IBM Software Group

Continuous integration using Rational Team Concert

Peter Steinfeld

Rational. software

November 4, 2009

Overview

- The importance of using continuous integration
- How to use Rational Team Concert (Jazz) to implement continuous integration for your project
  - Stream setup
  - Build setup
  - Foolproof delivery to the integration stream
  - How to debug broken builds
- *The conversation you want to have*

The importance of continuous integration

"Happy families are all alike; every unhappy family is unhappy in its own way."

Leo Tolstoy (the first line of Anna Karenina)

- Integration is hard
  - It requires project-wide communication and coordination
  - It frequently uncovers unanticipated problems
- Integration takes time
  - It's difficult to predict how long integration will take
  - Since integration is project wide, a problem in one area can block progress in another
- Integration typically happens at the end of a project, the worst time to introduce risk and delay

Critical aspects of continuous integration

Successful projects practice continuous integration

- ✓The project has a single main stream of development – the *integration stream*
- ✓The integration stream builds successfully every day
- ✓Automated tests run as part of the build
- ✓Clear project status is visible via the build results

Perspectives of project stakeholders

Project manager

- ✓ *Knows project status by just looking at the build results.*
- ✓ *Knows that a feature has been implemented when it's in a build.*

Developer

- ✓ *Delivers code and tests to the integration stream continuously.*
- ✓ *Is careful to not break the build.*
- ✓ *Fixes broken builds, no matter who broke it.*

Tester

- ✓ *Build status is clear from the published build results*
- ✓ *Latest good build is the one to test.*

Barriers to continuous integration

- Integration delays of delivery of functionality. It takes extra work for developers to keep pace with the integration stream.
- Integration incurs the overhead of dealing with build problems. Such problems may be unrelated to a specific delivery.
- Integration requires expertise in the associated technologies – configuration management, builds, etc.
- Integration forces developers to communicate with each other and with the build team. Factors such as geographic disbursement can make this a real problem.
- There's never a good time implement continuous integration! We know that it's a good idea, but we're busy writing and debugging code.

Why you need continuous integration

- ✓Continuous integration reduces surprises and risk.
  - ✓There are many small integrations throughout the project rather than a few big ones.
  - ✓There's no "death march" at the end of the project when integration is forced.
- ✓Continuous integration promotes communication and cooperation among developers.
- ✓Continuous integration makes project status transparent.
- ✓Schedules are more predictable.
- ✓Continuous integration is good for morale. Everyone sees the project making steady progress.

Essential aspects of continuous integration

- ✓ A single integration stream
- ✓ Automated, daily build
  - ✓ Produce the customer deliverable every day
  - ✓ Build runs automated tests – if the tests fail, the build fails
  - ✓ Fix broken builds immediately
- ✓ Transparency
  - ✓ Build success is transparent
  - ✓ Functionality in the build is transparent

"Successful daily builds are the heartbeat of a software project. If you do not have successful daily builds, then you have no heartbeat, and your project is dead!"

Jim McCarthy, Microsoft VC++ product manager

How to set up continuous integration in Rational Team Concert

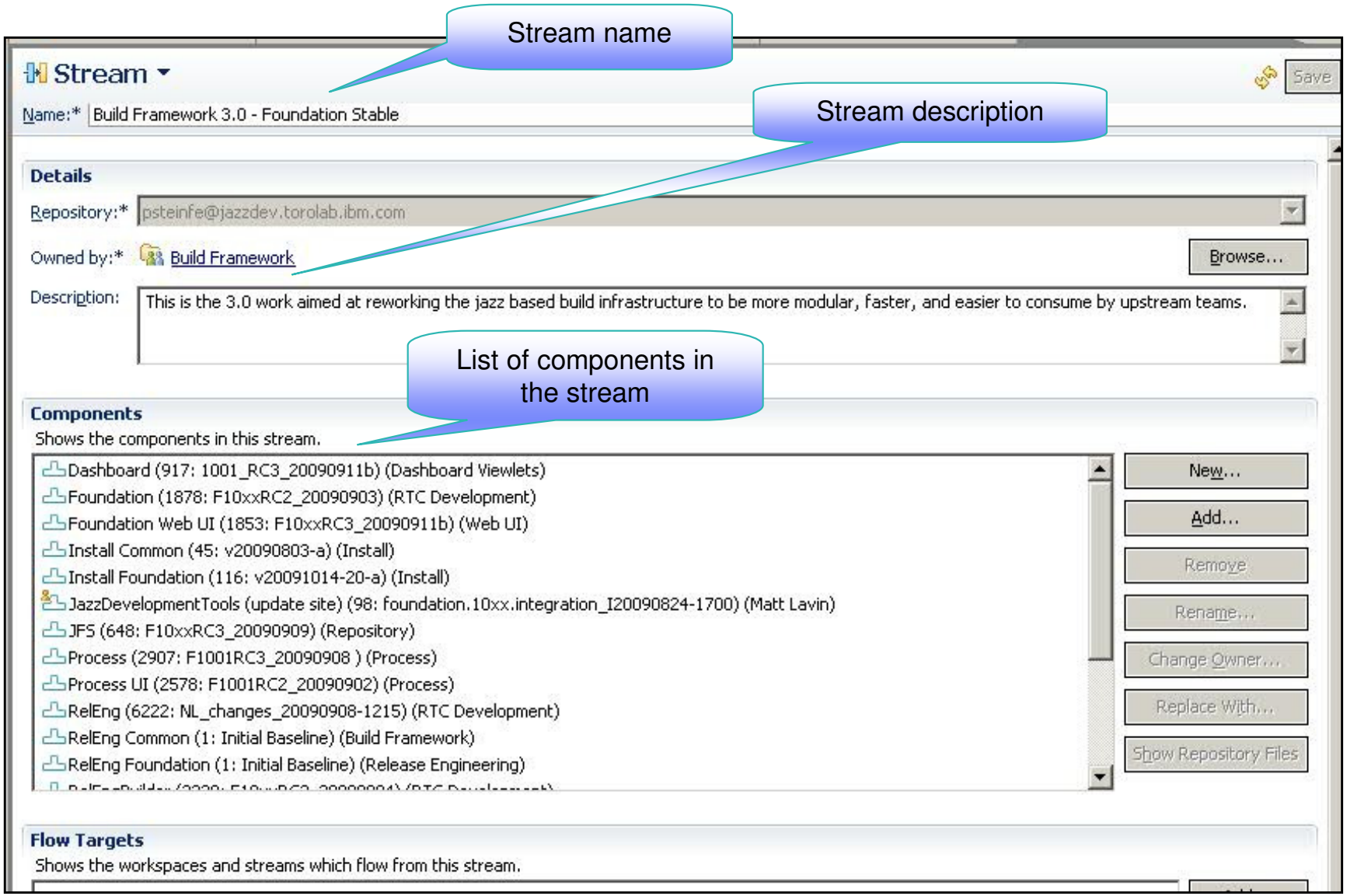
- ***Create an integration stream*** – This is where all of the project's source code lives.
- ***Each developer creates a personal Jazz workspace that "flows with" the integration stream.*** Each developer works in her own Jazz workspace and integrates it regularly with the integration stream.
- ***Set up a build of the integration stream.*** The build will produce the customer deliverables, run tests, and provide status.

