Rational. software

# Rational Team Concert — Agile team development with Jazz

*Jean-Michel Lemieux, IBM*
*Jazz SCM Component Lead*

*Geoffrey Clemm, IBM*
Jazz Interop Component Lead, Distinguished Engineer

IBM Rational Software Development Conference 2007

What keeps me **Rational**?





IBM

# Talk Outline

- Jazz and Concert overview
- Demo 1 – Guided tour
- Jazz Views for ClearCase
- Demo 2 – ClearCase and Jazz interop

# What is Jazz?

## Innovation

A major investment by IBM to create a scalable, extensible team collaboration platform for seamlessly integrating tasks across the software lifecycle


## Tooling the Eclipse Way

A commercial project led by the IBM team that brought you the Eclipse Platform, tooling the agile practices of this proven open collaborative model


**Innovative Software Engineering**

## Community

Jazz.net – Jazz project venue for open commercial development of Jazz platform and Jazz-based products and an extension of the world wide Eclipse ecosystem


## Vision

A vision for the value and experience that future Rational products can bring to software and systems delivery teams

Rational.

# How is the Thinking Behind Jazz Different?

## Desktop Integration **à Lifecycle Integration**

Eclipse established a desktop client integration model for individuals, Jazz will establish a similar integration model across the lifecycle of software projects

