement) and the “when” (plan-item). In addition, Rational Quality Manager provides a Quality Management service that other OSLC compatible products can consume. In the scenario we showed the tester linking to the requirement; however the inverse can also occur, where Bob, the product owner links a requirement to a test case.

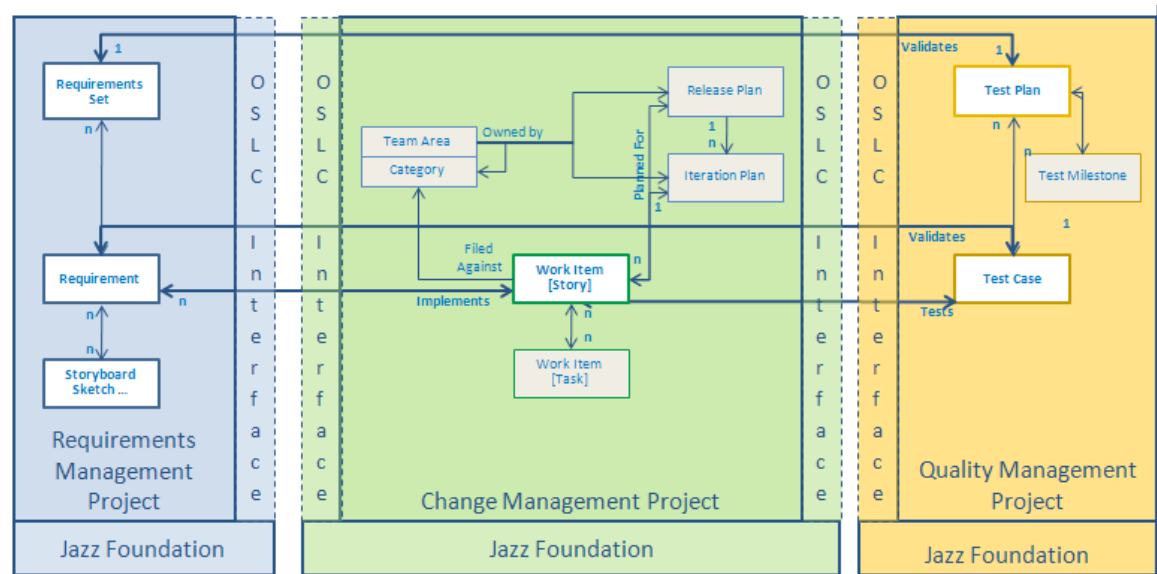


FIGURE 22. TESTING ALIGNS WITH REQUIREMENTS AND DEVELOPMENT



### 3.4 The secret web of a defect

In Quality Manager 2.0, a tester can link an execution result to a defect report managed in Team Concert. Quality Manager automatically populates the fields in the defect report with information from the test execution. The execution result has a link to the defect, the defect has a link to the execution result.

The “Affects” and “Blocks” link types indicate the severity of a defect which helps the development team when triaging defects. Figure 23 shows the relationships between test execution and defects.