The integration stream

Everyone is collectively responsible for the integrity of the integration stream.

- A Jazz stream captures a set of versions of files
  - Similar to the concept of a branch in other CM systems
  - The stream changes when people deliver new content
- Streams contain Jazz components
- Jazz components contain directories and files

A sample Jazz integration stream

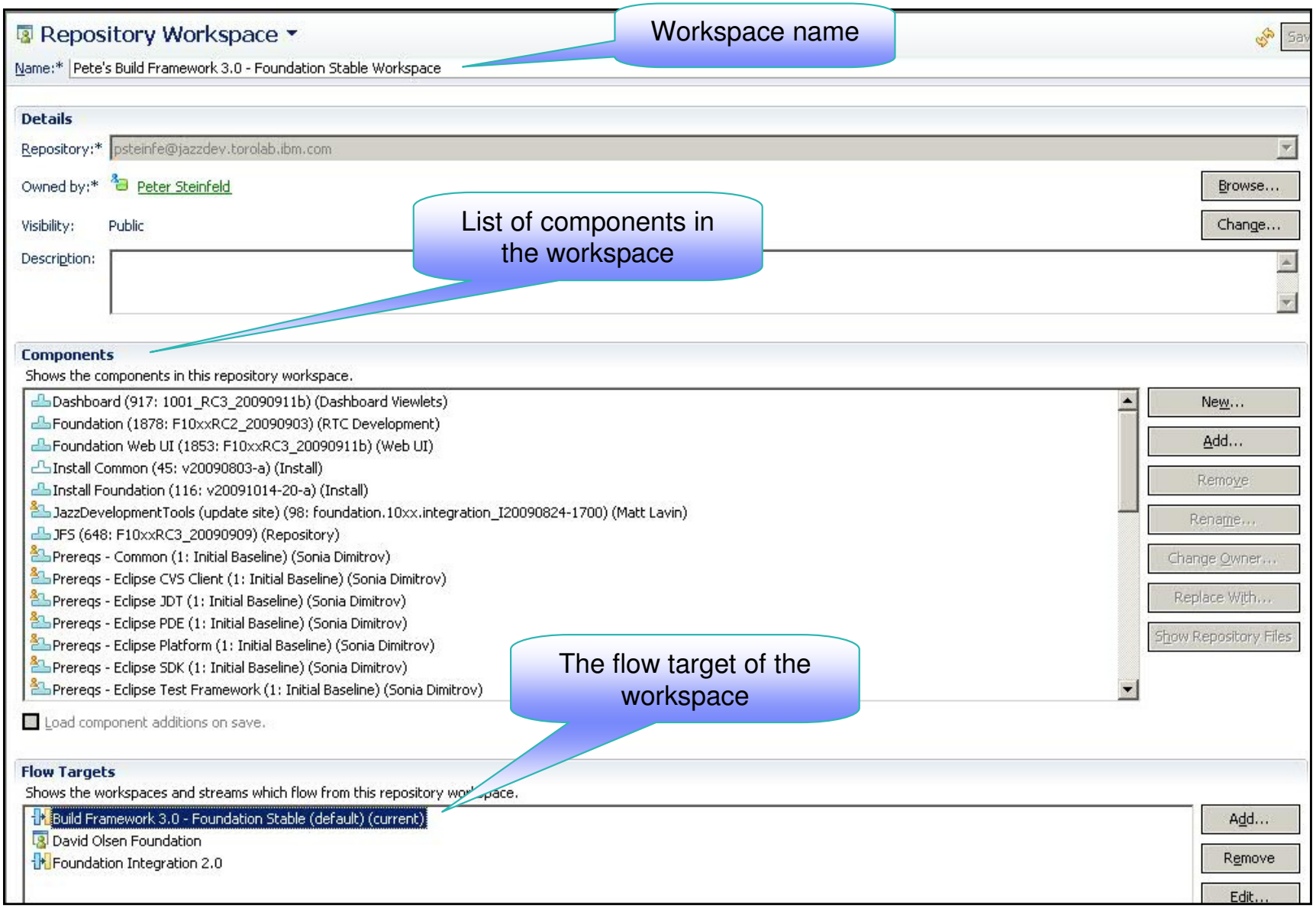


The developer's Jazz workspace

- Each developer has her own Jazz workspace
- All work is done initially in the developer's Jazz workspace
- The developer's Jazz workspace "flows with" the integration stream
  - When the Jazz workspace is created, its contents match the contents of the integration stream at the time of the creation of the workspace
  - "Accept" operations transfer changes from the integration stream to the developer's Jazz workspace
  - "Deliver" operations transfer changes from the Jazz workspace to the integration stream

Note that a Jazz workspace is not the same as an Eclipse workspace.

