IBM Rational Software Development Conference 2008

WHERE TEAMS ARE R-HEROES

Parallel Development with IBM Rational Team Concert

Jean-Michel Lemieux
Jazz Source Control Lead, Jazz Project Management Committee, IBM
Jean-Michel_Lemieux@ca.ibm.com

SDP22

Rational. software

Go to IBM

Overview

- Why another SCM?
- The basic building blocks
- Product level parallel development

SCM in the Open Source community

Git and Linus' popular Git presentation at Google

- Really, really fast! Easy to merge, and happens to be distributed.
- Simple: git init, git commit, git add.
- The cool tool on the block, but mostly for OS and small personal projects.

- Subversion is growing up with 1.5
  - Better merge support in 1.5
  - A lot of industry adoption, easy to setup and low administration.
- However, neither of the OS tools have focused on ALF integrations or process support. Their main focus in on developer productivity and less on organizational productivity.

Tech Talk: Linus Torvalds on git

* http://www.youtube.com/watch?v=4XpnKHJAok8

SCM in commercial organizations

- Most commercial projects are now being developed by distributed teams.
  - Transparency between teams is essential.
  - Isolation and coordination is important.
  - WAN friendly tooling is critical.
- Agile small teams
  - Many teams are small and focused on specific short lived projects.
  - They need access to planning, SCM, Defect Tracking, and Builds.
- Usability and getting started are important aspect whereas in the Open Source world features can often outweigh usability.

Why another SCM?

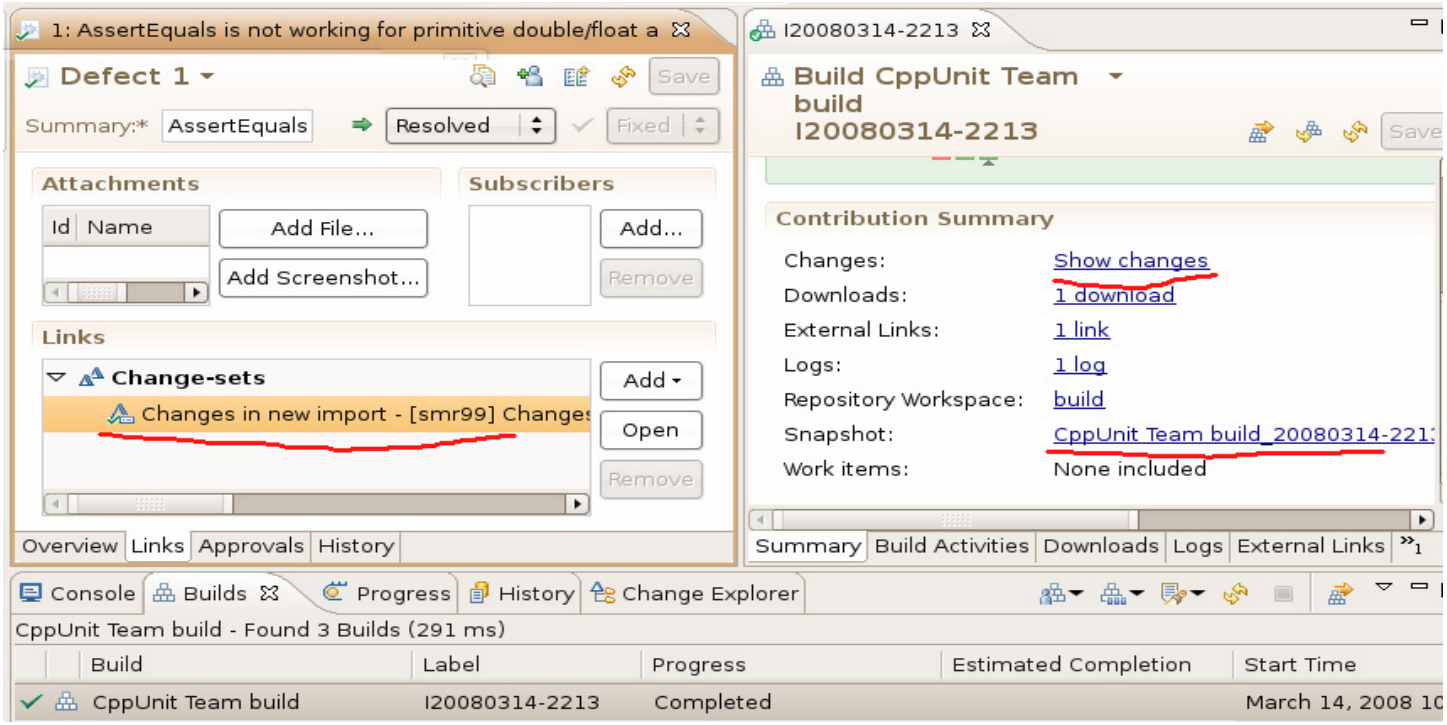
Our need to address both forces:

- **Integration**: Rich data model integrations with defect tracking, builds, process, and other ALM tools.
- **Modern SCM model**: change sets, streams, components, GDD support, and simple parallel development.
- **Easy of install and administration**: use standard middle ware, single zip install for tire kickers and small teams.

