Migrating to Rational Team Concert from SCLM and Other z/OS SCMs

Liam Doherty
IBM Corporation

Thursday August 15th, 2013
Session 13687
Agenda

- What is Rational Team Concert?
- Migration planning
  - Pre-migration
  - Migration
  - Post-migration
- Additional Considerations
- Additional Resources
IBM Rational Collaborative Lifecycle Management (CLM)

Best Practice Processes

- Product/Project Manager
- Analyst
- Developer
- Quality Professional
- Architect

- Architecture Management
- Asset Management
- Rational Requirements Composer
- Rational Team Concert
- Rational Quality Manager
- Build & Deploy Management

Robust extensible solution for the entire extended development team
Rational Team Concert (RTC): An open, extensible architecture

Supporting a broad range of desktop clients, IDEs and languages
Rational Team Concert Features

Work item tracking
Work items are the fundamental mechanism in Rational Team Concert to track and coordinate development tasks and workflows, all governed by your team’s associated process. In addition, work items are the hub for linkage between work items of different types or across projects.

Continuous builds
The Team Build component integrates the team’s build system into Rational Team Concert, providing build awareness, control, and traceability to the team. Team members can track build progress, view build alerts and results, request builds, and trace builds to other artifacts like change sets and work items.

Source control
The Source Control component of the Jazz platform integrates with Eclipse, Visual Studio, and other IDEs to develop code in an environment, integrated with Rational Team Concert.

Transparency/project health
The Team Reports and Web Dashboards components of Rational Team Concert help you keep tabs on the health of your project. Dashboard provides an at-a-glance view of work item queries, event feeds, reports, and other items that are critical to understanding your progress. Reports provide both real-time views and historical trends of builds, streams, work items, and other artifacts that your team works with.

Planning
The Planning component provides tools for both agile and waterfall planning. Users can balance the work, depending on priorities and plans are accessed and updated to reflect the team’s work.

Administration
The Jazz Team Server provides a web-based administrative UI for setup, configuration, and administration of Rational Team Concert.

Built-in Integrations
Rational Team Concert also integrates with other products, such as Rational Quality Manager, Rational ClearQuest and Rational ClearCase. The integration with Rational Quality Manager and Rational ClearQuest is part of the Collaborative Application Lifecycle Management (CLM) effort, which also includes Rational Requirements Composer. Collaboration tool integrations are also built-in for IBM Sametime, IBM Connections, and other popular tools.

https://jazz.net/projects/rational-team-concert/features/
# Rational Team Concert: An Overview

## Planning
- Integrated release/iteration planning
- Effort estimation & progress tracking taskboards
- Out of the box process templates: formal or agile

## Project Transparency
- Customizable web based dashboards
- Real time metrics and reports
- Project milestone tracking and status

## SCM
- Component based SCM enables reuse across projects
- Change set based for easy addition or removal of features
- Server-based sandboxes
- Can also work with SVN, Git, ClearCase or Synergy

## Work Items
- Defects, enhancements and conversations
- View and share query results
- Support for approvals and discussions
- Query editor interface
- ClearQuest or Synergy Bridge

## Build
- Automated work item and change set traceability
- Build definitions for team and personal builds
- Local or remote build servers
- Multi-level continuous integration
- Integration with Build Forge

## Jazz Team Server
- Single structure for project related artifacts
- World-class team on-boarding / offboarding including team membership, sub-teams and project inheritance
- Role-based operational control for flexible definition of process and capabilities
- Team advisor for defining / refining “rules” and enabling continuous improvement
- Process enactment and enforcement
- In-context collaboration enables team members to communicate in context of their work

---

**Planning**

- Integrated release/iteration planning
  - Effort estimation & progress tracking taskboards
  - Out of the box process templates: formal or agile

**Project Transparency**

- Customizable web based dashboards
  - Real time metrics and reports
  - Project milestone tracking and status

**SCM**

- Component based SCM enables reuse across projects
  - Change set based for easy addition or removal of features
  - Server-based sandboxes
  - Can also work with SVN, Git, ClearCase or Synergy

**Work Items**

- Defects, enhancements and conversations
  - View and share query results
  - Support for approvals and discussions
  - Query editor interface
  - ClearQuest or Synergy Bridge

**Build**

- Automated work item and change set traceability
  - Build definitions for team and personal builds
  - Local or remote build servers
  - Multi-level continuous integration
  - Integration with Build Forge

**Jazz Team Server**

- Single structure for project related artifacts
  - World-class team on-boarding / offboarding including team membership, sub-teams and project inheritance
  - Role-based operational control for flexible definition of process and capabilities
  - Team advisor for defining / refining “rules” and enabling continuous improvement
  - Process enactment and enforcement
  - In-context collaboration enables team members to communicate in context of their work

---

Complete your sessions evaluation online at SHARE.org/BostonEval
What is Rational team Concert?