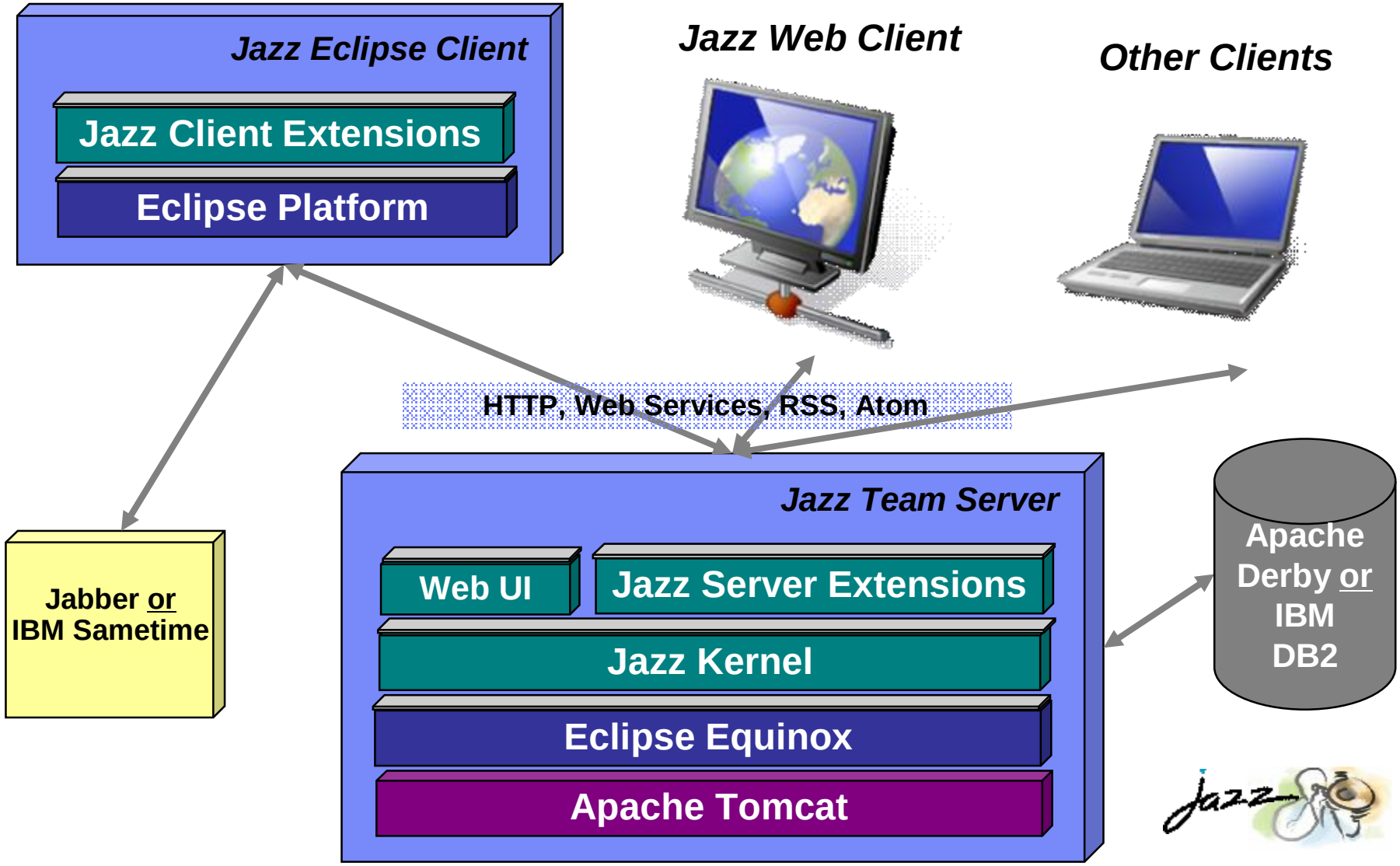
## Function First **à Team First**

Jazz will challenge us to think first about how people work together and then about the tool function needed by individual practitioners for specific roles

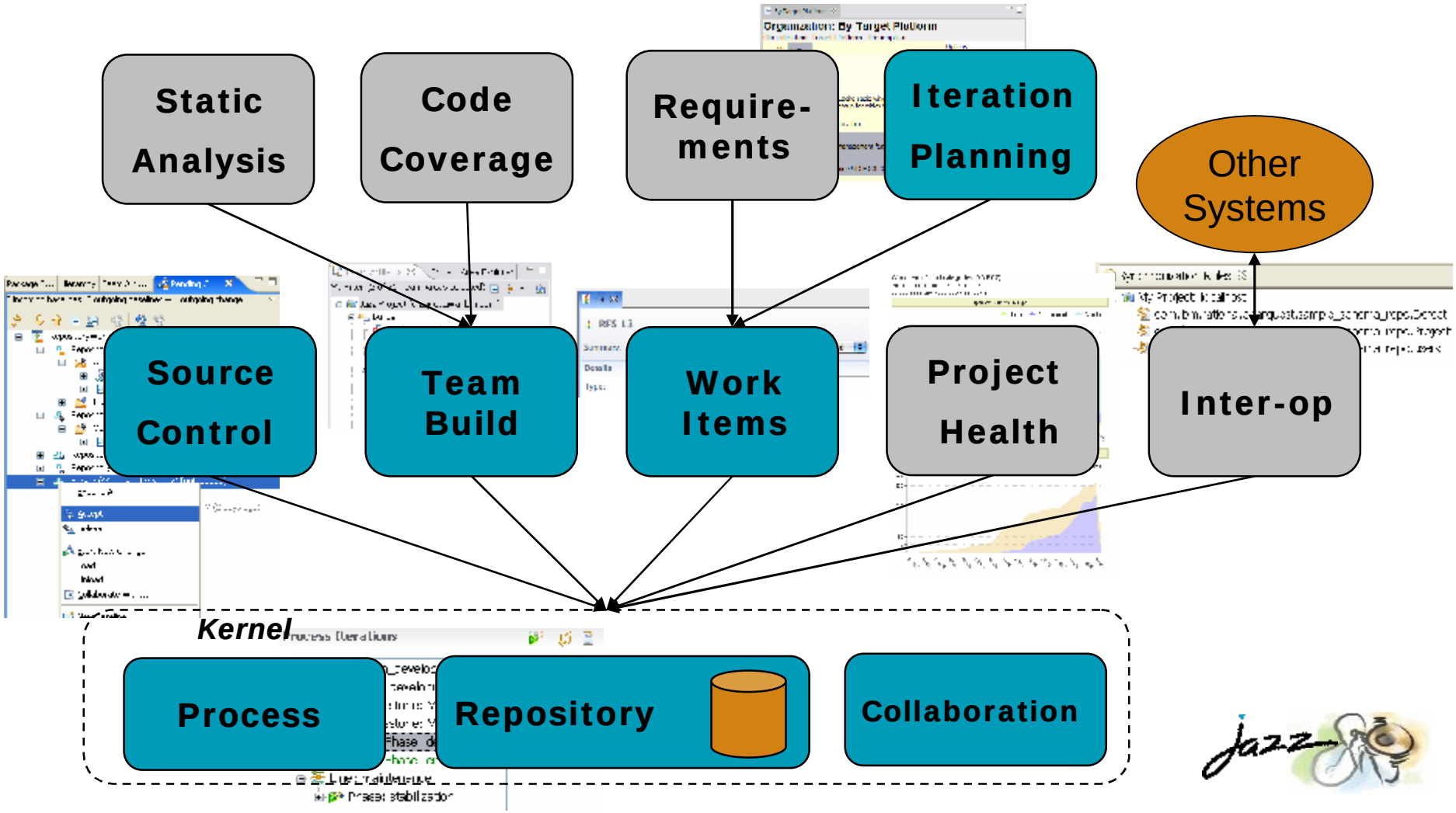
## Manual **à Process Aware & Transparent**