The freedom to experiment with tight integrations at the data and UI levels.

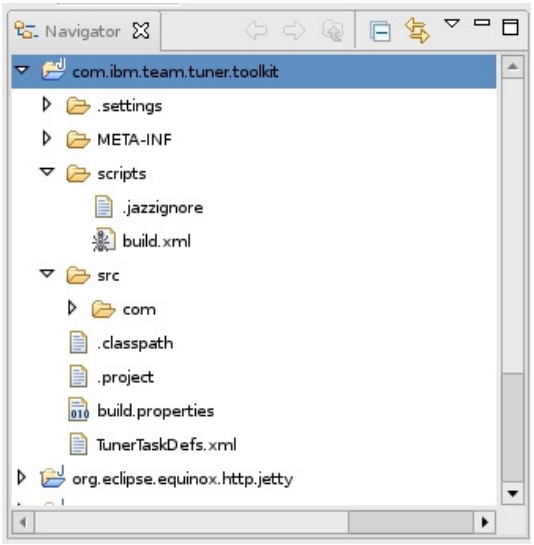
The Jazz SCM

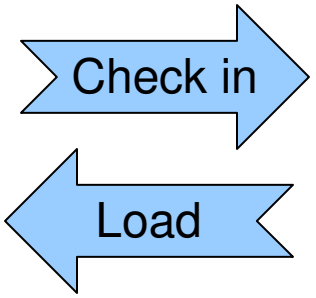
- It's no longer simply about the files or the SCM system.
- Leverage linking as much as possible between artifacts: builds, defects, plans...
- Rich integration into the application life cycle.

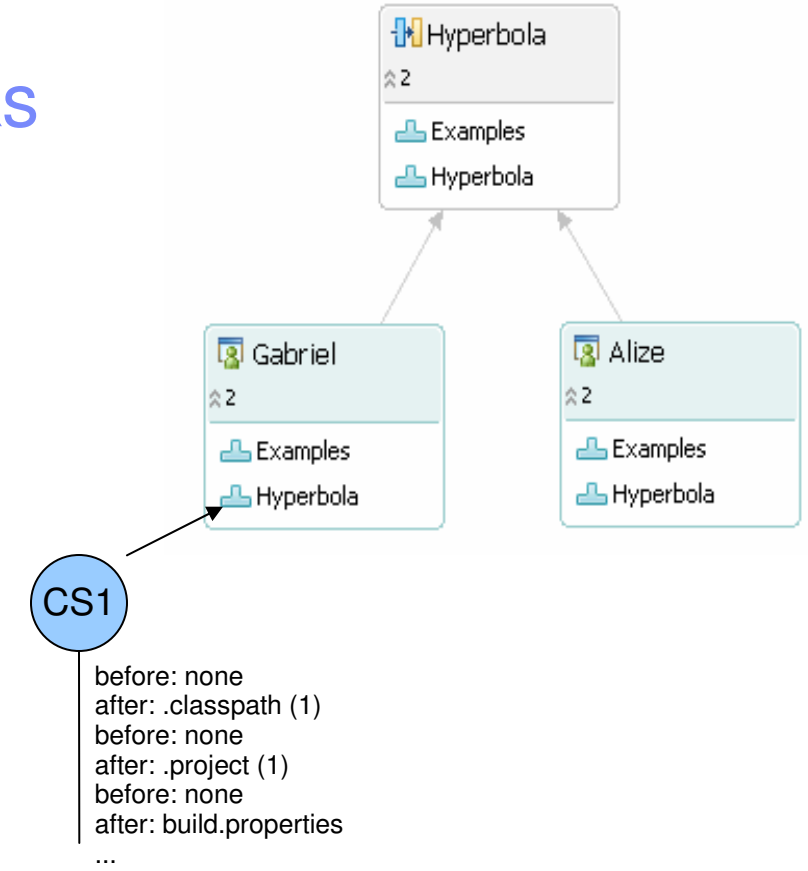


The Jazz SCM – Building Blocks

- Start with files on your local disk.