• So RTC is more than just an Software Configuration Management system
  • Process, Planning and Work items coupled with an integrated SCM provide a complete solution
  • Ability to manage distributed and z/OS source in the same repository makes for a more integrated SCM solution
  • Migrating your existing SCM to RTC is only part of the job
  • Migration gives you the chance to review your current process to see how RTC or the full CLM solution can help integrate all your processes into a single tool
## Rational Team Concert terminology

| **Stream** | Collection of components used to organize work, coordinate collaboration and integration, and capture the active configuration of each component. Related to a level in a hierarchy (e.g., promotion levels, releases, etc) |
| **Component** | Collection of related artifacts (i.e., sourcefiles are logically organized into components) that have the same lifecycle. Used to control access rights, facilitate sharing and reuse. Theoretical limit: 50000 files. **Recommended: 1000 – 2000 files / component** |
| **Repository Workspace** | Workspace for 1 user synchronized with a Stream and the "Sandbox". Situated on the RTC server. |
| **Sandbox** | Workspace on the hard disk (e.g. local eclipse workspace). Note: Through the build or CLI you have jazz metadata but no eclipse metadata. For ISPF Client a Sandbox is a collection of data sets with the same HLQ.MLQ. |
| **Change Set** | Contains a collection of consistent changes made to a configuration of a component. Means for flowing file and folder changes between repository workspaces and streams. |
| **Work Item** | Captures the tasks and issues to be addressed by the team members. Associated with change sets created by the developer. Automatically and dynamically populate plans and reports. |
| **Baseline** | Non-editable version of a component capturing an interesting point in time. The baseline is performed implicitly when a Snapshot is taken. Can be done manually on a given component. |
| **Snapshot** | Collection taken of all component baselines for a stream or repository workspace capturing an interesting point in time. |
### Rational Team Concert terminology (cont)

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Load</strong></td>
<td>Action that copies selected files and folders from the repository workspace to the sandbox (eclipse workspace or MVS data sets)</td>
</tr>
<tr>
<td><strong>Accept</strong></td>
<td>Action that allows for synching the repository workspace reference with changes delivered to the stream by other developers. Load of the accepted changes into the sandbox is automatically performed. Note – you can also accept change sets from a WI</td>
</tr>
<tr>
<td><strong>Check-in</strong></td>
<td>Action that allows to save local changes into the repository workspace, within a Change Set</td>
</tr>
<tr>
<td><strong>Deliver</strong></td>
<td>Action to push the workspace changes from the workspace to the Stream</td>
</tr>
</tbody>
</table>
Migration planning

• The migration from a legacy mainframe SCM system to RTC involves several steps before you have an operational system.
  • Pre-migration
  • During the migration
  • Post-migration
Pre-migration

• Install RTC!
• Create project areas
• Define the structure of your streams
• Define the structure of components in your stream
• Define the delivery flow between streams
• Identify the nature of the programs and the main language definitions you will need for your IT system
• Create system definitions for data set definitions, language definitions and translators
• Set translator variables
Planning your Rational Team Concert solution

• Various aspects of your current workflow will influence your final stream strategy, for example:
  • *How do you plan to do version/release maintenance*
  • *How do you want to handle emergency fixes*
  • *Do you require different integration levels for different teams.*
• Keep it simple when you first start
  • *As teams work in RTC and get familiar with how the SCM works, you can define new streams that will support additional needs.*
Planning your Rational Team Concert solution

- Where are you going to host your server and repository?
  - The RTC server can run on a multitude of environments
    - Windows, Linux, AIX, Unix, zLinux, z/OS, IBMi
  - The repository database can be hosted on a multitude of environments
    - DB2 on LUW, DB2 on zLinux, DB2 on z/OS, Microsoft SQL Server, Oracle to name a few
  - The server can be run on one system with the data base on another
    - eg: Server running on zLinux and database running on DB2 on z/OS
  - You need to choose the best topology for the size and complexity of your implementation
  - What are your current server administration skills?
Define your projects areas