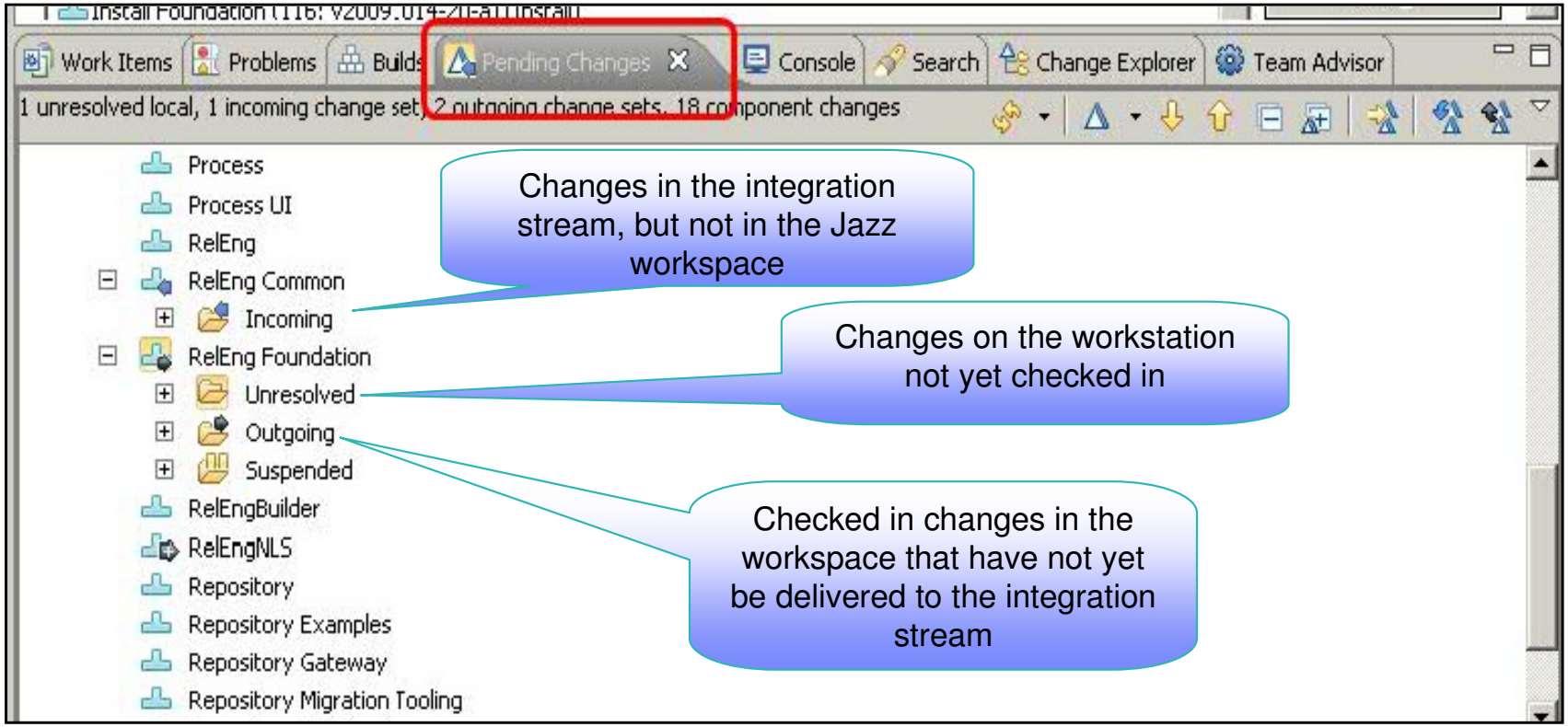
A sample Jazz developer's workspace



Jazz streams, workspaces, and files

- Different versions of the same file can exist –
  - On the developer's workstation
  - In the developer's Jazz workspace
  - In the integration stream
- At the moment a file is modified and saved, a new, different version exists on the workstation (unresolved).
- When the file is checked in, that version is transferred to the Jazz workspace (outgoing).
- When the change is delivered to the integration stream, that version will exist in the integration stream (incoming to other workspaces).
- The "Pending Changes" view shows the file states.