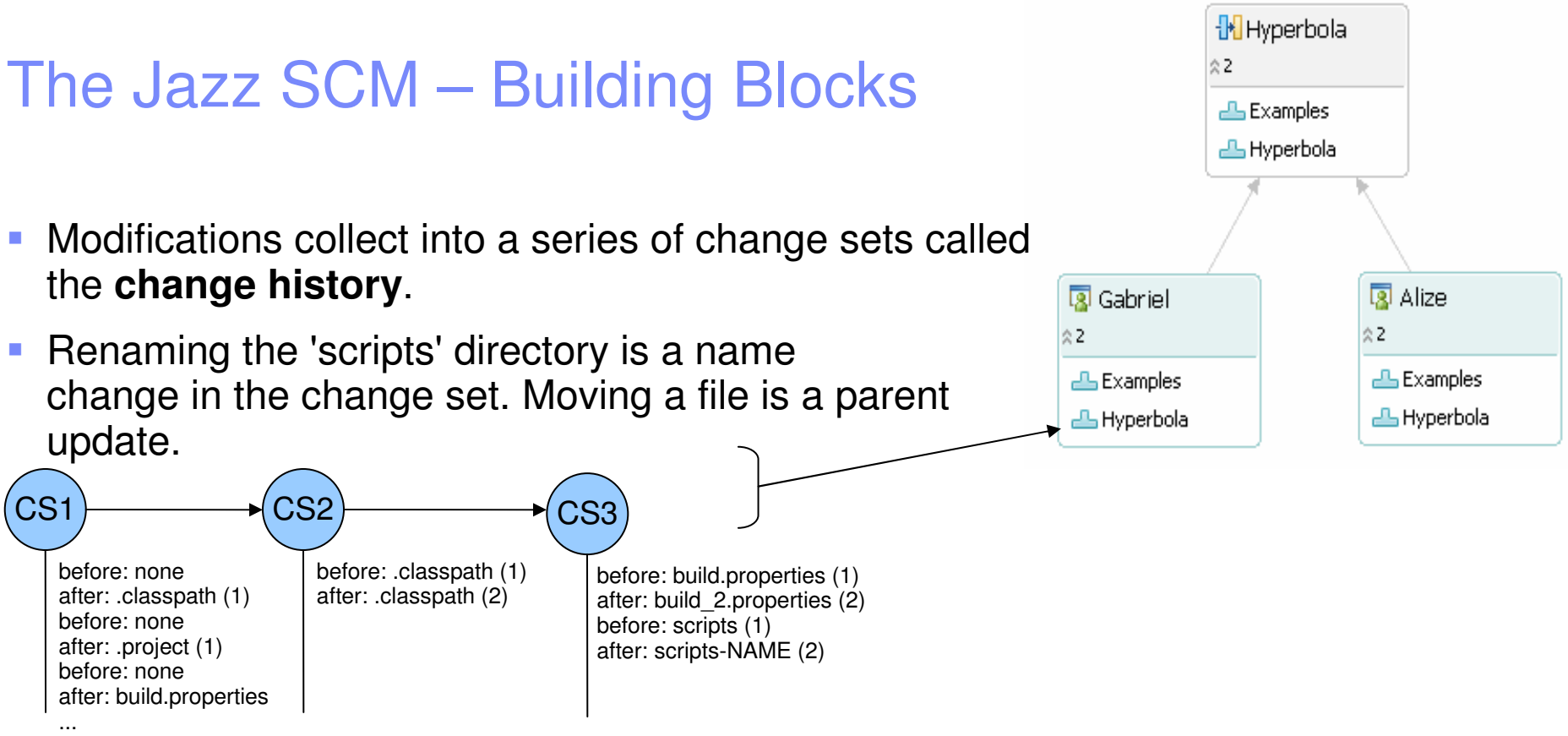




- Files are checked into a **change set**. The change set records the before and after states of each file. Content is stored separately and is delta-compressed.
- Every user gets a sandbox on the server called a **repository workspace**. Load the repository workspace to get a copy on disk.
- Change sets are owned by a **component**. A repository workspace can contain one or more components.

The Jazz SCM – Building Blocks

- Modifications collect into a series of change sets called the **change history**.
- Renaming the 'scripts' directory is a name change in the change set. Moving a file is a parent update.