Focuses on automating the unique dynamic of a team enhancing productivity by enabling real-time collaboration, visibility and transparency across the team, and process enactment

# Jazz Platform Architecture – Open Source middleware



# Jazz Platform and Concert components



# Concert for Agile Teams

- Agile teams need more than just an SCM system. They need tools to help them plan, organize, build, and ship their products.
  - **Benefits**: All phases of the project are tooled and integrated with the data in one single repository. One tool and one install per team member.
- Traditional SCM Agile best practices focused on the structure of the artifacts, Concert adds the structure of the team and its process.
  - **Benefits**: Projects and teams, the roles people play, and the process which is used is a fundamental aspect of Concert. Over time the process and team structure adopts to the needs of the team. Several best practice templates are available when starting a new project.
- Helps structure and plan iterations with measurable executable output so that the teams know what is expected of them.
  - **Benefits**: Development is a lot more about communication than writing code, it's critical that everyone understand the goals, timelines, and what is expected of them.
- Rich integration allows team members to easily switch between different roles which is typical for agile teams (planning, development, testing).
  - **Benefits**: Developers can update plans, start builds, and configure the team's process. Project managers can see the code, work items, and what is being delivered. Testers can see what is planned in the milestone. Agile roles for agile teams - less time wasted waiting for someone else to do something for you.
- Low total cost of ownership.
  - [illegible]

# SCM Agile Best-Practices* are supported

| | |
|---|---|
| **Mainline** | Minimize merging, and keep the number of active codelines to a manageable set by developing on a Mainline. |
| **Active Development Line** | Keep a rapidly evolving codeline stable enough to be useful by creating an Active Development Line |
| **Private Workspace** | Prevent integration issues from distracting you, and from your changes causing others problems by developing in a Private Workspace. |
| **Repository** | Set up a new workspace by populating it from a Repository that contains everything that you need. |
| **Private System Build** | Check to see that your changes will not break the build by doing a Private System Build before commiting changes to the Repository. |
| **Integration Build** | Ensure that your code base always builds reliably by doing an Integration Build periodically. |
| **Third Party Codeline** | Manage vendor code by using a Third Party Codeline. |
| **Task Level Commit** | Organize source code changes by task-oriented units of work. |
| **Codeline Policy** | Create a Codeline Policy to help developers decide when to check in code to a codeline and what procedures to follow before a checkin on each codeline. |
| **Smoke Test** | Ensure that the system still works after you make a change by running a Smoke Test. |
| **Unit Test** | Verify that a module still works after you make a change by running a Unit Test. |
| **Regression Test** | Ensure that existing code doesn't get worse as you make other improvements by running a Regression Test. |
| **Private Versioning** | Use Private Versioning to allow you to experiment with complex changes locally, yet still be able to take advantage of the features of a version control system. |
| **Release Line** | Maintain released versions without interfering with your current development by establishing a Release Line. |
| **Release-Prep Codeline** | Stabilize a codeline for an upcoming release while also allowing new work to continue on active codelines by doing the stabilization work on a Release-Prep codeline. |
| **Task Branch** | Have part of your team perform a disruptive task without forcing the rest of the team to work around them, using a Task Branch. |

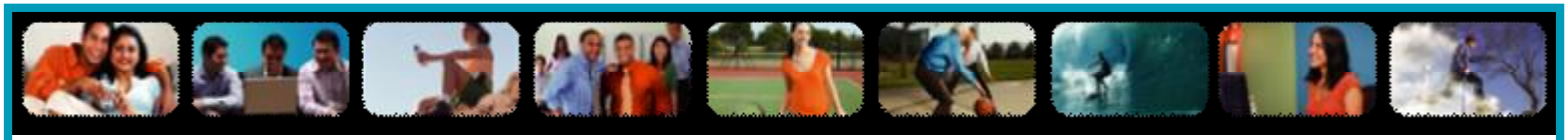
*Software Configuration Management, Patterns: Effective Teamwork, Practical Integration
By Steve Berczuk with Brad Appleton Published by Addison-Wesley (http://www.awprofessior