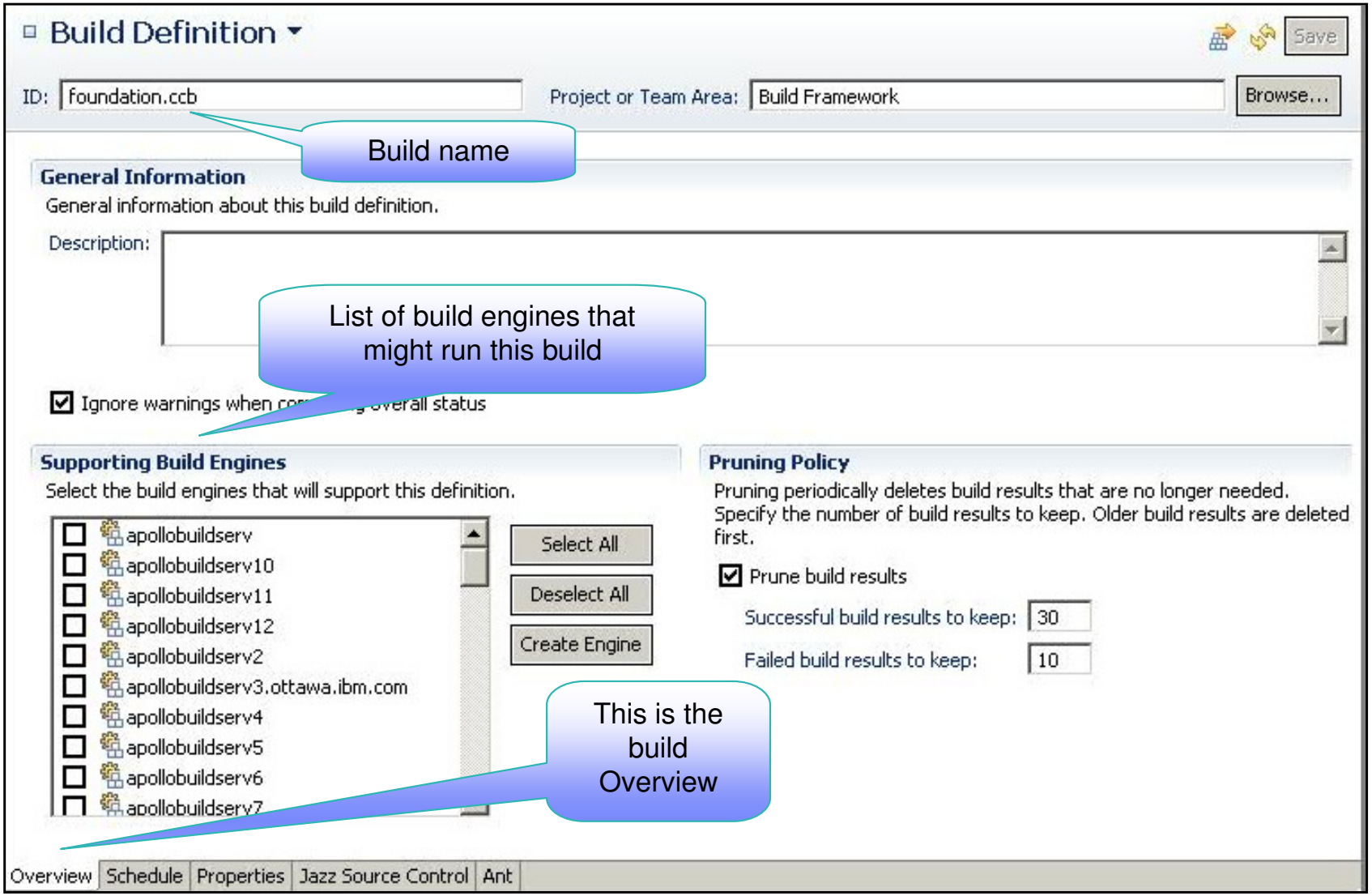
Using the Pending Changes view



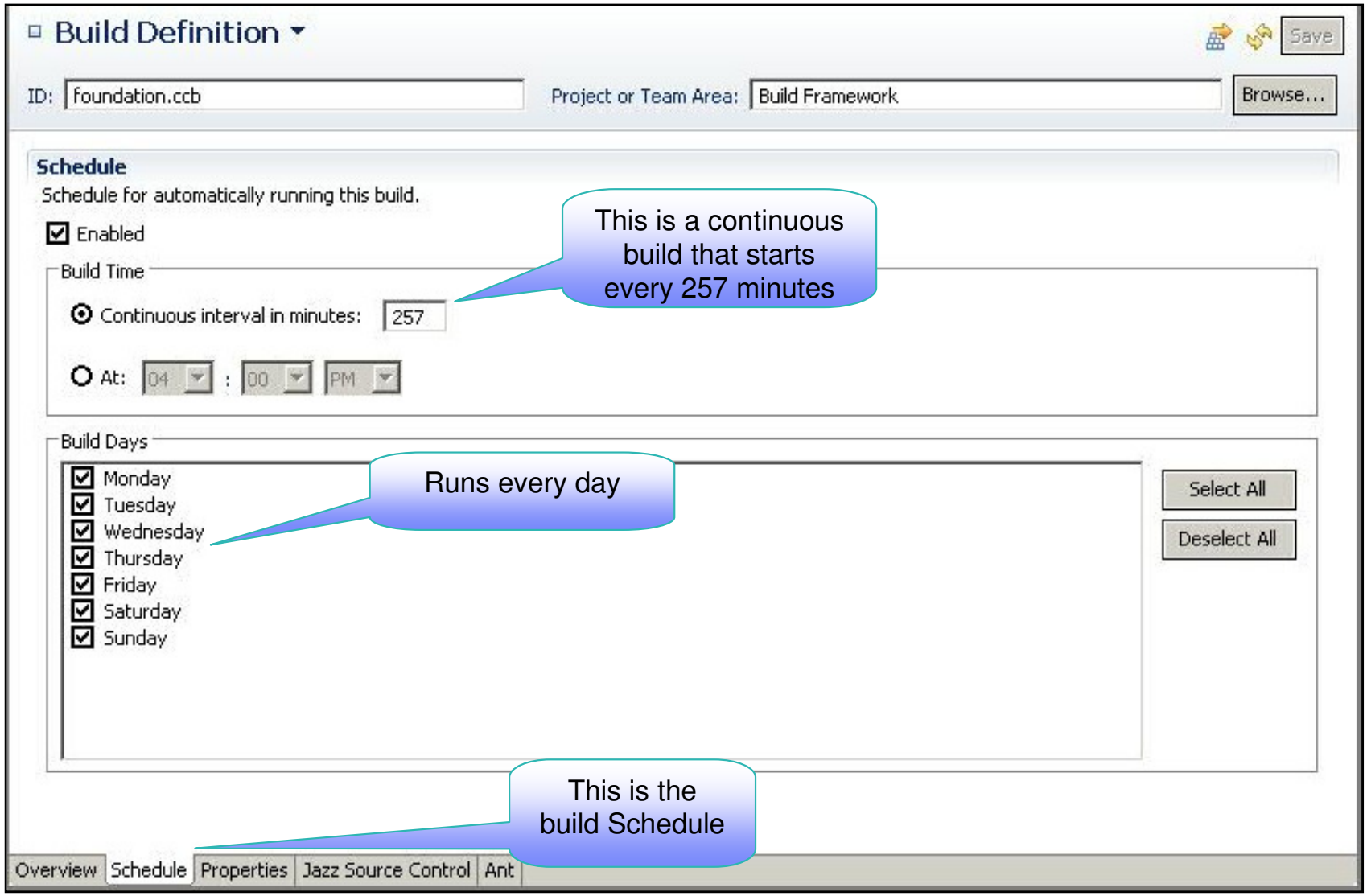
Setting up a Jazz build

- The build has its own Jazz workspace, just like a developer's Jazz workspace.
- You must create a script to do the actual build.
  - The build should produce the artifacts that will be delivered to the customer
  - The build should run the automated tests
- The build runs on an build machine
  - Multiple machines can be used for efficiency
- Schedule the build –
  - Schedule a continuous build for small teams
    - Continuous builds only run if new changes have been delivered
  - Schedule fixed build times for large teams and long builds
    - Fixed times enable developers to plan coordinated deliveries

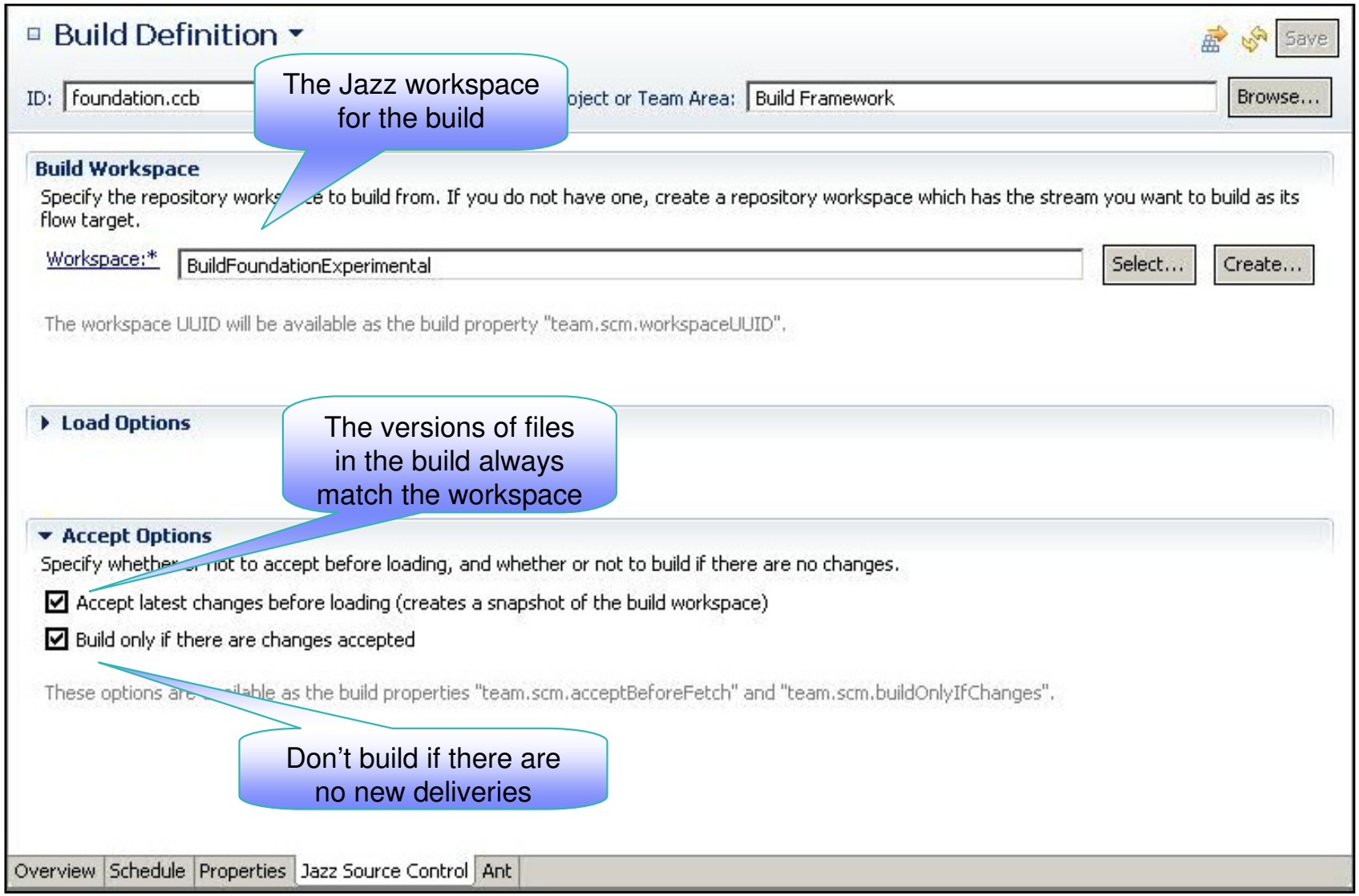
A sample Jazz build definition -- overview