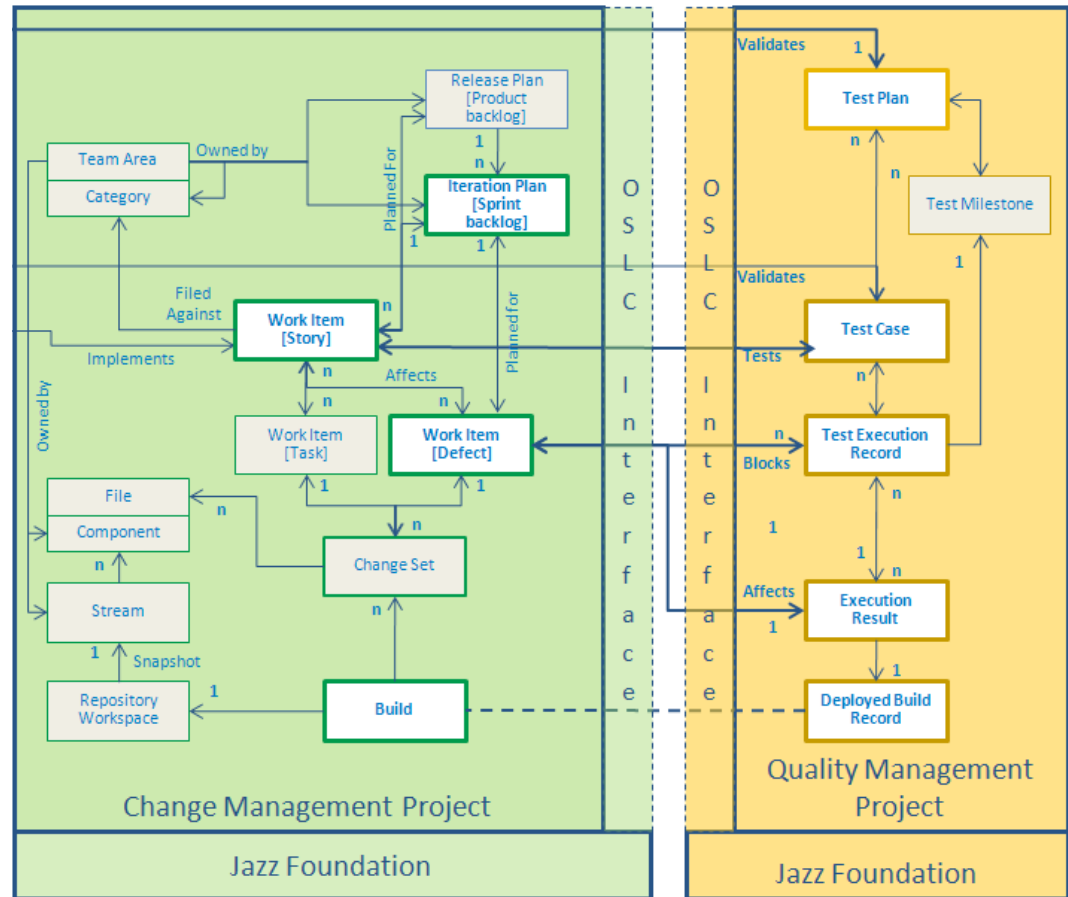


FIGURE 23. TEST EXECUTION RESULTS LINK TO DEFECTS

## 3.5 How it's woven

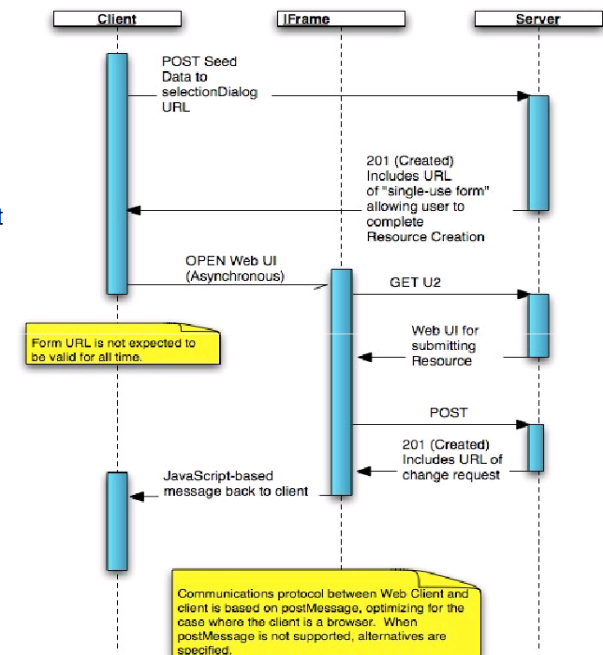
Behind the scenes the Jazz Foundation and Open Services for Lifecycle Collaboration are quietly at work providing a rich user experience. The Jazz Foundation provides the common web framework, dashboards with the ability to host cross-repository widgets, rich hovers and link types for easily recognizing the related artifacts.

Open Services for Lifecycle Collaboration provides open, public descriptions of resources and interfaces for sharing artifacts across the software lifecycle. The scenario demonstrated implementations of the completed change management specification. The scenario also provided previews what's to come with the requirements management and quality management specifications. In the scenario, the term artifact is used to describe the content the users interact with. In the programmable web, this is called a resource. The strategy treats all development artifacts as resources at the end of the URI, where both XML and JSON are supported formats. RESTful interfaces are used to GET, PUT, POST or DELETE data in each of the repositories. Each tool implements the OSLC specification for their domain (e.g. change management, requirements management, quality management)

In the example, when Bob chooses to link to, or create an artifact in Rational Team Concert, Requirements Composer calls the change management service using the public [OSLC change management specification](#). Per the specification (Figure 24), Composer delegates the user interface to Rational Team Concert, meaning, the details and semantics provided by Team Concert are consumed by and displayed in a Composer dialog. When Bob clicks OK, the Composer sends a PUT request to the Team Concert change management service. Links are created between both artifacts using link support provided by the Jazz Foundation. This same strategy is employed when Tanuj chooses to link to a work-item in Team Concert; Quality Manager calls the change management service using the OSLC specification.

### OSLC CM 1.0 – Delegation

- Resource Selection
  - ▶ A way to pick (search for and select
- Resource Creation
  - ▶ Use POSTed data to seed a Web UI, allowing delegated, user-attended creation of change requests from a loosely coupled client
- Creation and Selection rely on a simple JavaScript API to return URLs to the caller



**FIGURE 24.** OSLC DELEGATION

This provides a powerful and resilient integration which supports independent evolution of the products. The URL integrating the two tools can stay the same with each new upgrade, and changes can be made to the user interface (coming from the provider application) without compromising the consumer's integration. The delegating UI is helpful to the end user but it is even more helpful to developer. Without delegating the UI the product developers would have to have in depth knowledge about how to create a particular artifact. For example, creating a defect would require knowing which of the attributes are required and this would require knowledge about the used process and so on. This would increase the coupling between the products and increased coupling has a negative impact on independent evolution. This demonstrates two important architectural decisions to delegate complex capabilities to the provider and provide a simple way to discover a resource. Delegating the UI results in a coarse grained coupling only. Another benefit of this approach is that it keeps the barrier of entry low for existing products.