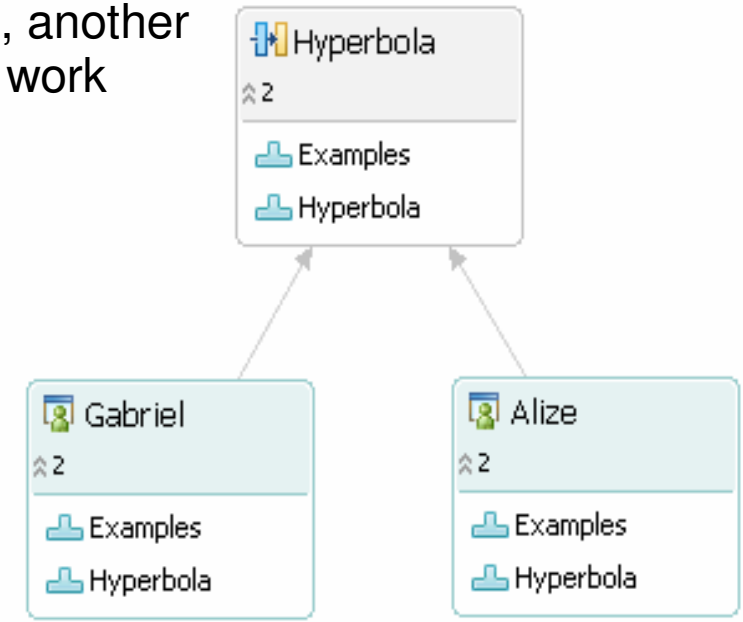
- Each component instance has a current **configuration**. This is the file tree that results from applying the change sets in chronological order.
- Change sets can be discarded, suspended, and resumed. They are the base currency in the SCM model.

The Jazz SCM – Building Blocks

- Change sets flow between repository workspaces and a **flow target** using **deliver** and **accept**.
- A **stream** is used to store a shared change history while the repository workspace is writable only by its owner. Streams are owned by a team.
- You can accept change sets from either a stream, another repository workspace, or cherry pick them from a work item or chat session.

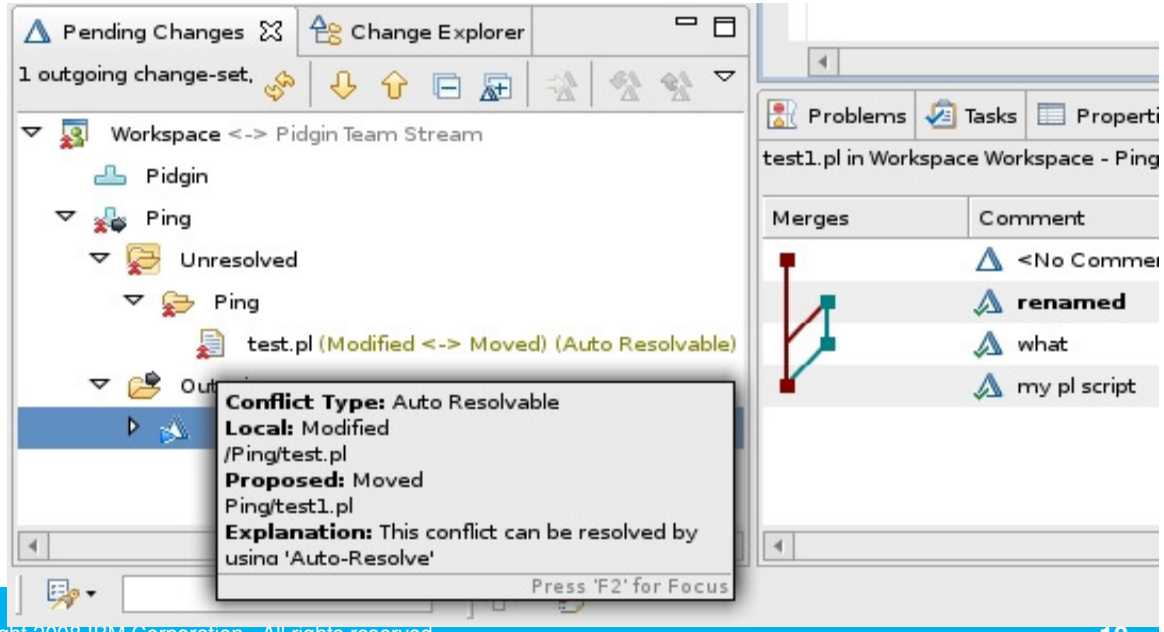
Parallel development is intrinsic in the design.

Branches are not part of our vocabulary. The word scares people. Instead, we talk about making and flowing changes.



The Jazz SCM – Building Blocks

- In a parallel world, **conflicts** will happen and must be easy to resolve. Deliver won't create a conflict, but accept or resume can.
- Conflicts are stored as meta data in the repository workspace and can be resolved by either accepting more change-sets, discarding, or merging.
- All types of structural conflicts (eg, moves, renames) are tracked and can be merged.