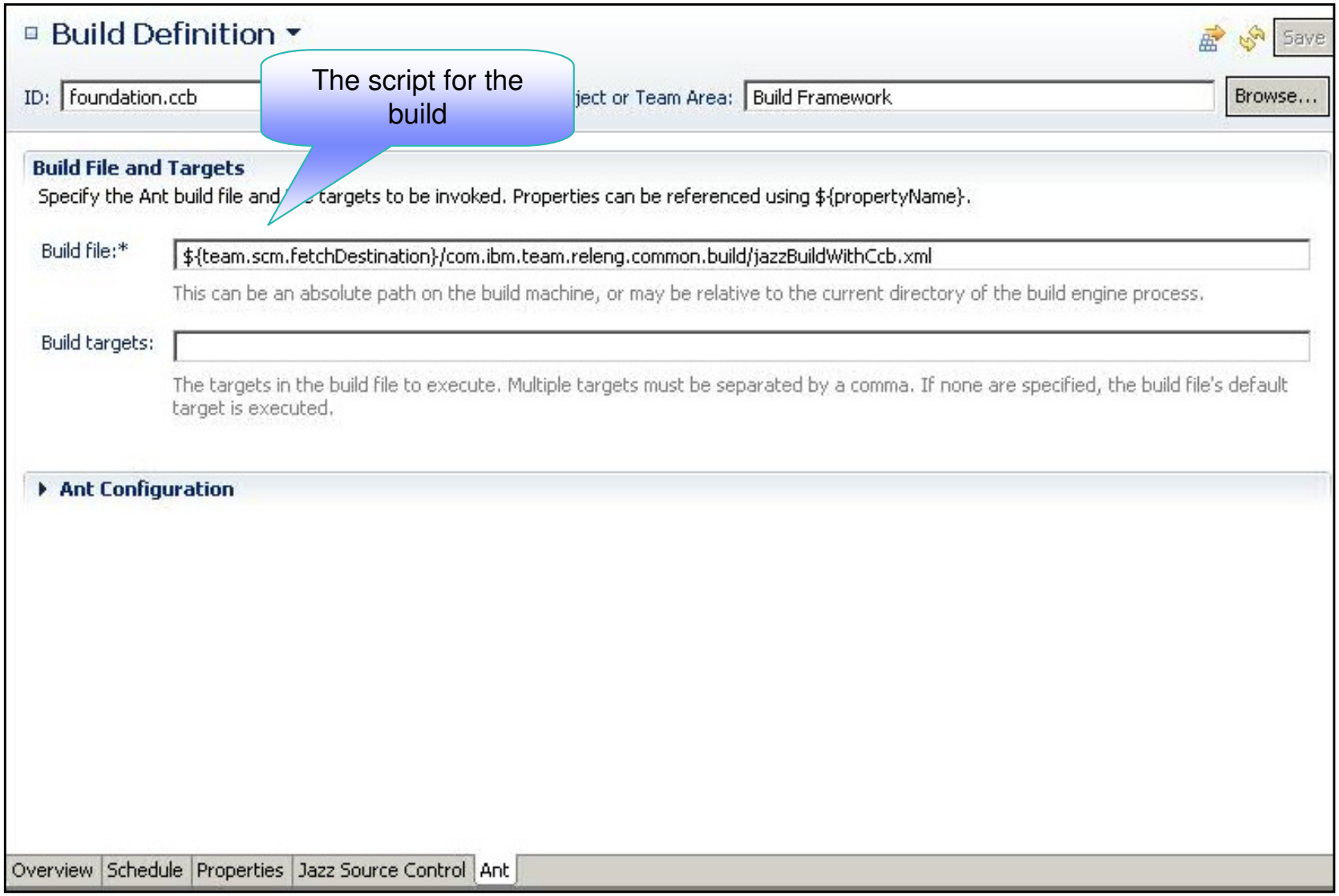
A sample Jazz build definition -- scheduling