## Chapter 4

# ALM Ecosystems – one size fits no-one, or haute couture?

We provided a single scenario that aligned business, development and testing efforts in a consistently open way. We also asserted that this is not the only answer. Let's look at a different set of tools and a new scenario. In this scenario a large enterprise has standardized on Rational ClearQuest as their change management system. They use Rational DOORS for Requirements. One of development teams uses Rational Team Concert integrated with a different SCM system (SVN). The testing team uses Rational Quality Manager. This scenario uses a mix of OSLC and traditional integrations and is shown in Figure 25.

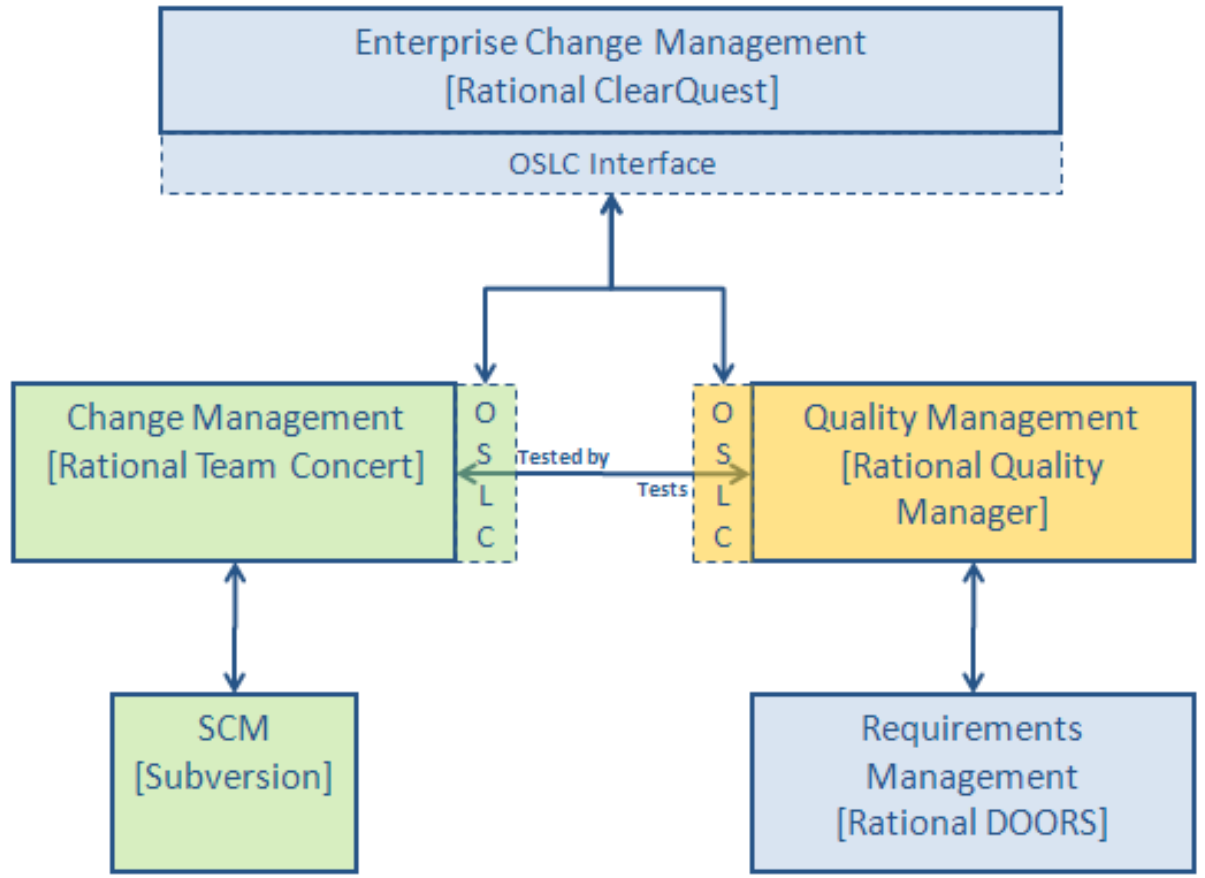


FIGURE 25. AN ALTERNATE SCENARIO USING OSLC AND TRADITIONAL INTEGRATIONS

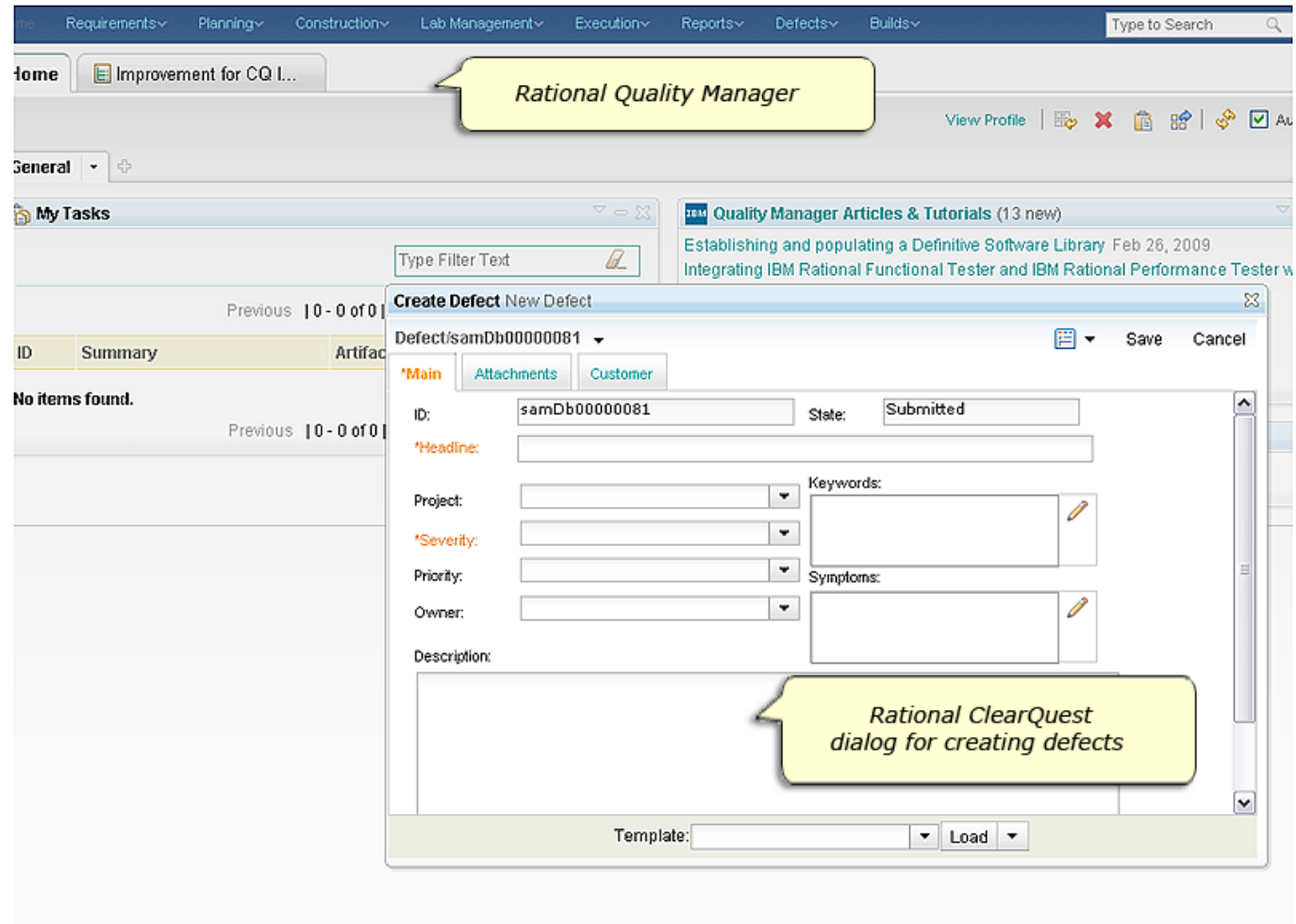
The Rational DOORS (and Rational RequisitePro) teams plans to contribute to the OSLC specification for Requirements Management, and at some point will consume the OSLC interfaces for Change Management.

This time let's start with the testers. Linking requirements and tests is a highly valued and commonly requested feature of any software development management tool. In a distributed collaborative environment this ability is an essential component of any automated tool. DOORS and RequisitePro are two products in the Rational stable that are moving toward OSLC support, that also provide integrations with Rational Quality Manager. But yet again we remind the reader that as additional products and vendors add support for the OSLC interfaces, the possibilities begin to open for any requirements management vendor.

The testers use Rational Quality Manager integrated with DOORS for Requirements Management. Testers can align their test effort with the requirements specified in DOORS by linking requirements to test plans and test cases.

When defects are found, they are submitted to the Enterprise Change Management system, which in this case is Rational ClearQuest. Because ClearQuest also supports the OSLC Change Management specification, the user experience remains familiar for the tester.

As shown in the next figure (Figure 26), Quality Manager calls the change management service provided by ClearQuest, and the ClearQuest user interface appears in a dialog hosted by Quality Manager. Testers can enter information about the defect and submit it to ClearQuest without leaving the Quality Manager user interface.