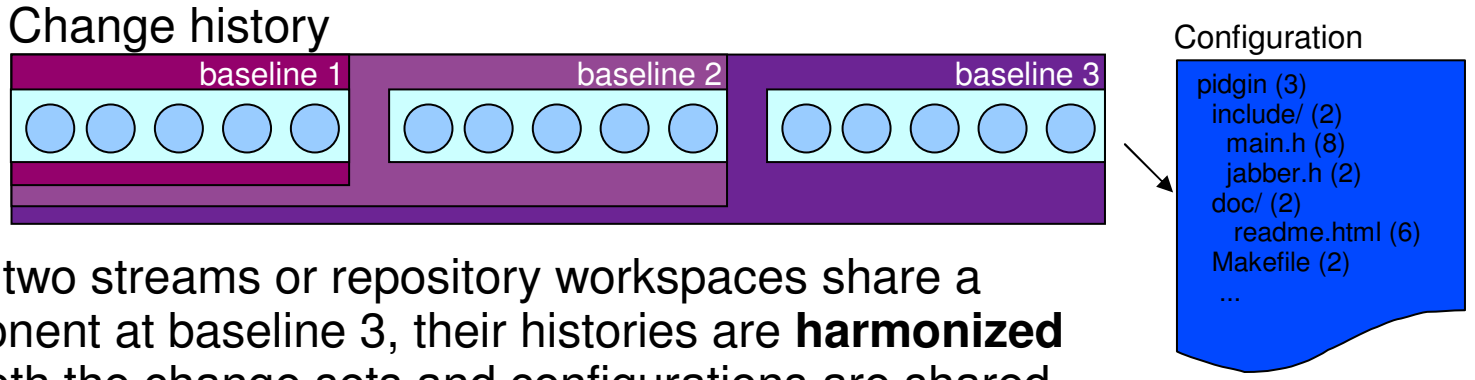


DEMO 1

The basics

Baselines and Configurations

- To save an immutable state of a change history you create a **baseline**. Baselines can also flow between flow targets using deliver and accept.
- Baselines are chained together to allow change history re-use in all streams and repository workspaces. This avoids having to copy change histories for each new stream or repository workspace.
- Configurations are also shared between streams and repository workspaces.



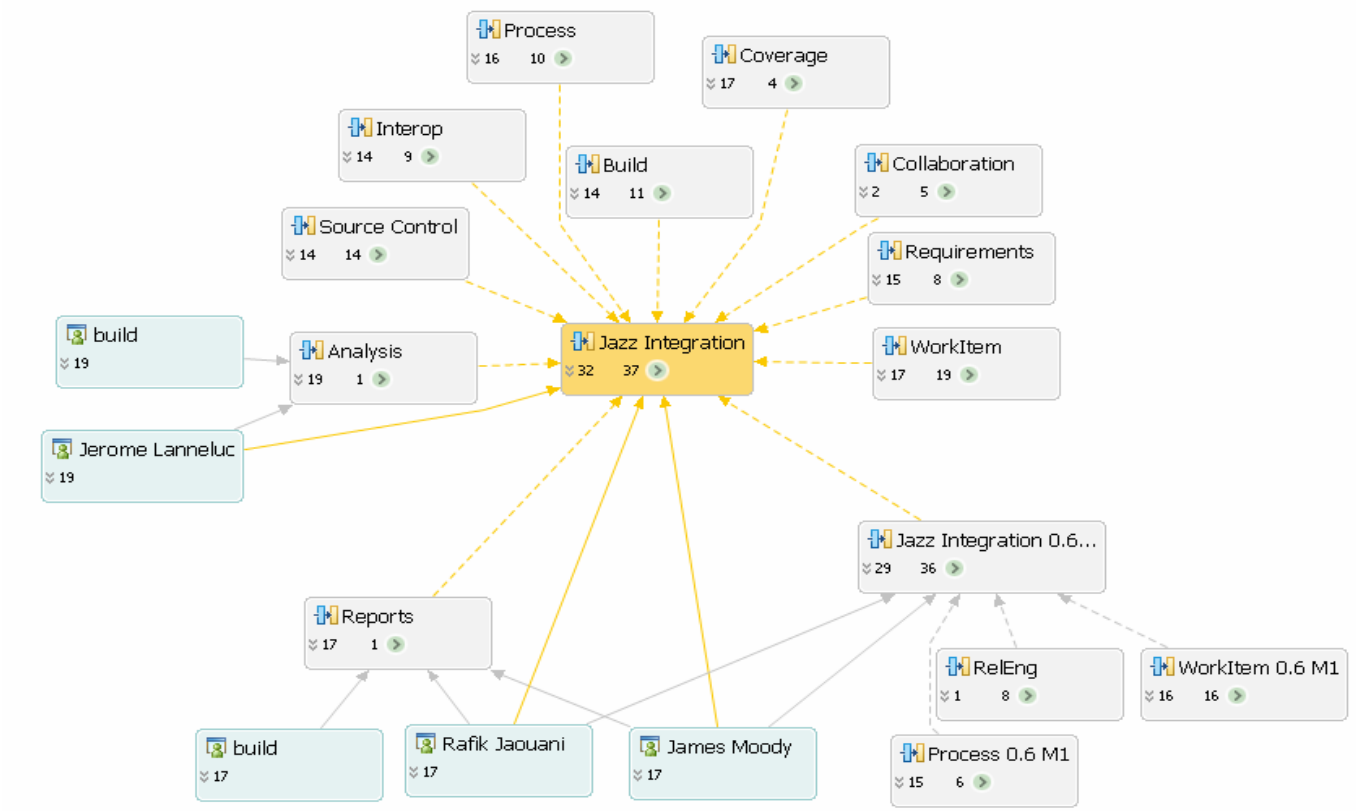
- When two streams or repository workspaces share a component at baseline 3, their histories are **harmonized** and both the change sets and configurations are shared.
- New changes are built on top of the shared state and over time new baselines flow and re-harmonize into streams and repository workspaces.