A sample Jazz build definition – source control



A sample Jazz build definition – build script



Running the builds

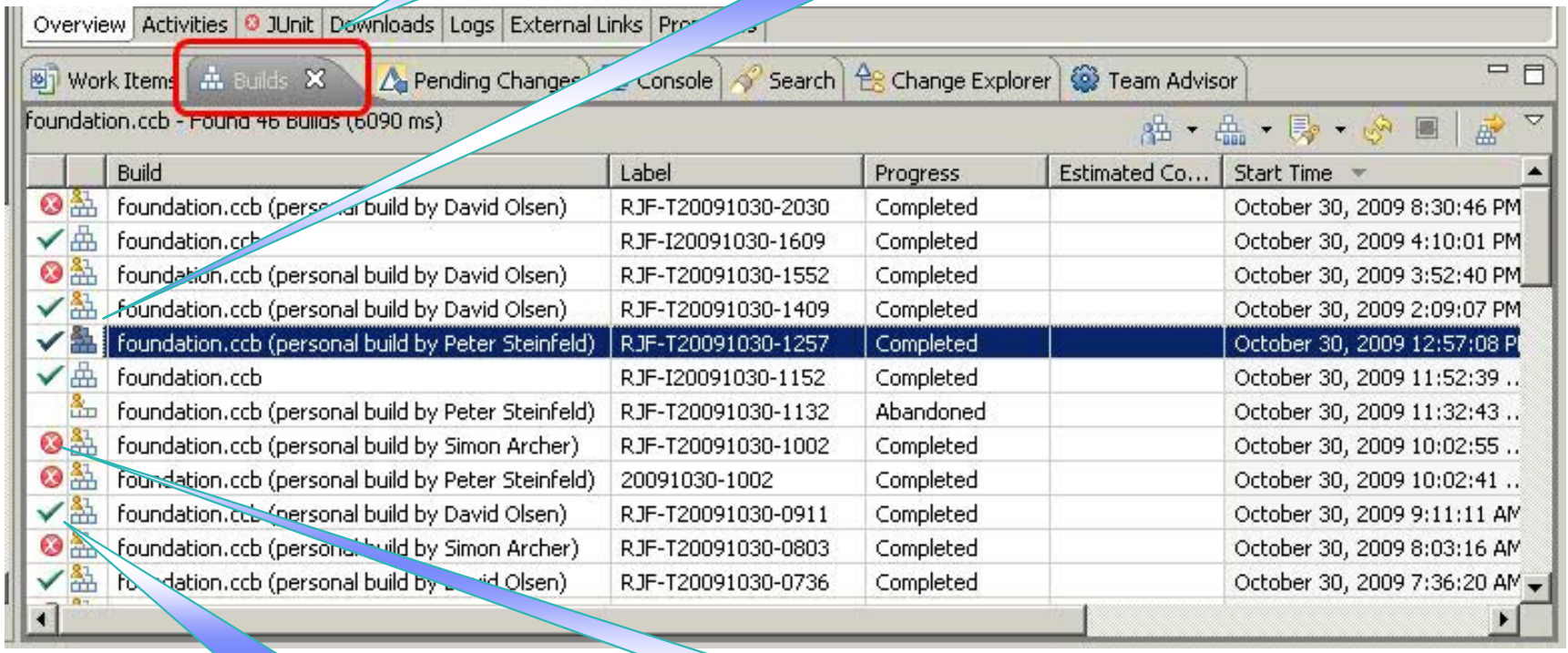
- Builds are run automatically or can be submitted on demand.
  - Continuous automatic builds only get run if new deliveries have been made since the last build.
- Each build run produces a build result
  - The build result has a clear pass/fail status
  - Build result contains log files
  - Build results contains links to artifacts produced by the build
  - Build results report test results

Build results are universally visible

Sample Jazz build results view

Build results view

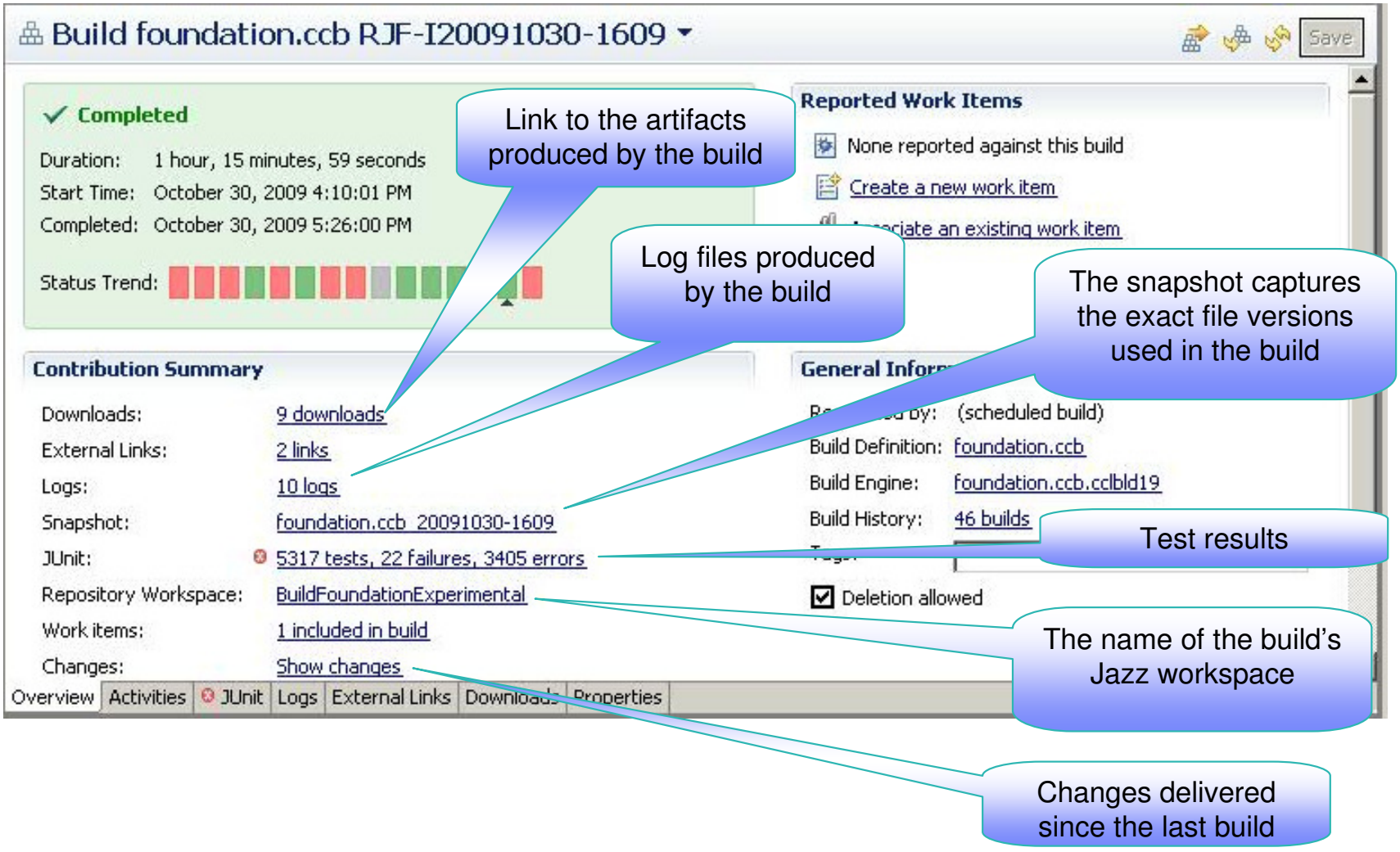
The build result for the highlighted build is displayed in its own view