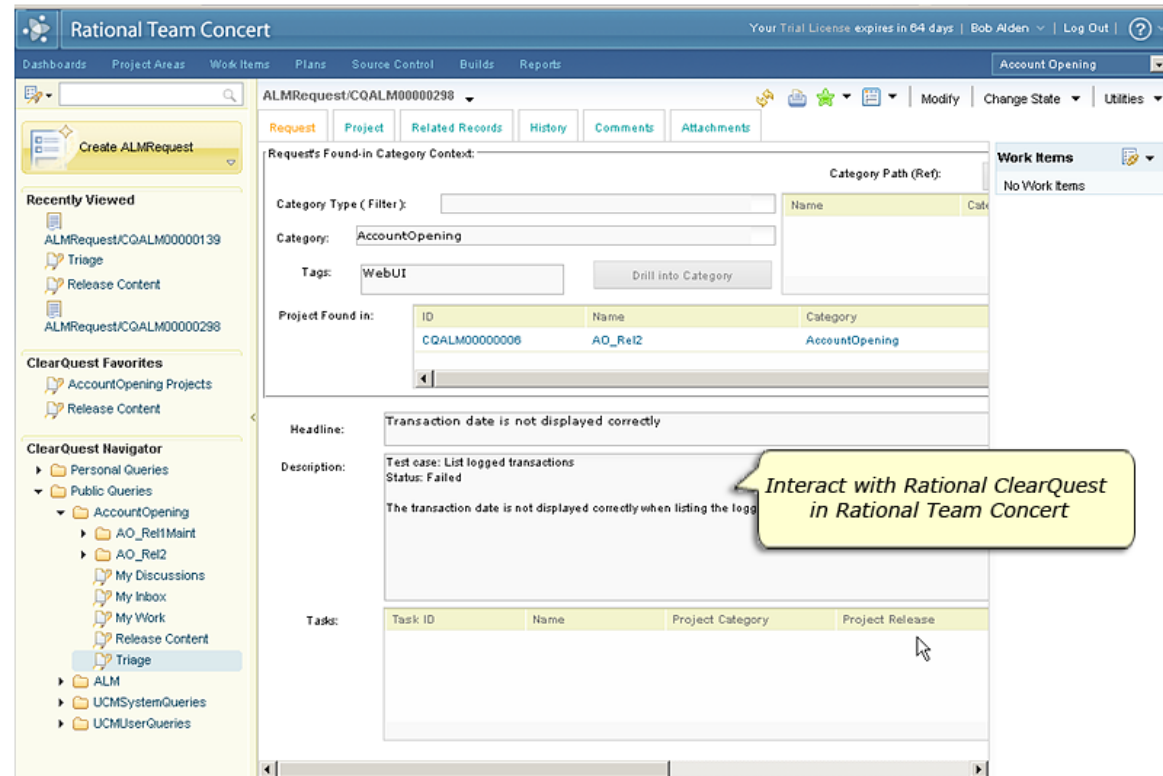


**FIGURE 26.** QUALITY MANAGER USES THE CLEARQUEST CHANGE MANAGEMENT PROVIDER



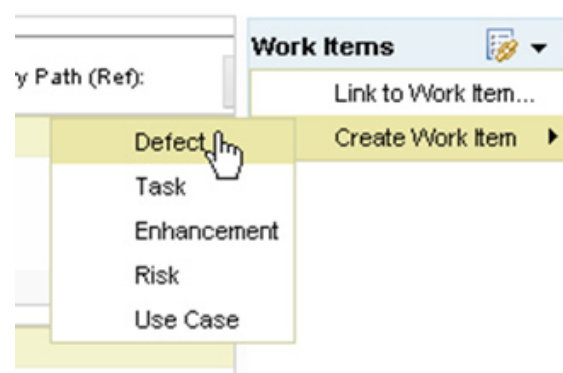
Users of ClearQuest triage all incoming defects. The first triaged defect belongs to the development team that uses Team Concert. The challenge is communicating and tracking a defect managed by ClearQuest but assigned to a team using Team Concert.

Using OSLC and the ClearQuest bridge to Team concert, users can link ClearQuest records to Team Concert work-items. Figure 27 shows the ClearQuest bridge in action. Notice the ClearQuest queries are available in the left hand navigation bar, and a full view of the ClearQuest record is presented on the right..



**FIGURE 27.** VIEWING A CLEARQUEST RECORD IN TEAM CONCERT USING THE CLEAR-QUEST BRIDGE

Any ClearQuest record can be linked to a Team Concert work-item. Figure 28 shows the menu that appears in the upper right corner of the ClearQuest record from the previous figure. The user interface is delegated to Team Concert and the 'Create Defect' dialog is presented, in-context to the user. Setting the required fields and clicking OK links the ClearQuest and Team Concert records.