Snapshots

- To save an immutable state of a repository workspace or stream create a **snapshot**.
- Snapshots don't flow, but are associated with streams, builds, or repository workspaces. You can promote a snapshot between streams.
- Snapshots are used to seed new streams or repository workspaces.
- Create snapshots for all important points in your development process. Milestone builds, releases, important builds...

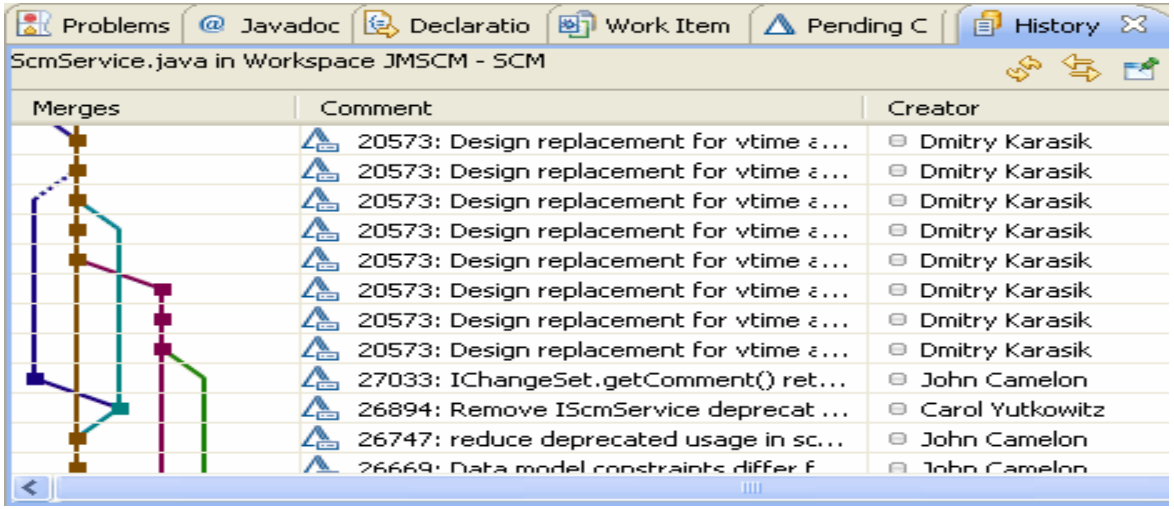
Teams of teams

- This simple model allows scaling to teams of teams.
- Provides isolation when needed at both the individual and team level.



Summary

- The Jazz SCM is entirely change set based. There is no file level branching or tagging.
- Focuses on user level simplification with the added power of parallel development.
- Conscious effort to never use the word branch in the UI. People are afraid of branches, but want isolation and the features they provide