- **Project Area structure**
  - 1 Project Area per line of business or application
    - This depends on the team structure & relationships between applications of the same LOB
  - 1 Common Project Area
    - To pool the RTC setting
      - *Roles and Process, ..*
    - To pool the shared definitions
      - *System Definitions (Language Defs, Dataset Defs, etc.)*
      - *Build engines (?)*
    - Propagation by inheritance to other PA
    - Defines the stream that publishes common components & frameworks
  - Access control
    - *Read/write to Admins only*
    - *Read-only for all team members*
Define the structure of the streams

- The stream maps to an application for an environment
- A stream must be complete
  - That means that it contains all the dependencies needed by all the programs in the stream
  - Including common elements (framework)
  - Can include Components from another stream (from same or another Project Area)
Hierarchy and Integration

PreProd

Test

Dev

Complete your sessions evaluation online at SHARE.org/BostonEval
Evolution of the streams over time

Project Area

Stream Dev Team Appl1
Stream Dev Team Appl2
Stream Dev Team Appln

Stream Development version N + 1

Possibility of Automation

Build

Development Integration Quality Assurance

Load Dev Load Int Load QA

Timeline management

T0 T1

Complete your sessions evaluation online at SHARE.org/BostonEval
Evolution of the streams over time

Stream Dev Team Appl1
Stream Dev Team Appl2
Stream Dev Team Appln

Rename

Stream Dev Team Appl1
Stream Dev Team Appl2
Stream Dev team Appln
Components

• Which logical units make up the applications (components)?
  • Put related artifacts or projects together so components make sense from code reuse, application build operations and team sharing perspective

• What are the common source elements used across several applications/modules?
  • Define components to be reused across applications, so they can be maintained by certain teams, or to be shared for all teams within a project area.

• Your development teams will work on a set of components for which they are responsible.
  • When structuring the components along with architectural details bear also in mind the organizational structure that will support it.
The component

- Corresponds to a part of an application
  - Divided by a topology of component types
- Component is owned by a team
- Single Platform
  - Simple grouping criteria
  - Stream by Platform / Team
  - A lot of components if dealing with a complex system
- Multi-Platform
  - No de-synchronization between client and server
  - Forces the same lifecycle for all technologies
- Focus of attention for the copybooks
  - Copy for public interface
    - By public interface we mean …Copy used by an application to call another application modules
  - Copy framework (cross-cutting)
    - By framework we mean … copy such as authentication or security related, not owned by a particular application
Project Area, Stream and Component Structure

- Workflow for publishing & adopting shared components
Project Area, Stream and Component Structure

- Project Area set up

Publish of the new baseline of API-1

Notify teams of changes of API-1
Project Area, Stream and Component Structure

- Ownership of components
Create System Definitions

- **Data set definitions** *describe data sets* involved in the build process
  - E.g., a COBOL compiler data set definition contains the name of the actual compiler PDS and member

- **Translators** *define a single step* in the build process.
  - It’s in the translator that you specify that the build will perform a compilation, link-edit, etc.
    - E.g., a COBOL compilation translator contains a reference to the compiler data set definition, default compiler options, required DD concatenation and allocations, and a maximum successful return code

- **Language definitions** *order the steps* in the build process
  - E.g., a language definition for a main program contains references first to the COBOL compilation translator and second to the link-edit translator
Translator comparison to JCL using data set definitions