**FIGURE 28.** LINK TO OR CREATE TEAM CONCERT WORK-ITEM LINKS TO CLEAR-QUEST RECORDS

Figure 29 shows the Team Concert work-item with a link to the ClearQuest record. Team concert users act on work-items like they always do, and when needed, can traverse the link to view the original ClearQuest record or use the rich hover with little interruption.

## Defect 53

Summary: \* Transaction date is not displayed correctly

Overview Links Approvals History

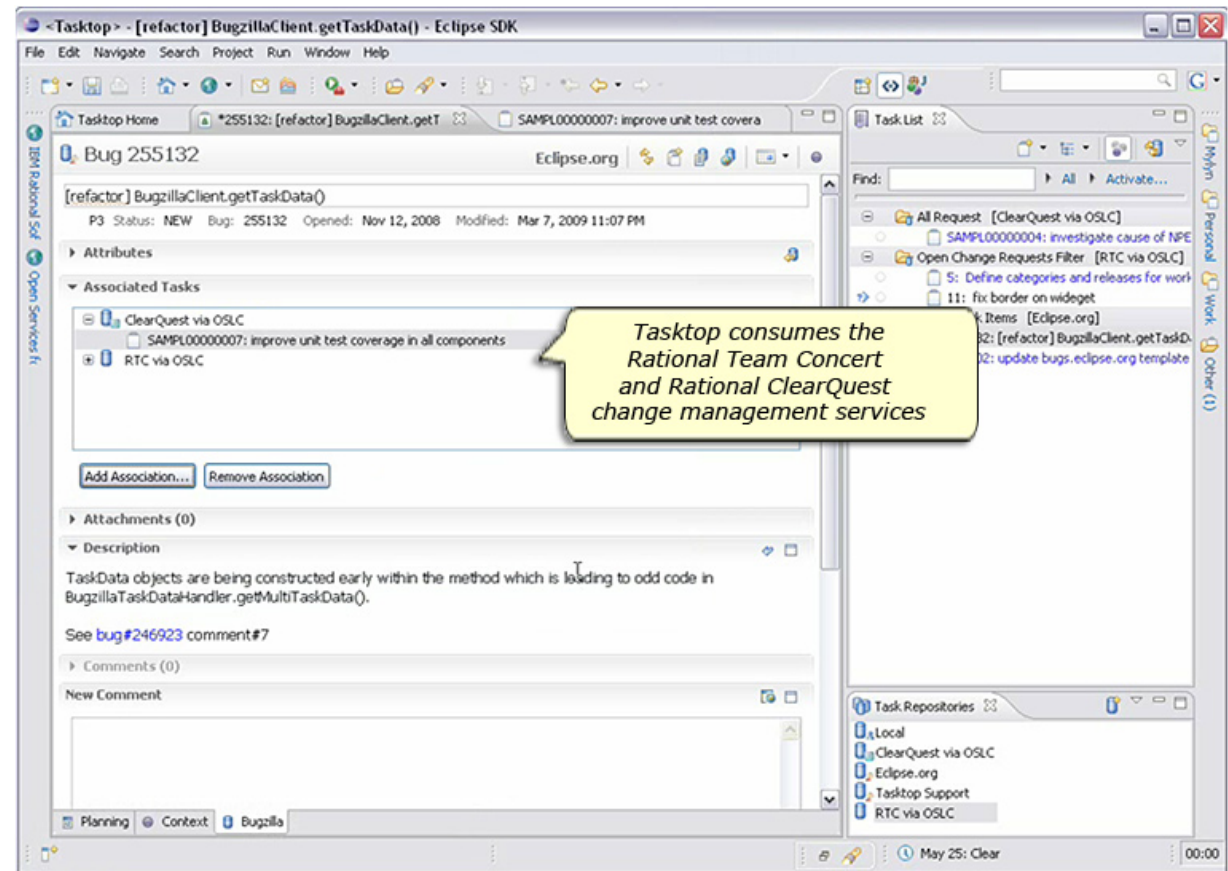
### Attachments

Add File:

**FIGURE 29.** THE CLEARQUEST RECORD AND TEAM CONCERT WORK-ITEM ARE LINKED

Another example is Tasktop's Mylyn. Mylyn supports task-focused programming. With Mylyn users can select the task they are focused on in their work from a change management system. Mylyn leverages OSLC to interface with different change management systems. OSLC helps Tasktop to reduce the number of integrations they have to provide for Mylyn. In (Figure 30) you can see how Mylyn connects to different change management systems. It accesses to RTC work items and ClearQuest records by consuming their OSLC change management services.

By using traditional integrations in combination with the OSLC specifications for requirements, change or quality management, any vendor can provide open mechanisms for linking resources across silos.



**FIGURE 30.** TASKTOP INTEGRATES WITH CLEARQUEST AND TEAM CONCERT VIA OSLC

# Open Sesame

*A phrase uttered by a genie from a bottle, or IBM Rational strategy?*

## Chapter 5

### Open Sesame:

A phrase uttered by a genie from a bottle,  
or IBM Rational strategy?