A check mark means the build succeeded

A red “X” means the build failed

A sample Jazz build result



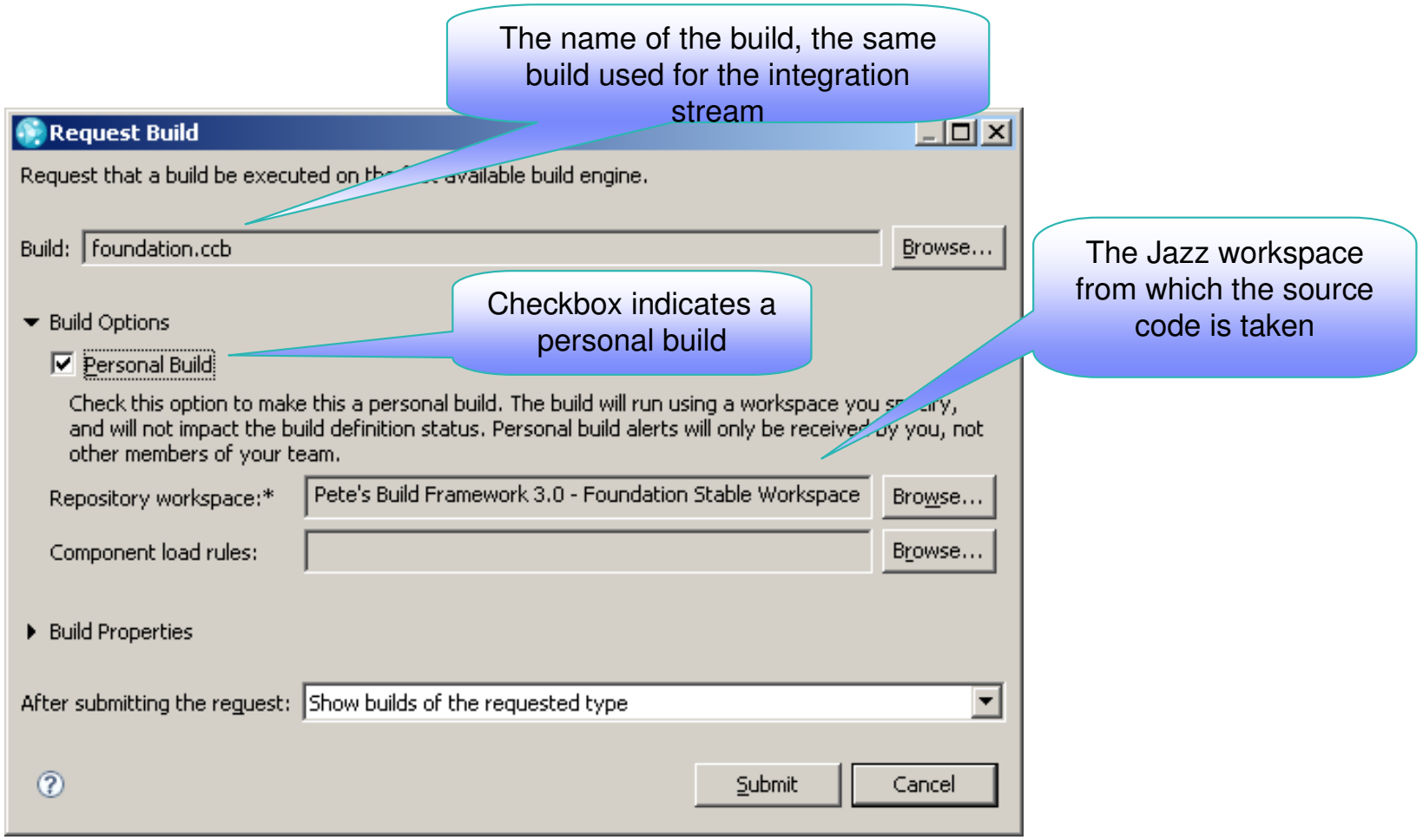
What happens during a build

- A build request is submitted, either on schedule or by user request
- For scheduled build requests with no newly delivered changes, nothing happens
- The build machines poll the Jazz server looking for build requests
- The build request gets matched to a build machine
- Changes are accepted from the integration stream to the workspace of the build
- A snapshot is created of the build's workspace
- The build machine fetches files using the versions in the snapshot
- The build script runs on the build machine
- Results are reported to the Jazz build result
  - Log files
  - Test results
  - Created artifacts

Personal builds

- Allow developers to do a test build of the contents of their Jazz workspace before delivering to the integration stream
- Similar to integration builds
  - Personal builds use the same build definition as the integration build
  - Personal builds run on the regular build machines
  - Personal builds use the regular build scripts
- But they're different …
  - Personal builds take input from a developer's Jazz workspace rather than from the build's workspace (the integration stream)
  - Personal builds do not create snapshots

Requesting a personal build



Daily project rhythm

- ✓ Developers deliver changes to the build
- ✓ Builds run and report results
- ✓ Broken builds get fixed immediately
- ✓ Everyone watches the builds for status
- ✓ System test engineers pick up test artifacts from the Downloads link on the build results page

Foolproof delivery process

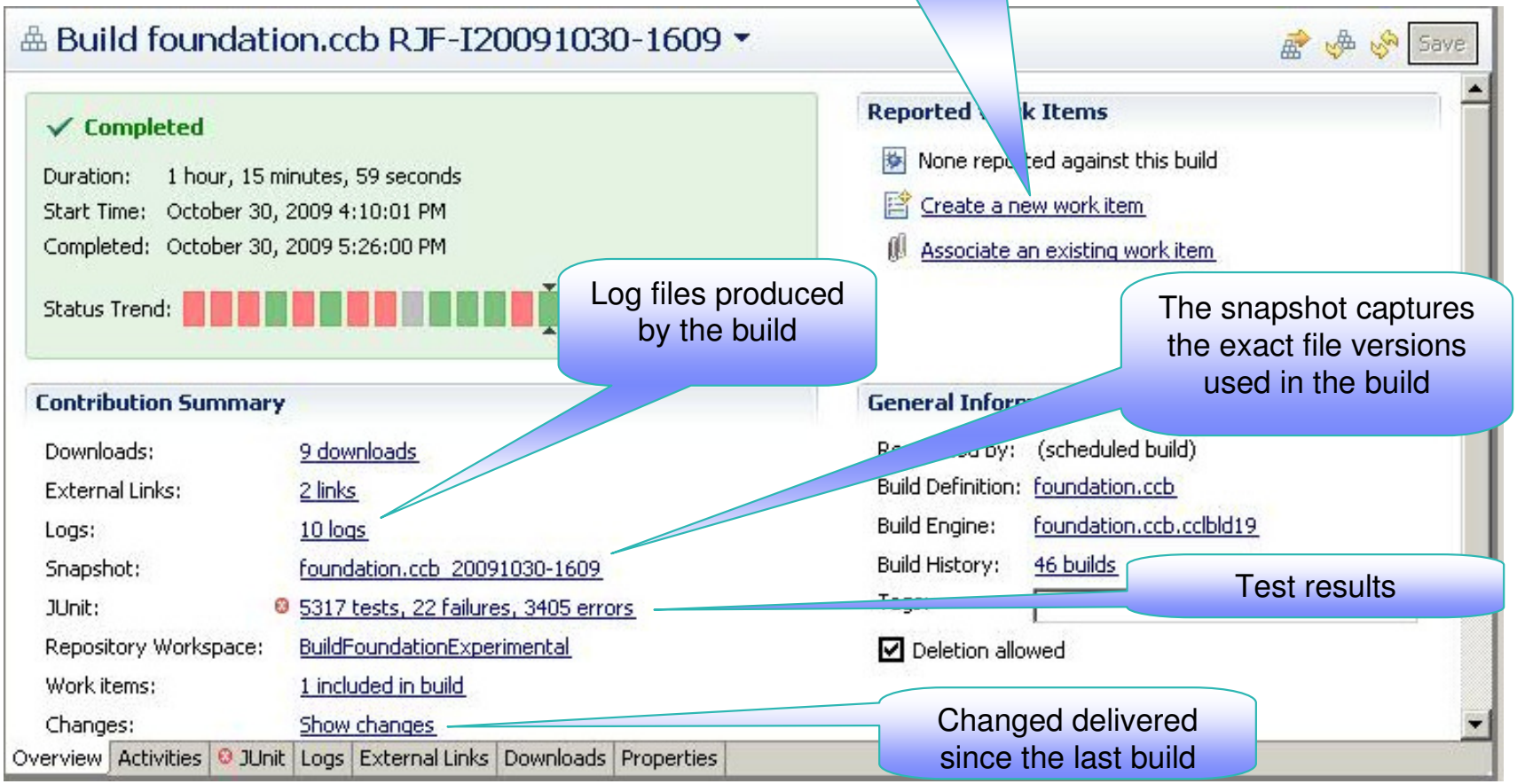
Developer deliveries must not break the build