<table>
<thead>
<tr>
<th>JCL Line</th>
<th>Corresponding data set definition name</th>
</tr>
</thead>
<tbody>
<tr>
<td>COBOL    EXEC PGM=IGYCRCTL,REGION=2048K;</td>
<td>COBOL Compiler</td>
</tr>
<tr>
<td>XX       PARM=(EXIT(ADEXIT(ELAXMGUX)),</td>
<td>No DSD</td>
</tr>
<tr>
<td>XX       'ADATA',</td>
<td></td>
</tr>
<tr>
<td>XX       'LIB',</td>
<td></td>
</tr>
<tr>
<td>XX       'TEST(NONE,SYM,SEP)',</td>
<td></td>
</tr>
<tr>
<td>XX       'LIST',</td>
<td></td>
</tr>
<tr>
<td>XX       'FLAG(LI)&amp;CICS&amp;DB2&amp;COMP)</td>
<td></td>
</tr>
<tr>
<td>XX       STEPLIB DD DISP=SHR,</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>XX       SYSLIB DD DISP=SHR,</td>
<td>Copybooks</td>
</tr>
<tr>
<td>DSN=F057699.TEST.RTC.COBOL</td>
<td></td>
</tr>
<tr>
<td>COBOL    SYSLIN DD DISP=SHR,</td>
<td>Temporary file (object deck)</td>
</tr>
<tr>
<td>// DSN=F057699.TEST.RTC.COBOL(EPSCMORT)</td>
<td></td>
</tr>
<tr>
<td>//COBOL   SYSLIN DD DISP=SHR,</td>
<td>Temporary file</td>
</tr>
<tr>
<td>// DSN=&amp;&amp;OBJ</td>
<td></td>
</tr>
<tr>
<td>//         {},SPACE=(TRK,(3,3)),</td>
<td></td>
</tr>
<tr>
<td>//         UNIT=SYSDA, DISP=(NEW,PASS)</td>
<td></td>
</tr>
<tr>
<td>//         DCB=(RECFM=FB,LRECL=256,BLKSIZE=2560)</td>
<td></td>
</tr>
<tr>
<td>SYSUT1    DD UNIT=SYSALLODA,SPACE=(CYL(1,1))</td>
<td></td>
</tr>
<tr>
<td>SYSUT2    DD UNIT=SYSALLODA,SPACE=(CYL(1,1))</td>
<td></td>
</tr>
<tr>
<td>SYSUT3    DD UNIT=SYSALLODA,SPACE=(CYL(1,1))</td>
<td></td>
</tr>
<tr>
<td>SYSUT4    DD UNIT=SYSALLODA,SPACE=(CYL(1,1))</td>
<td></td>
</tr>
<tr>
<td>SYSUT5    DD UNIT=SYSALLODA,SPACE=(CYL(1,1))</td>
<td></td>
</tr>
<tr>
<td>SYSUT6    DD UNIT=SYSALLODA,SPACE=(CYL(1,1))</td>
<td></td>
</tr>
<tr>
<td>SYSUT7    DD UNIT=SYSALLODA,SPACE=(CYL(1,1))</td>
<td></td>
</tr>
</tbody>
</table>

Complete your sessions evaluation online at SHARE.org/BostonEval
Translator Variables

- Variable overrides

[Image of a configuration screen with options for data set definition, default options, DD names list, and an example of defining variables with a list of values for a specific file.]
Migration

• Define the baselines that you want to import so you can capture history in the RTC SCM
• Import a baseline from the legacy system and dispatching them to the right component or project
  • Methods of migration, zimport, ISPF Client
• Iterate to capture the needed history till the current version
• Initially migrate a subset of modules that cover all the different types of source code
  • Use this subset to test your builds to make sure language definitions are correctly defined and that everything that needs to be built actually does
Migrating your source code to Rational Team Concert

- **RTC provides an import utility called zimport**
  - The *zimport* SCM command line tool (aka “mass import tool”) imports your PDS members directly into the repository
    - Automatically creates the proper zComponent project structure
    - Automatically creates a data set definition based on characteristics of data set on host
    - Automatically (optionally) associates language definitions with each member
  - You can build a source code version history of your major releases by running a series of zimports with the same repository workspace
- **You can also use the RTC ISPF Client to import a new PDS**
Migrating your source code to Rational Team Concert

- **zimport preparation of data**
  - By Component
    - Separate out by type into a PDS
      - *Cobol*
      - *Cobol/DB2*
      - *Assembler*
    - Line numbers if they exist must be stripped before import. If they are getting re-gen in PDS – also it look cleaner
      - *Cobol*
        - *72-80 can leave if there are comments you want to keep*
        - *1-7*
        - *Others?*
  - *zimport* will scan the entire catalog looking for the datasets you define
    - Make them a unique HLQ - Userid as HLQ for example
  - **Gotchas**
    - Zimport will try to recall dataset from HSM. When it scans the catalog and does not find the dataset - It will fail over and over
    - Line numbers will cause RTC merge to fail
Migrating your source code to Rational Team Concert