- 5.1 OSLC facilitates communication, behind the scenes
- 5.2 The Jazz Foundation reduces total cost of ownership
- 5.3 Jazz.net: A open, innovative, collaborative, community

## 5 Open Sesame: A phrase uttered by a genie from a bottle, or IBM Rational strategy?

Software delivery teams that are considering consolidating on a single tool must ask whether that tool will honestly suit the needs of every constituent in the lifecycle over the long run. We believe that no vendor could possibly do that, and few if any software development teams would want to. First, a tremendous amount of information already exists in development repositories throughout the enterprise. Second, each discipline and each team has its own culture and has probably settled on their tool of choice. Third, as you cross business units, the decision making authority is spread across organizational lines. Under these circumstances it is nearly impossible to get all groups to agree on, and become trained on a single tool. Last, every delivery team in every enterprise works with a different set of tools, new tools come to market at a rapid pace, and there are always requests to integrate some other tool, whether purchased or home grown.

Perhaps an absurd comparison will help make the case. Would you expect all information on the world wide web to be consolidated into a single repository? Of course not! But what you do expect is to be able to navigate the web to access and view the information regardless of where in the world it is stored.

We believe the software delivery ecosystem should be treated in the same way. The data can be housed anywhere. But what you expect is the ability to link, navigate and track artifacts regardless of where they are stored. The goal is for software development tools to become viewers and editors of commonly used data - to open the silos between the business, development and test tools. This belief led us to launch the Jazz project.

We started this eBook by introducing OSLC, Jazz Integration Architecture, Jazz Foundation and C/ALM scenarios. The scenario presented is an example of C/ALM scenario, and we hope you can see the value of using an 'outside-in' approach to driving C/ALM integrations. The integrations in the scenario are powered by OSLC implementations and the Jazz Foundation.

## 5.1 OSLC facilitates communication, behind the scenes

The community at Open Services for Lifecycle Collaboration is working to provide open public descriptions of development artifacts (resources) and the interfaces for sharing information across the development lifecycle. We saw examples of OSLC providers and consumers throughout this eBook, but that was just the beginning.

Imagine building a software delivery ecosystem with providers of services where all you needed to add is a single URL to consume the service, and your users could create and navigate a Collaborative ALM web of artifacts! It's a tremendously powerful opportunity. Calls for participation are listed on the open-services.net web site (Figure 31), and we invite everyone to join.

Home About Community Wiki Learn

### open-services.net

Open Services for Lifecycle Collaboration (also known as OSLC or Open Services) is a community effort to help software delivery teams by making it easier to use lifecycle tools in combination. The OSLC community is creating open, public descriptions of resources and interfaces for sharing the things that software delivery teams rely on, like change requests, test cases, defects, requirements and user stories.

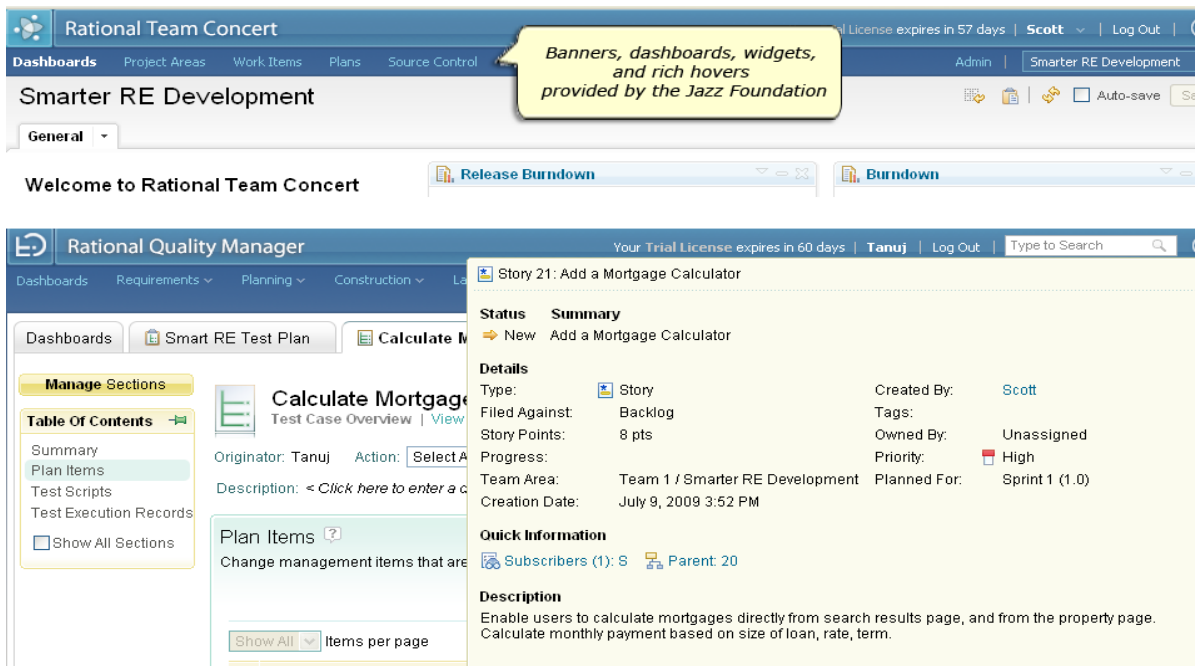
*OSLC is focused on interoperability across technology implementations - a challenge best addressed through common specifications and standards.*