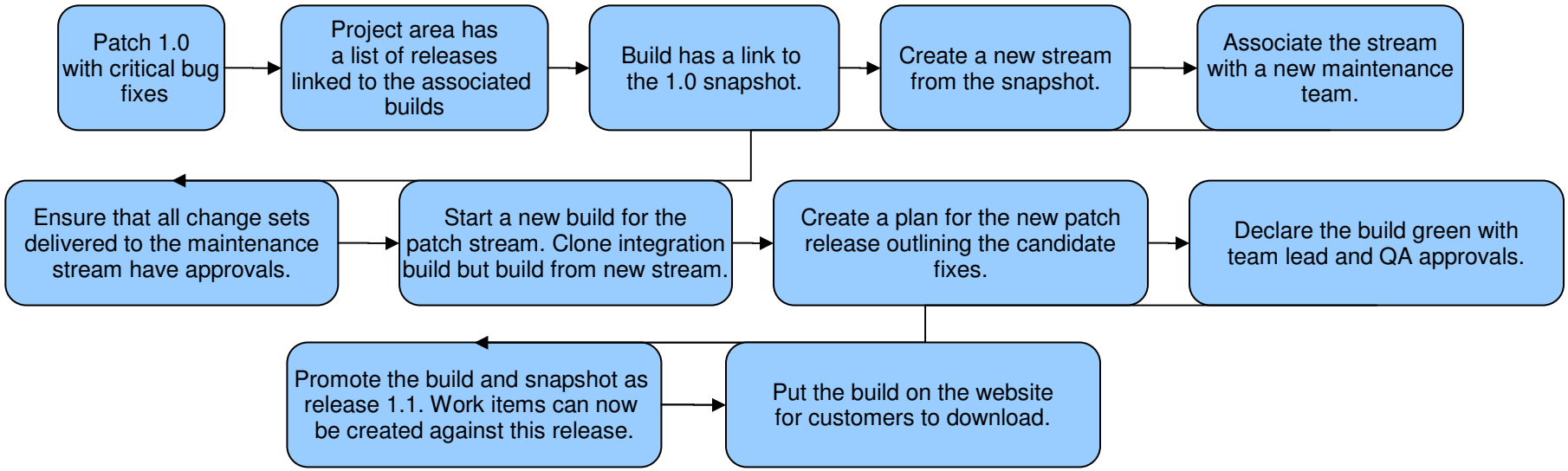
Parallel Development

- The model supports several levels of parallel development:
  - **Repository workspaces** – Provides constant isolation. You don't have to make your changes visible to the team just to backup or use the repository features.
  - **Suspend and Resume** – Provides task level isolation for personal work.
  - **Work Item links** – Provides light weight task level isolation for personal of team work. Work on a feature, attach to a work item and discard from your workspace. You or someone else continues the work by accepting the change sets back into their repository workspace.
  - **Streams** – Provides team isolation.
  - **Development Lines** – Provides process isolation.

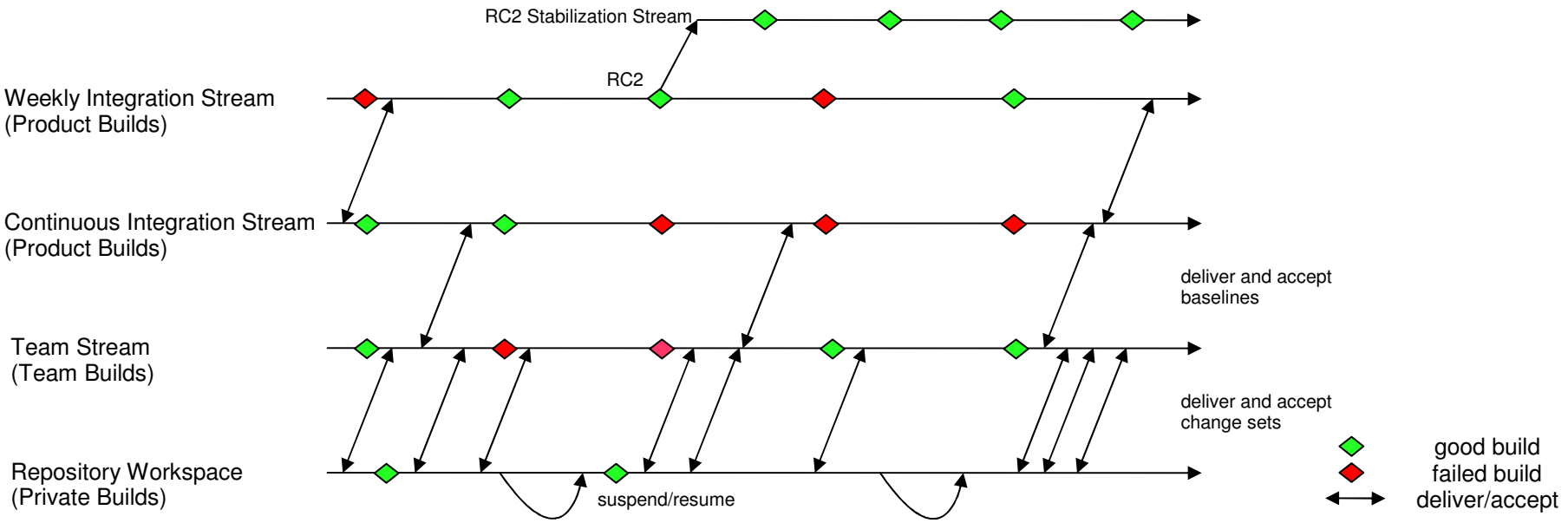
Isolate ***work*** not ***people***. Encourage transparent planning and flexible isolation.