• What to import?
  • All source, recommendation is that the production baseline versions are imported
    • Cobol, PL/I, JCL, Procs, etc
    • Adding in all versions will be very costly and time consuming
      • If the IBM services team is engaged they have additional tools to help
        • For example - A procedure has been developed to off load older versions that can be viewed through ISPF when necessary
  • SCLM Language definitions, Endevor processors or Changeman skeletons need to be converted to RTC definitions
    • Language Definitions
    • Translators
Migrating your source code to Rational Team Concert

• Once zimport has been complete you can set up the rest of your system definitions
  • Create any additional Data set definitions
    • zimport will have created data set definitions for “inputs”
      • COBOL, PL/I, ASM, JCL
    • Use RTC dialogs to create data set definitions for “outputs”
      • OBJ, DBRM, LOAD
    • Also create data set definitions for temporary files
Migrating your source code to Rational Team Concert

- Once you have all your translators and language definitions defined, you can assign them to the relevant files if you didn’t do that in zimport
Post-migration

- Migrate your builds
- Tune the system definitions by running builds on a representative subset of your applications
  - You need to ensure all your code can be built and your language definitions are correctly defined
- Prepare the system to trigger dependency builds that involve only the new developments in RTC:
  - Run combination of builds and simulation builds to create the build maps
- Populate your stream hierarchy
- Populate the build maps in your stream hierarchy
Migrating your build to Rational Team Concert

- RTC supports capabilities for the operations of building, promoting your code, packaging and deploying it, as such;
  - You will need to understand the current build process of your applications: what are different technologies in use for building your applications and how you build them.
  - “Stages” of your source code and where do you build applications, just at DEV or at various levels of the hierarchy
  - How are your applications deployed, with all the details of your target deployment locations and your runtime locations
Migrating your build to Rational Team Concert

- **Build Definition**
  - Contains the build characteristics
  - Repository workspace that flows to team stream containing the source code
    - Repository workspace must be readable by the build user
  - What do I want to build? Whole repository workspace or subset of programs
  - Language definitions to be built
  - Sandbox location

- **Build Engine**
  - RTC representation of a process running on a build machine that executes build requests

- **Build Agent**
  - Executes the build
  - Located on z/OS (for mainframe)
  - Accesses RTC to retrieve source code and other information

- **Build request and build result**
  - Representations of the request to run a build and the output from the build run
RTC z/OS builds: How it all hangs together

- **Build Workspace**: Tasks run on the host, such as compilation. A build can handle several different tasks in the order shown.

- **PDS HLQ**: Corresponds to a STEP in the process to run on the host.

- **USS Load Directory**: Contributed to the step to execute. Corresponds to programs/files/PDS lines used by EXEC, DD, SYSIN,… as in JCL.

- **Build Definition**: Allows you to automatically associate a behavior with a type of file in the RTC repository.

- **Language definition**: Corresponds to programs/files/PDS lines used by EXEC, DD, SYSIN,… as in JCL.

- **File Extension**

- **Translator**

- **DataStream (flow)**

- **Build Engine**

- **Component**
  - zProject
  - zOSSRC
  - zFolder
  - zFile1
  - zFile2

- **ATTENTION**: The name of the directory on USS must be unique to each build definition.
Simulation Build

- When you initially migrate you may not want to rebuild all your source
  - This is a time consuming task
  - If you rebuild you really need to retest
- You only want to build things that have changed since your migration
- Simulation build will go through your code and create/update build maps so that the code you migrate looks current
- Any subsequent changes will then force a rebuild of just the changed modules

New in RTC 4.0.4
Additional Considerations

• Security
  • https://jazz.net/wiki/bin/view/Main/ZosBuildAgentSec
  • https://jazz.net/wiki/bin/view/Main/DependencyBuildScenarioOpenSSLSetup

• ISPF Client set up
  • https://jazz.net/help-dev/clm/topic/com.ibm.jazz.install.doc/topics/c_client_ispf_installation.html

• Promotion

• Deployment

• RDz Integration
Additional Resources

- Jazz.net
  - [https://jazz.net/library/](https://jazz.net/library/)
    - Articles, videos, tips, documentation, and more
  - [https://jazz.net/library/#type=video&project=rational-team-concert](https://jazz.net/library/#type=video&project=rational-team-concert)
    - Videos on various RTC features. Just search for keywords

- zimport additional resources

- Developerworks resources on migration