By agreeing on common specifications for lifecycle resources and the services to access them, we can eliminate traditional barriers between tools and open the door to new forms of collaboration. OSLC can bring value to software delivery teams and tool providers alike, from the most Agile to the most ceremonial of projects, and for commercially-licensed, open source, and internally developed tools. [More](#).

*Laurent Lachal, Ovum Open Source Research Director*

**FIGURE 31.** WE INVITE YOU TO JOIN THE COMMUNITY AT OPEN-SERVICES.NET

## 5.2 The Jazz Foundation reduces total cost of ownership



**FIGURE 32.** THE JAZZ FOUNDATION PROVIDES A COMMON UI FRAMEWORK

The Jazz Foundation also provides a set of services that help drive a common user experience in the reference example. Note how the web user interfaces, pictured in Figure 32, for Team Concert and Quality Manager (and soon, Requirements Composer) share a common look through the use of banners, rich hovers, dashboards and cross-repository widgets.

In the reference scenario, we briefly mentioned the use of project timelines and process templates. Each of the products uses a Project time line, with process templates that provide process enactment within each tool, which are a service provided by the Jazz Foundation. The Jazz Foundation also provides delegated authorization, which was not apparent in the scenario. Subsequent versions of the foundation will include additional services such as user administration, cross-repository query, and many more services aimed at reducing the total cost of creating, maintaining, and owning a C/ALM solution.

## 5.3 Jazz.net: An open, innovative, collaborative, community

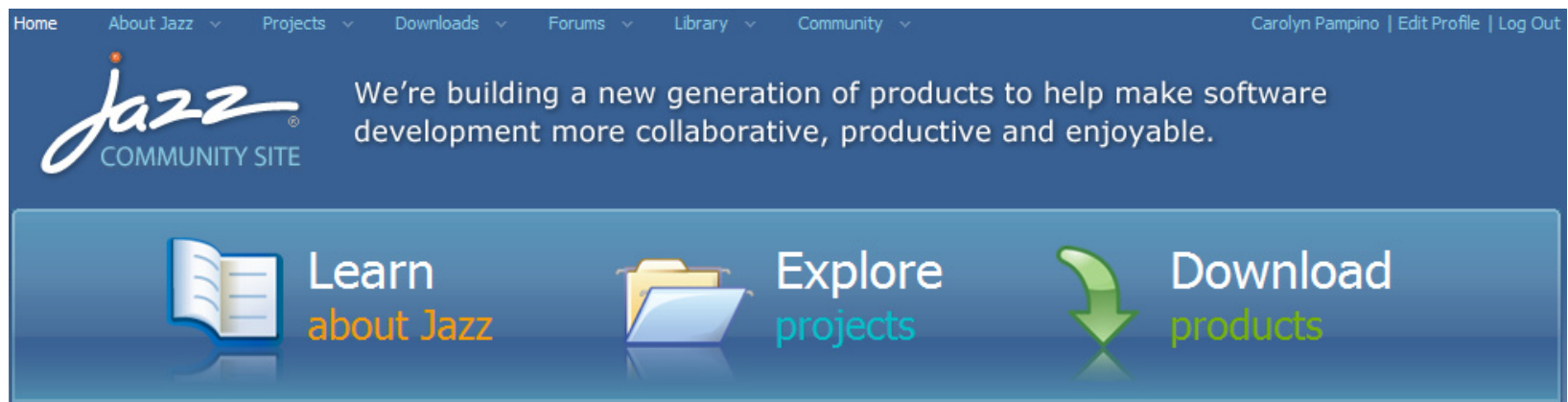
Thanks to many of you, Jazz.net is a huge success. We have completely re-launched the site and added more projects, downloads, forums and media to help you get started with this new generation of C/ALM products and integrations. (See Figure 33)

The project section contains detailed information about each of the products highlighted in this eBook along with additional projects that were not covered.

We develop our products in the open and invite you to contribute your ideas or concerns. You can browse the forums to see what other community members are discussing. The library is full of articles, videos, and tech notes that you can read.

Project plans and dashboards are public so you can see what our teams are planning to deliver and how they are tracking against their goals. Early milestone builds, source code, and completed projects are available for download on a trial basis. You can even comment on, or submit work-items to the development teams.

The project list is growing, and the action never stops, so be sure to join us at [Jazz.net](http://Jazz.net).



**FIGURE 33.** A GREAT NEW EXPERIENCE AT JAZZ.NET