It's not just about SCM

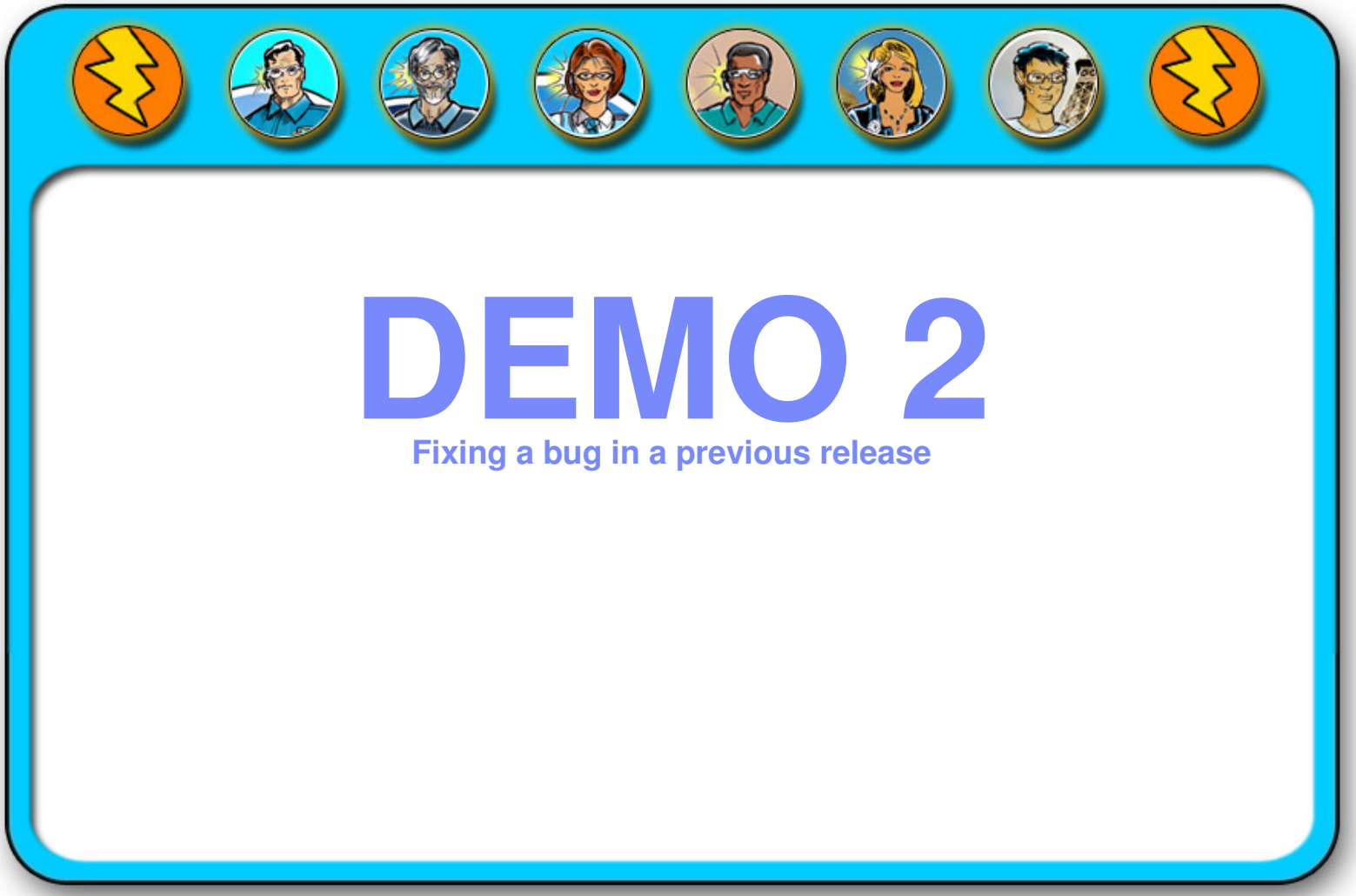
- There is much more to product level parallel development then patching the code.
- Team level productivity and turn around time for delivering fixes is impacted by the tooling of the entire process.
- Consider the number of tasks involved in patching a release that aren't related to SCM, but are tooled in Rational Team Concert.



How the Jazz team works



- Each build references a snapshot of the artifacts that were built. Reproducing a build or patching a build is as simple as creating a new stream from the snapshot. There is no planning needed to tag or baseline, builds take care of that for us.
- Developers deal mainly with accepting and delivering to their team stream. Flowing changes to and from the integration streams is the responsibility of one person on each team.



DEMO 2

Fixing a bug in a previous release

THANK YOU

Learn more at:

- IBM Rational software
- IBM Rational Software Delivery Platform
- Process and portfolio management
- Change and release management
- Quality management
- Architecture management
- Rational trial downloads
- Leading Innovation Web site
- developerWorks Rational
- IBM Rational TV
- IBM Rational Business Partners