- ✓ Jane, our developer, accepts from the integration stream into her Jazz workspace – now her Jazz workspace matches the integration stream
- ✓ Jane develops new code and automated tests in her own Jazz workspace
- ✓ Meanwhile, other developers deliver changes to the integration stream
- ✓ When Jane is ready to deliver, she accepts changes from the integration stream into her Jazz workspace *again* and does any necessary integration
- ✓ Jane does a personal build from the files in her Jazz workspace
- ✓ If the personal build succeeds, she delivers to the integration stream
- ✓ If the personal build fails, she continues working until she gets a successful build in a workspace that contains a mix of the current integration stream and Jane's changes

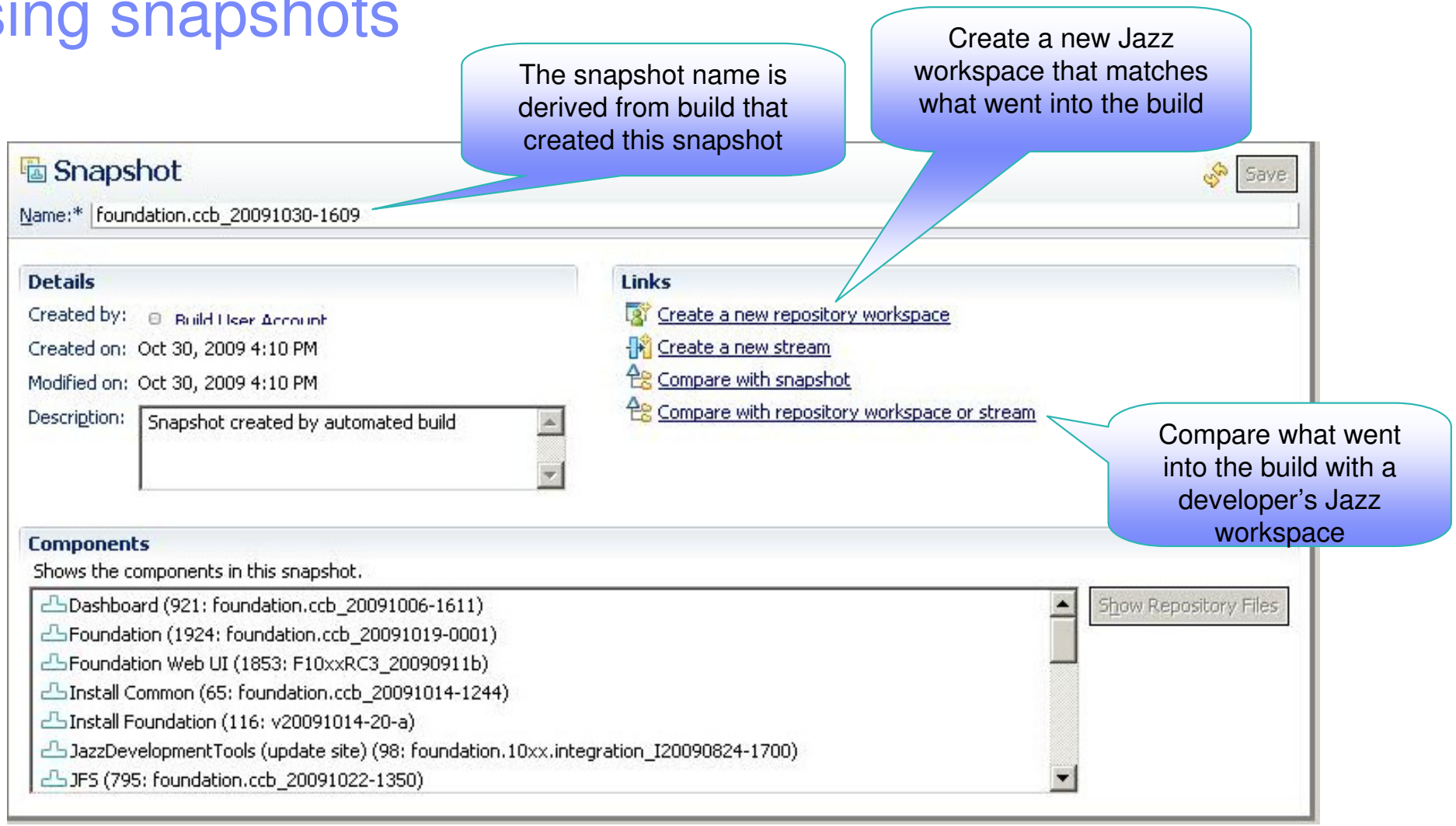
Fixing broken builds

- Create a work item
- Analyze the log files
- Analyze the test results
- Look at the changes delivered since the last build
- Use the build's snapshot to reproduce the build's contents on a workstation
  - Compare the snapshot to Jane's workspace
  - Create a Jazz workspace with the same contents as the snapshot to reproduce the problem

A sample Jazz build result



Using snapshots



The conversation you want to have …

Your manager	You
How's the project going?	*We have clean builds every day.*
What have you implemented so far?	*We've completed features A, B, and C, and are working on D and E.*
How do you know that A, B, and C are complete?	*We have automated tests that check them in every build.*
That's great! How can I try them out?	*Just grab them from latest build. Let's do it right now …*

References

- jazz.net -- https://jazz.net/index.jsp
- Jazz build FAQ -- https://jazz.net/wiki/bin/view/Main/BuildFAQ
- Personal builds -- http://jazz.net/wiki/bin/view/Main/BuildPersonal
- psteinfe@us.ibm.com

Questions?