# Extended Agile features supported

| | |
|---|---|
| **Team First** | Teams and members are first class entities in the Repository with configurable roles and processes. |
| **Awareness** | Team and member events as Atom or RSS feeds are provided for all interesting changes that occur in the repository that affect you: deliveries, work item changes, build status, process changes, role changes, plan changes. |
| **Iteration Planning** | Iteration plans with unstructured and structured information can be used to transparently plan and track progress. |
| **Daily Planning** | Allows developers to track and manage the work they have committed to in an iteration. |
| **Custom Workflows** | Allows teams to customize the workflows they use for making changes and managing their work items. |
| **Process Enactment** | Customize based on the project phase the process rules that help your team be more productive, such as ensuring that correct licenses are in all file, all changes have an associated work item, or resolved work items are assigned to the correct milestone. |
| **Process Permissions** | Configure based on role and project phase who can modify which artifacts. |
| **Approvals** | Support for tracking peer reviews for build promotion, code changes, or designs. |
| **Build Reproducibility** | The artifacts included in any build can be loaded with a single click from the build result. Makes it easy to reproduce and debug failures in specific releases of the code. |
| **Build Promotion** | Builds can be promoted either by using a stream based promotion of the changes or by tagging the builds. |
| **Build Comparison** | To see what went into a build, you can compare two builds and see what has changed. |
| **Traceability** | Everything is linked in Jazz: change-sets to work items, work items to builds, test cases to builds. |

# World Class SCM, Work Item, and Build features

- SCM
  - The usual SCM checklist: atomic commits, change-sets, renames are tracked, multi-streaming is cheap (branching).
  - Multi-stream development including change-set cherry picking, history sensitive merging, and visual stream editing.
  - Flexible conflict resolution for all conflict types, renames, evil twin, structural, and content auto-merge. Allows agile teams to refactor without stopping all development.
- Work Items
  - Rich in-place editors with built-in traceability and linking.
  - Configurable work item types, states, workflows, and approvals.
  - Iteration and personal planning with free-form and structured plan editors.
- Build
  - Brings build alerts, progress monitoring, and control to the team in a rich UI.
  - Provides valuable linkage between builds, work items, and change-sets.
  - Designed to work with the team's existing build tools.



# Demo

# Plan, Develop, Stabilize

# What are Jazz Views for ClearCase?

- A way to use Team Concert as a “front-end” for ClearCase
- Three variants:
  - Single User (Derby/Tomcat on your PC)
  - Small Team (Derby/Tomcat on local server)
  - Medium Team (DB2/WAS on an IT server)
- Currently, only for UCM
  - Stretch goal for first release: base CC as well

# Architecture

- Runs in the Team Concert client
- Talks to ClearCase using the WVCM API defined in JSR-147
  - Provides maximal alignment with Team Server
  - A ClearTool-based WVCM provider being built for:
    - Access to pre-Team Server versions of ClearCase
    - Access to dynamic views
    - Access to snapshot views

# Single User Variant

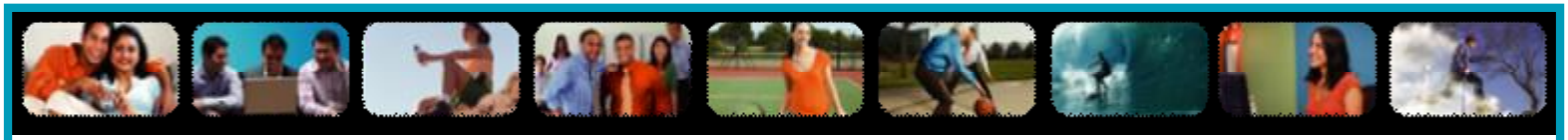
- Provides a superior disconnected experience.
  - Private checkins
  - Disconnected access to activities
  - Suspend/restore of change-sets
  - Private baselines
- Requires a PC with sufficient memory and disk
  - We're still measuring what is "sufficient"

# Small Team (Derby/Tomcat) Variant

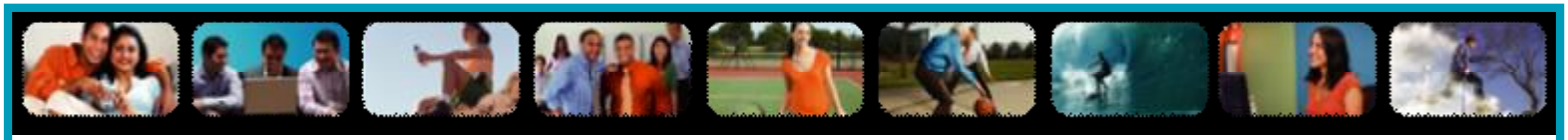
- All the individual benefits of single user access
- Provides the collaborative capabilities of Team Concert
- Minimal administrative overhead of server
- Minimal resource usage on user's machine

# Medium Team (DB2/WAS) Variant

- Provides collaborative benefits of Team Concert to a broader team
- Administrative overhead from DB2/WAS
- Minimal resource usage on user's machine



# Demo 2



# Thank You